

# Backup Management on a Large Network

Florin B. Manolache, Octavian Rusu

**Abstract** — Automatic network backup of a large number of computers in a heterogeneous environment is a resource-intensive task, both for the infrastructure and for the system administrators. The goal is to create an environment where the system administrator time allocated to backup services scales with the number of users and not with the number of computers or the amount of data. This paper presents some of the solutions implemented and currently used for optimization of backup management on an university network.

**Keywords** — backup management, networking, backup server, open source.

## I. INTRODUCTION

AN important role of the network is to provide an infrastructure for remote backup of computer data. There are many backup solutions currently available [1], some open source, others commercial, some easy to configure but lacking features, others very complex and refined but cryptic for the regular user, some on disk, others on tape.

While offering backup services during the last decade for a large network of computers in an academic environment, the authors experimented with multiple solutions and a large array of software packages. This paper offers an overview of the experience we got in the field and presents some of the work we have done to improve the efficiency of the backup process. Some of the ideas presented here are already implemented and used in a production environment containing more than 1000 computers including desktops, servers, and clusters. Other ideas belong to an open source software project for backup management, which is a work in progress.

Most existing backup solutions are tailored for the desktop user, usually involving local or network backup storage space (DVD, USB external disk, web service) and software that copies files from the main working area to the media. These solutions are expensive in terms of infrastructure and user time, and have deep privacy implications. They do not provide a systematic backup history; most users gladly skip backing up their systems. User-driven backup solutions are effective for laptops or home computers, but cannot be considered reliable and appropriate for enterprise data.

Florin B. Manolache is with the Mellon College of Science, Carnegie Mellon University, Pittsburgh, PA, USA (e-mail: florin@cmu.edu).

Octavian Rusu, is the CEO of RoEduNet and Digital Communications Department from the Alexandru Ioan Cuza University, Iasi, Romania (e mail: octavian@roedu.net).

Another class of solutions consists of enterprise level automatic backup setups. These offer the possibility of backing up a large number of computers via the network, and work very well for small groups of up to 100 computers. Unfortunately, as the number of computers rapidly increased in the last years, the existing backup software and network bandwidth needs don't scale properly.

Several main inconveniences of the existing solutions applied to large networks are:

- (a) increased complexity and lack of flexibility of the setup;
- (b) large amounts of bandwidth are involved;
- (c) the lack of reliability of the backup software involves important workload for the system administrators.

The next section defines a general representation of the backup software and explores its main points of failure.

## II. GENERAL STRUCTURE OF THE BACKUP SOFTWARE

Backup software deals with backup volumes. Each backup volume consists of:

- (a) a description that contains the list and properties of stored files, date and type of backup, name of computer, and backup history;
- (b) the data itself in an archive containing copies of the files which are eventually compressed.

Backup software has two important tasks:

- (1) Creating backup volume archives and the reverse operation of extracting files. This is a low level operation that can be performed in many ways and is specific to different platforms, operating systems, filesystems, and storage media. It is the most time consuming and unreliable part of the backup process.

- (2) Volume management consists of implementing policies concerning the contents of volumes, where the archive files are stored, type of backup, history, problem detection, and signaling. Most existing backup software has an extremely rudimentary volume management system based on a configuration file specifying hardwired values. For the best performance, volume management should be adaptive, using algorithms that reconfigure and optimize the archiving process on every run.

The most common point of failure of a backup system is connected to the creation of archives. This process is likely to encounter problems because of large files that may change while they are archived, bad media, network availability, or target media full. In our environment, we get an average of 1-2 processes getting stuck every week, usually on file servers connected to parallel clusters running big computation jobs. Recovering from such a

problem can be as simple as killing the offending process, which is usually the case, or as complicated as having to rebuild and reconfigure an entire file server. We found that the failure of archiving processes is not the fault of backup software, but just a consequence of the computing environment. These failures are statistical events that have to be taken into consideration by the volume management part of the backup software by implementing checks for each of the time consuming steps of the process, as well as automatic workarounds and signaling for the most common problems.

Consequently, what makes a good backup software is indeed the quality and the flexibility of the volume management part. That is the area where most improvements can be made; also this is the difference between a cheap consumer grade backup solution and a very expensive enterprise grade software.

