;login:

THE MAGAZINE OF USENIX & SAGE
June 2003 • volume 28 • number 3

inside:

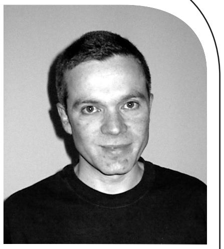
USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

organized pruning of file sets

by Diomidis D. Spinellis

Diomidis Spinellis is an Assistant Professor in the Department of Management Science and Technology at the Athens University of Economics and Business; he is the author of *Code Reading: The Open Source Perspective* (Addison Wesley, 2003).

dds@aueb.gr



In a number of backup scenarios, backup files simply accrue in a directory that should be periodically cleaned up. Typical examples include backup data from databases, PDAs, network clients, and routers. The data contents of the above are often not directly backed up onto tapes, but are copied to disk-based files by an appropriately scheduled cron(8) job.

The continuous increase in disk capacities enables us, in many cases, to keep multiple sets of these backup files in a given directory as a substitute for a regular tape backup. These files are typically kept in a manageable size by periodically purging all old files. Well-organized tape-based backups, however, offer an additional advantage: Through a carefully staged tape retention schedule, users can often retrieve files much older than the number of retained tapes would suggest.

A tape backup schedule might, for example, involve daily incremental backups, weekly full backups retained for two months, and monthly tapes retained for two years. If I discover today that sometime in the previous six months I deleted a file I had created a year ago, I can go to the retained monthly backups that followed the file's creation and retrieve it from there.

Backups to files are not often organized in this manner. Two approaches I have seen for managing their size involve either naming each file with a periodically repeated date element, such as the day of the week or month, so that newer files will overwrite older ones, or tagging each file with a unique identifier, such as the complete date, and having a separate script remove files older than a given date. Both approaches, however, lack the property of selectively retaining a subset of older files. More elaborate schemes can, of course, be constructed by carefully synchronizing and staging separate cron(8) jobs, but I have never seen them applied in practice. The problem of selectively retaining old files gets especially difficult when the backups are created at irregular intervals – for example, each time I synchronize my PDA or remember to back up my cellular phone directory.

On the other hand, a file-based backup scheme offers the additional possibility of automatically examining all the retained files and selectively pruning those we decide are not worth keeping. The key concept for deciding which files to keep is a *retention schedule*. In tape-based schemes, this simply revolves around weeks, months, and years. If we have a tool for managing the file pruning, we can be more creative in selecting a retention schedule and, hopefully, use one that will, violating Murphy's Law, offer us an increased probability of recovering that old file we discovered missing.

Retention Schedules

When I first decided to work on the file pruning problem, I considered using an *exponential* retention schedule. I would like to keep yesterday's backup, a backup from two days ago, then backups aged 4, 8, 16, 32, and 64 days. With 10 files I could cover a period lasting more than a year. This schedule uses two as the schedule's base; one could select any smaller number to increase the number of retained files or a larger number to decrease them. The idea behind this schedule is that recent backups are more valuable than older ones.

Creeping featurism made me think of different possible schedules. One other possibility is a Fibonacci schedule. Here the retention sequence starts with 1, 1, and each subsequent term is the sum of the two previous ones: 2, 3, 5, 8, 13, 21 34, 55. At that point, I had to wonder which of the two schedules was better for securing my valuable data.

