OpenVMS Backup Theory and Practice

John Gillings
Software Systems Consultant, OpenVMS Ambassador

1.0 introduction

Everyone knows that computer system backups are very important. However, for some, the act of backing up a system is little more than a voodoo ritual in which poorly understood incantations are recited over data. Yes, the tape drive spins around, and something is written, but it is not necessarily a useable backup of the system. Unfortunately, a backup strategy is usually discovered to be inadequate at precisely the wrong time: when trying to restore a system after a failure.

As businesses become more and more dependent on their computer systems, the inability to recover from data loss can result in a complete business failure. Purveyors of tape systems can tell you the percentage of businesses that go broke within a year after losing their data. To prevent such disasters, it is vital that backups be integrated into regular system maintenance. Backups must also be performed in a manner that guarantees the complete restoration of data.

2. What is a backup?

2.1 *Definition*

In this discussion, a backup is defined as a *strategy* for collecting *sufficient information* about *the state of a system* so that the system can be *restored* at some time in the *future*. Italics indicate key elements in the overall concept of **restoring the state of the system**. In particular, there is nothing in the definition about the "contents of disks." An important misunderstanding about backups is the result of focusing on disks and tapes and failing to consider the *system as a whole*. The contents of disk drives are certainly an important part of the state of the system, but so are the contents of main memory and of caches or buffers, including controller caches. Even the state of applications programs that are capable of modifying memory or disks are important; in other words, anything that affects the current operation of the system is important. Furthermore, notice that the emphasis is on a **strategy** rather than a specific command or procedure. Finally, notice that the objective is to collect **sufficient information**, not necessarily *everything*.

2.2 Key points

By keeping 3 key points in mind, you can ensure that your backup strategy is both reliable and workable:

- **Restore the state of the system**
- **Use a strategy, not a command**
- **Collect sufficient information**

An additional benefit of knowing your system better will be your ability to minimize the resources and effort required to perform backups.

3. Why do you need backups?

3.1 In the real world, things go wrong…

In an ideal world, backups would be unnecessary. Computer systems would work exactly as intended all the time. Unfortunately, computer systems have no concept of the ideal world, and Murphy's Law seems to affect the entire computer industry: if something can go wrong, it will. Think of a computer system as a machine in a correct or an incorrect state. Enormous numbers of potential system states exist, the vast majority of which are incorrect. Also, a frightening variety of events can cause the state of a system to transition from a correct to incorrect. Some transitions are easily reversible. Others are not. Good backups will allow you to restore your system to a correct state under all circumstances.

3.2 What sort of things?

Some examples of potentially harmful events are hardware failures, application errors, operating system bugs (which do happen, sometimes), power failures, fires (even in someone else's building), floods, and, of course, human errors (such as entering **DELETE *.*;*** in the wrong directory or asking innocently, "What's this button do?").

3.3 Backups are data insurance

You can think of a backup strategy as a form of insurance policy: you pay premiums in the form of capital equipment (extra disk drives, tape drives, tapes, and so on), storage costs, labor costs, and CPU time. You collect by having your system restored after a disaster. You must read the fine print to understand what events your policy does and does not cover.

Besides these basic elements are the "excess" (that is, how much it costs to make a claim) and the "trade-off" between the premium and the level of coverage -- in other words, are you paying too much for what you are getting?

4. formulating a backup strategy

Just as there is no such thing as a universal insurance policy, no single, universal backup strategy is appropriate for everyone. If there were, everyone would run HP Universal Backup for OpenVMS/VAX/Alpha™/Itanium®, and there would be no need for further discussion. Backup strategies, however, must be tailored to a specific site and business model. By understanding your specific circumstances, you can formulate your backup strategy to maximize coverage and minimize costs.

Be sure to document your strateg fully, including detailed descriptions of daily, weekly, monthly and annual procedures, and various levels of recovery procedures: single database, single disk, single node, whole system, and full disaster recovery. A strategy must go beyond what you do to back up data from disks. Your backup strategy should be an integral part of your operations beginning with your initial system planning.

4.1 Know your risks
First and foremost, you need a realistic estimate of what your data is worth. Computer staff can rarely answer this question. You need to ask your accountants and executives, "If we lost this data, how much would it cost to replace it or to continue operations without it?" Once you know the value of what you are insuring, you can realistically judge the cost of the "premiums" required to cover it. You might also need to know the cost per unit time of missing data so that you can evaluate the cost of restoring the state of a system after a failure. You might be able to reduce the cost of the "premiums" by taking a higher "excess" (that is, a longer time to recover).