The next section presents our setup for large scale network backup and comments on what was done to make the backup process as adaptive and trouble free as possible.

### III. MAKING BACKUP EFFICIENT

The task consists in providing automatic backup via the network for a rapidly growing number of computers (currently about 1200) consisting of desktops, servers, and computer clusters, spread in five buildings on different subnets which are part of the Carnegie Mellon University network.

Over the years, there was an accentuated increase in the number of computers, and the backup services were scaling very badly because of the nature of the software which was designed for desktops or small groups of computers. The system administrator time allocated to maintain the backup systems was growing faster than the number of computers. So in the late 90's we decided to fill in some of the gaps in the backup management part of the software using home-grown shell scripts. The (evolving) result of that initiative is the production software we are using today. Later on, we decided it was worth creating a new backup utility that would adaptively take care of most of the typical problems that we encountered. An open source software [2] named Lightweight Adaptive Backup System [3] written mostly in python [4], [5], was created and is still under development at labsproject.org. As a final objective for this project, good backup software should scale the system administrator time with the number of users, and not with the number of computers or amount of data. In other words, most of the time should be spent towards restoration of lost files, and not on the backup process itself.

Our backup infrastructure consists of a number of backup servers, several of them in each subnet (Fig. 1). These servers are typically dedicated computers containing large disks used exclusively for storing backup volumes. However, some workstations which have important storage needs, and many of the file servers associated with clusters, act as backup servers for the local files on a set of

dedicated backup disks.

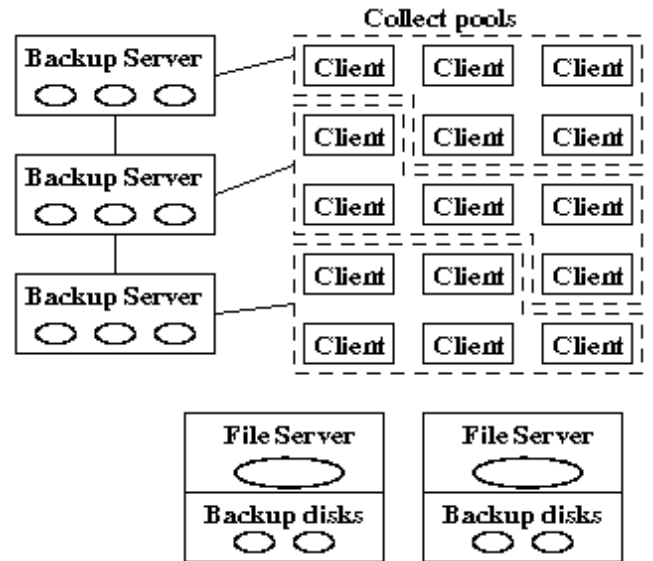


Figure 1. The backup infrastructure using dedicated backup servers.

The backup procedure is run via two sets of cron-driven shell scripts, one for the storage servers that collect volumes remotely from a pool of clients, and the other for building volumes on the clients. The client side script does volume management, as described in the previous section, and runs the archiving application. The volumes are stored either in a temporary area of the disk to be retrieved later by a backup server, or on a local dedicated backup disk. The server side script polls the clients and collects the completed backup volumes from their disks as they become available.

Let's see what is interesting about this setup, and how it is helping the efficiency of the backup system, as well as what are the challenges to be solved.

1. The archive is always created on a local disk for maximum speed. Since the amount of files to be archived is not predictable at the system setup, the main challenge is to make sure that the backup process doesn't fill the disk and doesn't need lots of sysadmin attention. For systems with local backup disks, this is as simple as adaptively tuning the backup history length to accommodate new backups, and providing large enough disks to be able to keep at least two full backups. For systems that have the volumes collected by backup servers, the objective is achieved by splitting the files into multiple backup volumes, and doing full backup to at most one volume in each cycle; the total size of files in each volume must not exceed the existing free space on the local disk where the volume archive will be temporarily created and stored. Ideally, the volume management algorithm should adaptively adjust the volume contents before each backup cycle if needed.