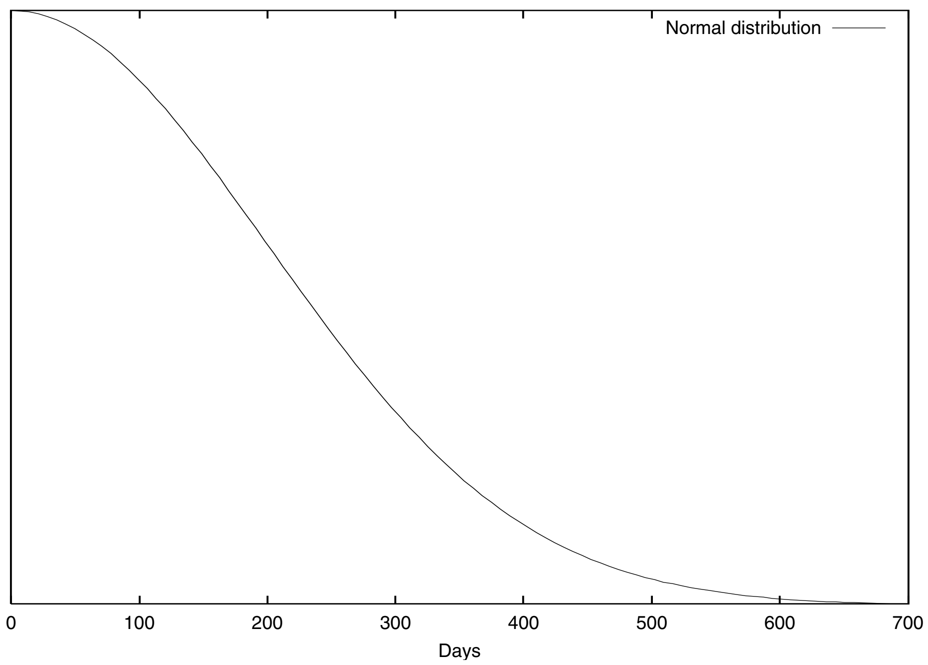


Figure 1

It turns out that neither is. If we were to sample data recovery requests in a large data center, we would probably find that the age of the requested files would follow the ubiquitous bell-shaped *normal*, or *Gaussian*, distribution. The exact shape of the bell is determined by the *standard deviation* of the requested file ages; this expresses the variation (in the same unit as we measure the file ages) between the ages of different requested files. Since recovery requests from all users (apart from pointy-haired managers) always refer to the past, the shape is actually one-half of a bell curve. You can see the normal curve for a standard deviation of 200 in Figure 1.

The formula defining the normal curve is actually quite complex:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}}$$

but once it is coded in a program, its application can be a breeze. The curve represents the probability that a file of a given age will be requested.

You can see that, following intuitive expectation, as files age they are less likely to be needed. In order to distribute our archive files in a way that reflects this diminishing probability distribution, we need to define our retention interval schedule so that the interval's length is proportional to the probability of requiring a file within that interval. This is represented by the area under the curve for the given interval; for the mathematically inclined, the area for an interval from a to b is given by the integral

$$\int_a^b f(x)dx$$

We therefore need to divide the whole area under the curve into a number of equally sized parts, as many as the files we can afford to retain, and then calculate the respective intervals.

Unfortunately, there is no mathematical formula with a finite number of terms that can give us the numbers we are looking for. Initially, I wrote code to numerically integrate the normal function, adjusting the interval while moving back into time. A few days later, my colleague Stavros Grigorakakis, reading a draft of these notes, pointed me to an excellent analysis of the Gaussian function available online at *http://mathworld.wolfram.com/GaussianDistribution.html*. There I found that the cumulative distribution function (the integral I was painstakingly calculating) can be determined by means of the so-called *error function*, which – surprise, surprise – is part of the UNIX C math library. You can see in Figure 2 how we would spread 30 files in a period of around 2000 days using an exponential distribution with a base of 1.3 and a normal distribution with a standard deviation of 1000. For comparison purposes, I have also included how a Fibonacci distribution and an exponential distribution with a base of 2 would appear in the above scheme; only 18 files would fit in the Fibonacci distribution and 12 in the base-2 exponential.