Note that systems operations staff should not make these kinds of policy decisions. Their jobs are to provide information to the policy-makers, who can then make an informed choice. Once a policy decision has been made, you need to implement it. Also, be aware that costs can be deceptive. For example, a DLT drive is relatively inexpensive, but to pay an operator overtime to feed it tapes after hours can be very expensive. A high-capacity tape library that can operate unattended might initially be more expensive but, over the long term, might be much cheaper -- and more reliable.

Finally, keep in mind that one person's trash is another person's treasure. For example, most sites would not suffer too greatly if they lost their ACCOUNTNG.DAT file. However, if your system is a bureau service, and the contents of that file are used to calculate customers' bills, then losing it could be financially catastrophic.

4.2 Know your data
Conceptually, the simplest and most reliable type of backup is a standalone snapshot of everything. All disks are saved as full image copies. If you can afford the down time, this method provides you with the fastest and most reliable recovery path. You just lay down all the disks and reboot the system. However, this strategy is rarely acceptable. It is far too labor-intensive because it cannot really be automated (although some console management products come close to automation). Also, with the large size of disk farms and disk drives today, a standalone snapshot takes far too long, even using the fastest available tape drive technology.

By understanding your data, you can reduce the amount of information that you need to capture to restore the system to a correct state. Even if you have a 50GB database, it is highly unlikely that all the data changes every day. Therefore, if you can save just the changes since the last full backup, you might be able to dramatically reduce the time spent performing backups. The trade-off is that restoring the system state will take longer because the changes need to be applied to the database after restoring the last full backups.

Other approaches include segregating different classes of data. For example, if you can keep all read-only data on a single physical disk, you might not need to back it up at all. You could burn truly read-only data onto a CD or DVD -- perhaps even making multiple copies. The CD could then either be read directly or be kept as a backup that could, in emergencies, be read directly.

4.3 Test your strategy

Testing is the part of a backup strategy that is most often overlooked. Without testing, you cannot know with certainty that your recovery will work as expected at a critical moment. The worst possible time to discover a fatal flaw in a backup strategy is after you suffer a failure. Unfortunately, that is precisely the time such flaws are often discovered.

To test your strategy, set aside a weekend when the system can be shut down, or arrange to use some test hardware that is as similar as possible to your production system. Make the recovery as realistic as possible. Your goal is to bring up a production system using your written disaster recovery plan. Time the operation, and document any problems. Then use the results to review your plans.

Should you suffer a real failure, hold a post mortem of the recovery procedure to review how it worked in practice and what improvements could be made.

5. How should you back up your system?

Backing up a system requires recording an instantaneous, known system state. The problem is that the state of any system can change within microseconds, yet recording the state can take hours. Unless a system can be placed in a special state that prevents unwanted changes (such as standalone backup), steps must be taken so that the backup mechanism ignores state changes that occur after the target "known state." This technique is known as **checkpointing**. The application code draws a conceptual line, and only changes that occurred before that point are included in the backup.

Because checkpointing is application-specific, generic system tools cannot perform online backups of live data without the cooperation of application logic. Unfortunately, backups are often perceived as a system-related, and application designers do not integrate backup functions into their applications. In reality, the opposite is true. Backups are fundamentally application-related. Operating systems must provide tools to assist in capturing information for backup purposes, but applications must decide which information should be captured. If you cannot afford down-time for off-line backups, your application code must build in the ability to make live backups because no "wands" exist to do it for you magically.

5.1 What tools are available to help you?

The following sections outline tools that you might find useful in performing backups.

5.1.1 Database management systems
Database systems such as Oracle, Oracle/RDB and Ingres interact with application code at a transaction level, with built-in support for checkpoints. Although the application must declare the checkpoints, the DBMS does most of the hard work. Details are beyond the scope of this talk, however.

5.1.2 RMS Journalling

If your applications use RMS files, you can use RMS Journalling to assist in backups. The application code, however, must still declare the checkpoints explicitly or implicitly. See the RMS Journalling manual for details of what it can do for you and how you can use it.

5.1.3 The Backup utility
Although BACKUP is very useful in performing backups, it is really just a tool for moving bits from one place to another. It has no built-in magic: it simply copies the files you tell it to copy. If they are the right files, well and good. If not, the GIGO principle applies. Read the BACKUP documentation carefully to understand all its features.