2. The archives contain compressed files. Compression saves disk space and network bandwidth on the expense of CPU cycles which are plenty to spare in modern computers. Backup software compresses files or archives since its early days, this is not a new idea. Surprisingly,

most modern backup packages either don't use compression in the default configuration and it needs a lot of work to be reinstated, or it is not the most efficient type available. Compression can be used at the file level (on the fly, as the archives are created) or at the archive level (the entire archive is compressed). We found that applications like afio [7] that do compression on the fly achieve the best performance time wise and generate the most robust archives against media corruption. The typical compression ratio we achieve is about 1/3.

3. The scripts implement a set of checks that will ensure the process can resume or recover after network or power failures, server or client reboots, stuck archiving processes, and other events that may interfere with the backup process. The checks can be realized in two ways:

(a) using lock files to signal when a certain stage of the process is active, and then decide if that process is stuck or crashed using the age of the lock file;

(b) using watchers that are stage specific and monitor the growth of the archive or network transferred files. Following the checks, a recovery procedure is applied if needed. The recovery procedure can be automated for the typical failure scenarios, or can consist of sending an email to the system administrator for further investigation.

4. The multiple dedicated backup server approach provides very good usage of the available disk storage space and bandwidth. It offers redundancy that will allow keeping the backup service functional even if some backup servers are down or unreachable. In such cases, the clients can be moved from the collection pool of the unusable backup server to another server that is functional.

By implementing these ideas, we managed to run the backup service without any data loss for over a decade, while keeping the system administrator time at very low levels (mostly regular maintenance, reorganization of volumes, and file restorations). Some interesting observations about the use of backup service are:

(a) An amount of disk space equal to the size of live data on the disk is enough to keep more than two months of backup history, full backup every month, and incremental backup every other day.

(b) Most data restoration requests come from users deleting their files, and not from hardware failures.

(c) By replacing the backup storage disks every other year and storing them, it is possible to recover valuable data that was deleted many years ago.

The next sections present the production and development implementations of the backup service.

#### IV. PRODUCTION IMPLEMENTATION

The production implementation consists of a set of shell scripts that work with lower level utilities (tob, afio, scp).

The servers collect volumes through a shell script that uses scp for transfer. The collection pools are maintained and modified manually. Failure recovery from network problems is realized via lock files; the process is considered stale if the lock file is older than three days.

The Unix clients build volumes using afio for archiving

and compressing files. Archiving is driven by a modified version of Tape Oriented Backup (tob) [6]. On top of tob we have a set of home grown shell scripts that automatically manage the storage location of the volumes, the type of the backup, and do failure recovery using lock files. However, the failure recovery is not yet completely automatic, and killing of the archiving/compressing processes by hand is needed in many cases.

For Windows clients we're using ZoneMinder which has a high initial setup time, but can be used for several desktops at a time. In our environment that is not an important drawback since the number of Windows systems is very small.

#### V. DEVELOPMENT IMPLEMENTATION

The implementation of our backup management ideas through shell scripts resulted in complicated code, which is hard to develop, to debug, and to adapt to new conditions. So we decided to start fresh with a project that has a modular structure allowing a high degree of flexibility. The project is named Lightweight Adaptive Backup System (LABS) and is hosted by labsproject.org. The software is written in python and is mostly developed by a group of graduate and undergraduate students.

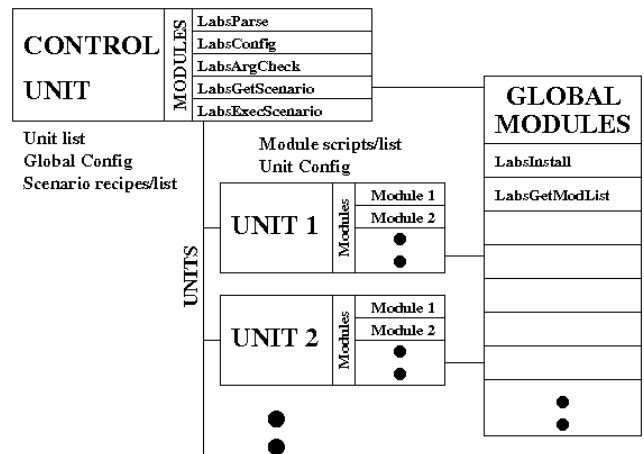


Figure 2. General structure of the LABS software.