5.1.4 The DIRECTORY command
The DCL command DIRECTORY is not really a tool for performing backups, but, rather, helps you analyze the dynamic of your data. A simple command such as **DIRECTORY/MODIFIED/SINCE=date disk:[000000...]** often reveals just how little data has changed on a particular disk.

5.1.5 Spare disks
With the availability of cheap, large-capacity SCSI disks, having a few spare disks can be every cost-effective in your backup strategy. With spare disks, you have much more flexibility in both saving and restoring information. Calculate the hours of your time a 36GB SCSI disk is worth. Also consider the time for an engineer to deliver a replacement disk. If a failing disk can be replaced immediately, you might save yourself much more than the cost of the drive.

5.1.6 Appropriate hardware
Tape drives are constantly becoming smaller, faster, and higher-capacity. Although having the latest technology might not always be necessary, making sure that your drive or drives can handle the loads you subject them to is very important.

Far too often a tape drive is added to a system as an afterthought. When designing your system, think about how long a full backup will take, keeping in mind that a storage shelf full of disk drives can contain a lot of data. Even with the fastest tape drives available, a full backup can take several days. In calculating system costs, make sure you include the price of backups.

5.1.7 Archive/Backup System for OpenVMS (ABS)
If you have a large system, ABS can help automate your backup strategies, control your tape devices, and manage your media. For example, ABS can quickly locate which tape contains a specific file.

5.2 Techniques that do not work

A primary motivation for writing this article has been to reveal common practices that some people mistakenly believe will give them a valid online backup copy of a disk. True, you might get a useable copy of a disk, but then again, you might not. Using one of these practices is a bit like buying a cheap insurance policy. When you try to make a claim, will it be honored? Would you bet your business on it?

The following sections describe some common backup techniques that do not work.

5.2.1 Using the /IGNORE=INTERLOCK qualifier

The /IGNORE=INTERLOCK qualifier has a single purpose: to change the severity of a "file access conflict" error from ERROR to WARNING. As its name implies, the /IGNORE=INTERLOCK qualifier causes the system to bypass file system locks that protect the integrity of the file. The only certainties about a file backed up with an ACCONFLICT warning is that the file existed and that is was approximately the same size as the file in the backup. You might get a useable copy of the file, but you also might get only garbage.

The irony of using /IGNORE=INTERLOCK is that often those files that generate warnings are just the ones that need to be backed up, because they are the only files that are being actively modified.

5.2.2 Breaking shadow sets

Another poor backup technique is to break a physical member out of a shadow set, take an image backup of it, and then put it back into the shadow set. This is a very poor practice for two reasons.

- First, for the same reason that the /IGNORE=INTERLOCK command is not reliable, no guarantee exists that files open when the shadow set was dissolved are in a consistent state. Since OpenVMS Version 7.3 (or Version 7.2 with the latest shadowing ECOs), dismounting a shadow set member guarantees integrity at the disk structure level but does not necessarily guarantee the integrity of files that were open at that time. Prior to OpenVMS Version 7.3, no guarantees existed for the accurate contents of a removed shadow set member.
- Second, if you reduce a shadow set to a single member, you risk having latent bad blocks on both disks. Keeping two members in a shadow set at all times means that a bad block will affect you only if the same LBN goes bad on both members simultaneously. The chance of this happening is vanishingly small. Remember that bad blocks are detected only when they are read; therefore, a bad block can remain latent for a long time. When the block is read and detected, shadowing software tries to recover the data from the other member. If the shadow set is dissolved, the bad blocks on the remaining member are "exposed" because a recoverable backup copy no longer exists. Similarly, BACKUP copies (and marks as bad) any bad blocks on the removed member. These bad blocks are exposed if the backup is ever restored.

5.2.3 Having single-member shadow sets

Another poor practice is to make all disks single-member shadow sets. The theory behind this is that if a disk starts to show errors, a second member can be added to the shadow set and allowed to copy up. Then the failing disk can be removed. Although it is difficult to find fault with this logic, the practice rarely works. For a start, shadowing software copies corrupt data the same way it copies normal data. Also, bombarding a sick disk with lots of I/Os might not be a very good idea. If the disk is bad enough, shadowing software might not wait for the full copy to complete before kicking it out of the shadow set. You can imagine the results of that scenario -- not a pretty picture.

Single-member shadow sets are sometimes used to add a member, let it copy, and then break it out to make the backup. As described earlier, this produces unreliable results.