The structure of the software is shown in Fig. 2. The building block is named unit. All the units have the same structure consisting of a main body that takes care of information flow, and a set of sequentially executed modules. Each module performs a small task resulted from partitioning the job of the unit. The units can take options, can use their own configuration file, and have a set of global variables.

Every unit has a very well defined function, and is logically isolated as much as possible from the rest of the code. That allows:

(a) multiple programmers to independently work on different units without much knowledge of each other's code as long as they respect the API described in the project wiki;

(b) easy change of the choices for different tasks; e.g. switching between methods of network volume collection

using scp or rsync takes only changing one module and very little knowledge of the programming details of the other modules or units;

(c) the same code structure and a lot of the actual code can be used for easy startup of other projects.

One of the units named the Control Unit has a central function. In fact execution of the software implies running the code in this unit. The command line options become options for the Control Unit and the global configuration file of the software is actually this unit's configuration file. The labs command line contains the name of an action. Actions are defined by scenarios, and a scenario corresponds to a sequence of units to be executed. The Control Unit basically identifies the action in the command line, and then runs the corresponding scenario by executing the units. Each unit receives a fresh global set of variables containing some global values (as the location of different directories) and the variables defined in the global configuration file. Passing variables from one unit to the other is restricted and discouraged, the preferred communication method being through files. This choice is justified by the following:

(a) speed is not a concern at this level; archiving or moving around backup volumes is much slower than the management software itself;

(b) using files gives a very good description of the state of runs in case of a system crash, and that helps a lot with recovery;

(c) the structure of files exposes the API between different units.

Table 1: Some of the actions implemented in LABS.

backup	Run a backup pass on a computer.
collect	Retrieve volumes from the pool hosts to the storage server.
config	Use first to build config files; or use for re-configuration.
dfree	Gets the free disk space on the partition containing a directory.
extract	Extract files specified by restore requests.
get	Get the backup files from a remote host.
help	Start help interface (default action).
info	Display backup system status.
restore	Place a restore request using an interactive interface.
watch	Watch a volume file creation/transfer and interrupt if stalled.

Some of the actions are shown in Table 1. One design problem for scenarios was the implementation of loops. The developers decided to keep an open mind about creating loops, but to avoid them if possible, and that was possible so far. Most of the cases when iterative action was needed were handled by launching multiple instances of LABS. That solution is more robust (one stuck process doesn't stop the entire backup service) and suits the new generation of multi-core computers well.

One example illustrating how the system works is the retrieval of the backup volumes from a client to a server. For this case, "labs collect" is used, which launches an instance of "labs get" and one of "labs watch" for each client. The "labs get" process does the actual file transfer, while "labs watch" watches the progress of the transfer and provides recovery in case of failure (e.g. network failure or client going down). If "labs get" finishes successfully, "labs watch" marks the request as fulfilled and logs it.

The project is still under heavy development. We plan to start using the labs software for backup starting sometime in 2008.

## VI. CONCLUSION

It is possible to configure and run a low maintenance backup service that will cover a large number of computers. Depending on the specifics of the environment, a small number of improvements can transform a high maintenance, hard to configure backup software into something usable that scales nicely with the number of users. This was achieved by observing the typical points of failure of the existing system, and building automated ways to resume the backup or recover from these failures. That resulted into the storage client/server structure presented in this paper. Some interesting areas under current investigation consist of automatic and adaptive reconfiguration methods applied to the volume contents on the clients and collection pools on the servers. A new software project described at labsproject.org is developed to cover those areas.

## REFERENCES

- [1] W. Curtis Preston, *Backup & Recovery*, O'Reilly, 2006.
- [2] Karl Fogel, *Producing Open Source Software*, O'Reilly, 2005.
- [3] *Lightweight Adaptive Backup System*, <http://labsproject.org/>.
- [4] *Python Programming Language*, <http://python.org/>.
- [5] David Ascher; Alex Martelli; Anna Ravenscroft, *Python Cookbook*, 2nd Edition, O'Reilly, 2005.
- [6] Stephen van Egmond, based on previous work by Karel Kubat, *Tape Oriented Backup*, <http://tinyplanet.ca/projects/tob/>.
- [7] Koen Holtman, *afio*, <http://freshmeat.net/projects/afio/>.