Having a single member shadow set is very nearly pointless. As many of you are painfully aware, shadowing software is notoriously complex and sometimes causes problems. Single-member shadow sets enjoy none of the benefits of shadowing and are exposed to the inherent risks of complexity as well. Because physical spindles are probably the cheapest component in shadowing, you are almost certainly better off building a two-member set.

5.2.4 Substituting RAID for proper backups
RAID controllers make storage management simpler and much more flexible. Different types of RAID sets can improve performance or data redundancy, or both, but they do not eliminate the need for proper backups. You should, however, take advantage of the features of RAID sets in your backup strategy to make backups faster and cheaper to perform.

5.3 Techniques that do work

Enough gloom and doom. Ways exist to perform backups of live (or, nearly live) data safely. At worst, you can minimize down-time to a few minutes. Explanations of some of these techniques follow.

5.3.1 Using CONVERT/SHARE on simple indexed files
A simple indexed file is one in which records do not depend on each other or on records in other files. If you can accept the limitation that an update to a record is either saved or not saved (that is, no partial updates are saved), then you can use the CONVERT/SHARE command to make an accurate copy of an open RMS indexed file. This technique assumes that the application code opens the file for shared access. Consider the SYSUAF.DAT file. If the system manager is resetting user passwords while the CONVERT operation is taking place, the backup copy will or will not reflect a particular change, depending on the exact timing of the CONVERT operation. According to RMS ordering rules, if a particular change is not copied, subsequent changes are also not copied.

Next consider the somewhat more complex case of adding a new user, which involves adding a record to the SYSUAF.DAT file and adding a "logically linked" record to the RIGHTSLIST.DAT file. If a job is performing CONVERTs on both files at the time, you might get both changes, neither change, or one of the changes, depending on the order of the CONVERT operations. In this situation, the "database" as a whole (consisting of 2 files) might be inconsistent because you cannot be sure that the backup procedure will treat both changes atomically. Depending on the type of database, this behavior might be acceptable. In this particular case, you could back up first the RIGHTSLIST.DAT file and then the SYSUAF.DAT file. In that way, you could get both updates, neither update, or only the UAF entry. (You cannot get only the rightslist entry.) You can easily automate a procedure that ensures a valid rights identifier for every username by using the MCR AUTHORIZE ADD/IDENTIFIER/USER=* command. Therefore, using the CONVERT/SHARE command might be an acceptable way for you to back up your authorization information online.

5.3.2 Using shadowing correctly

A method exists for safely using shadowing to achieve a nearly online backup. This method requires a minimum of application cooperation. The procedure is as follows:

1. Add a spare disk to a 2-member shadow set.
2. Allow the full shadow copy to complete.
3. Shut down all applications that use this disk (closing all files).
4. Dismount the third member.
5. Restart applications.
6. Back up the spare disk at your leisure.

This method avoids the problem of bad blocks because each new member is guaranteed to be free of bad blocks immediately after the copy completes. Also, at least one good copy of each block is on the original two members. Because you shut down the application code (closing all files), the files are all in a consistent state.

Depending on the application, your total downtime might be just a few minutes. The costs include the spare disk (recall my saying they'd come in handy) and the time to perform the shadow copy.

5.4 A special case - the system disk/system files

Your system disk is a special case. First, the only supported way to make a full backup of any system disk is standalone with the /IMAGE qualifier. This method is not negotiable. Second, the majority of information on the disk is read-only and is readily available from distribution media. Finally, nearly all the important system disk files that change are always open: SYSUAF.DAT, RIGHTSLIST.DAT, QMAN$*, other "cluster personality" files, various log files and journals, and so on. This means that those of you who regulary perform a nightly $BACKUP/IMAGE/IGNORE=INTERLOCK SYS$SYSDEVICE: tape:SYSTEM.BCK/SAVE are basically wasting your time and tape. The files that are being backed up do not need to be (How many copies of HELPLIB.HLB do you really need?), and the files that do need to be backed up, are not. Note that the same arguments apply no matter where your cluster files reside.

Again, no universal solution to this problem exists. However, a rough outline for a backup strategy for your system disk might look like the following:

- Perform image backups immediately before and immediately after upgrades. (You may also want to do an image restore to tidy up after the upgrade.)
- Copy highly volatile files to a directory on another disk nightly using the CONVERT/SHARE command. These files include SYSUAF.DAT, RIGHTSLIST.DAT, VMSMAIL_PROFILE.DATA and VMS$PASSWORD_HISTORY.DATA.
- On the other disk, back up the directory that contains the data. This directory can also be considered tocontain a "hot standby" copy of the data, ready for immediate use.
- Using CONVERT/SHARE weekly or as required, copy less volatile files such as SYSALF.DAT,

NETPROXY.DAT, NETOBJECT.DAT, NETNODE*.DAT, LMF$LICENSE.LDB and VMS$AUDIT_SERVER.DAT to the same location.
- Restart and archive weekly or fortnightly log files such as ACCOUNTNG.DAT, SECURITY_AUDIT.AUDIT$JOURNAL and OPERATOR.LOG. Note that you should never "restore" these files in the usual sense.
- Back up other files that change, such as command procedures in the SYS$MANAGER directory, when needed
- Perform restores by laying down the latest image backup, booting minimum, and restoring the volatile files as well as any ad-hoc changes. You could even automate this practice in the startup procedure by using user-defined SYSGEN parameters (for example, USERD1).

The above list is by no means exhaustive. You need to analyze your system disk to determine which files change and then decide which ones you need to keep.

5.5 What about queues?

The queue manager database is quite tricky to back up. You must first stop the queue manager process with using the STOP/QUEUE/MANAGER/CLUSTER command. (**Warning**: this command stops batch jobs that are running.) You can now use the COPY or BACKUP commands for the SYS$QUEUE_MANAGER.* files in the QMAN$MASTER directory. (However, you cannot back up these file while the queue manager is running.) When restoring the files, make sure that you rename SYS$QUEUE_MANAGER.QMAN$JOURNAL to version 1 before starting the queue manager. Otherwise, this file will be ignored.

Also, note that backing up these files with the BACKUP command is generally not recommended. Therefore, how can you protect yourself against loss of the queue manager database? Reconstructing the queues themselves is quite a straightforward procedure. You can easily use the output of a SHOW QUEUE/ALL/FULL command to generate a command procedure to build queues from nothing. This is an example of a backup that is a logical, rather than a physical, copy of the information being backed up.

Similarly, you can back up your license database by making a copy of the LMF$LICENSE.LDB file and generating a list of LICENSE REGISTER commands using the following command:

```
LICENSE ISSUE/PROCEDURE/DATABASE=COPY_OF_LICENSE.LDB
*/ALL/OUTPUT=file
```

(**Warning**: issue this command only against a copy of the license database, because the command places all PAKs in an ISSUED state, preventing them from loading.)

This leaves only the queue entries, which can be divided into 3 groups. The first group contains transient, ad-hoc jobs that happen to be on the queue when you are thinking of backing up. Such jobs, it can be argued, should not be restored at an arbitrary future time, because they have probably already completed, and you cannot predict the effect of rerunning or reprinting such a job.

The second group contains recurring, self-resubmitting jobs that should always be scheduled or be running. Although such jobs need to be restored, they are probably better handled by system or application startup procedures. When starting an application, check that its associated jobs are correctly scheduled. If not, resubmit them. This method avoids problems that system crashes, power failures, or data loss can cause. Another possibility is to use a scheduler product to control such jobs.

The third group are ad-hoc jobs that were scheduled but had not run at the time the queue manager database was lost. There is no simple answer for the last category, but I believe such jobs are quite rare. Combined with the rarity of losing the queue manager database, greatly increasing the complexity of a strategy to account for doubly rare events seems unreasonable. If necessary, use the output of a SHOW QUEUE/ALL/FULL command to reconstruct jobs as well as queues.

6. But what if my operation is 24 x 7 x 365?

If you really run a 24 x 7 x 365 operation that cannot tolerate any down time at all, then you need to be especially careful to ensure that your backup strategies really work. No operating system with reasonable performance can be expected to provide your backups for you. It is therefore mandatory that your application code be written to allow both backups and restores to occur in parallel with normal processing. No magic wands exist, but with careful application design and by using checkpoints and journaling and by having well-designed databases, there are no intrinsic reasons why you cannot achieve true 24 x 7 x 365 operations. Just do not expect them to happen by themselves or to be free.

7. Summary

Hopefully, the information presented here has made you think about the objectives of making backups and whether your current strategies are achieving those ends. Other questions you should ask yourself include:

Do I understand my real risks?
Do I understand my data?
Do I have a workable recovery plan?
Have I tested my plan?
Am I using the right tools and hardware?
Am I backing up the right files?
Does my BACKUP command really work?
Am I backing up more than necessary?
How am I dealing with my system disk or files, or both?
Do my applications have build-in backup capability?

Please contact your local Customer Support Center for more information.