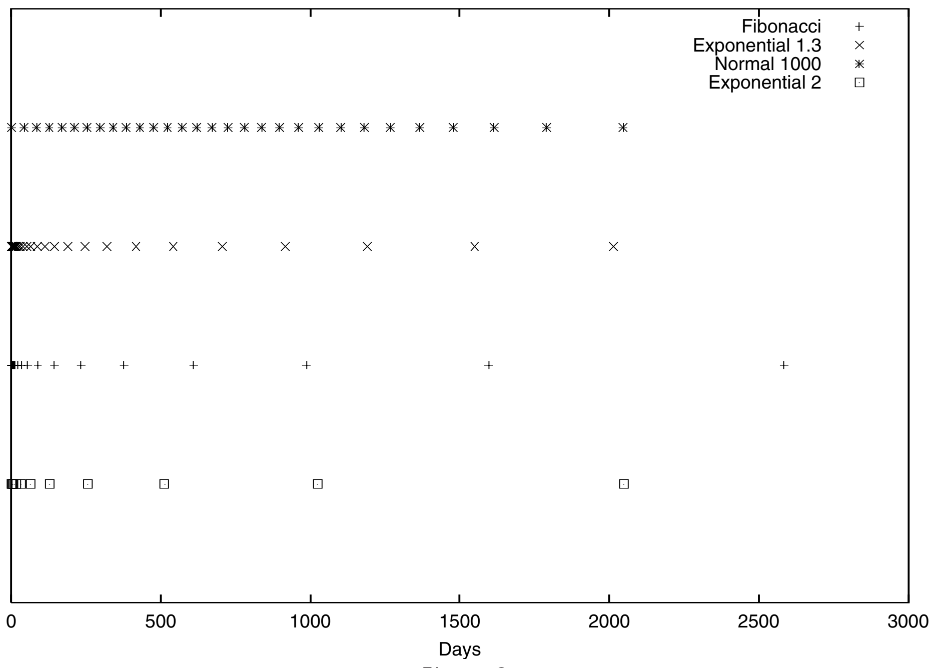


Figure 2

The Prune Tool

Putting code where my mouth is, I wrote a C program to implement the file-pruning strategies described above. It is available for download in source form through a BSD-style license from *http://www.spinellis.gr/sw/unix/prune*. *Prune* will delete files from the specified set, targeting a given distribution of the files within a certain time, while also supporting size, number, and age constraints. Its main purpose is to keep a set of daily-created backup files in manageable size while still providing reasonable access to older versions. Specifying a size, file number, or age constraint will simply remove files starting from the oldest, until the constraint is met. The distribution specification (exponential, Gaussian, or Fibonacci) provides finer control of the files to delete, allowing the retention of recent copies and the increasingly aggressive pruning of the older files. The retention schedule specifies the age intervals for which files will be retained. As an example, an exponential retention schedule for 10 files with a base of 2 will be:

1 2 4 8 16 32 64 128 256 512 1024

This schedule specifies that for the interval of 65 to 128 days there should be (at least) one retained file (unless constraints or other options override this setting). Retention schedules are always calculated and evaluated in integer days. By default *prune* will keep the oldest file within each day interval, allowing files to gradually migrate from one interval to the next as time goes by. It may also keep additional files, if the complete file set satisfies the specified constraint. The algorithm used for pruning does not assume that the files are uniformly distributed; *prune* will successfully prune files stored at irregular intervals.

Prune is invoked through the following syntax:

prune [**-n**|**-N**|**-p**] [**-c** *count*|**-s** *size*[**k**|**m**|**g**|**t**]|**-a** *age*[**w**|**m**|**y**]]
 [**-e** *base*|**-g** *standard deviation*|**-f**] [**-t a**|**m**|**c**] [**-FK**] *file ...*

The numerous options reflect the tool's flexibility. You can specify the distribution to use (exponential, Gaussian, or Fibonacci) using the -e, -g, and -f options as well as the constraints for the number (*count*), *size*, or *age* of the files to retain using the -c, -s, and -a options. By default the constraints are used to specify the upper limit of the size or number of files that will be retained. If more files can be accommodated (because, e.g., An important aspect of a disk-based backup system is the employed retention schedule.

some intervals are empty), or the specified size limit has not been reached, *prune* will retain additional files, deleting old files until the constraint is satisfied. The -F flag can be used to override this behavior. On the other hand, if a constraint is violated, *prune* may not retain any files in a given interval; the -K flag can be used to always keep at least one file in each interval. Finally, the -t flag allows you to specify whether *prune* will use the creation, access, or modification time of the specified files for determining their age.

The following examples illustrate some possible uses for *prune*:

```
ssh remotehost tar cf - /datafiles \ >backup/`date +'%Y%m%d'`
prune -e 2 backup/*
```

backs up remotehost, storing the result in a file named with today's timestamp (e.g., 20021219). Subsequently, prunes the files in the backup directory so that each retained file's age will be double that of its immediately younger neighbor.

```
prune -g 365 -c 30 *
```

keeps at most 30 files. The ages of these files will follow a Gaussian (normal) distribution, with a standard deviation of one year.

```
prune -e 2 -s 5G *
```

prunes the specified files following an exponential schedule so that no more than 5GB are occupied. More than one file may be left in an interval if the size constraint is met. Alternatively, some old intervals may be emptied in order to satisfy the size constraint.

```
prune -F -e 2 -s 5G *
```

acts as above, but leaves no more than one file in each scheduled interval.

```
prune -K -e 2 -s 5G *
```

acts as in the first example of the 5GB-constrained series, but leaves exactly one file in each interval, even if this will violate the size constraint.

```
prune -a 1m -f
```

deletes all files older than one month; it uses a Fibonacci distribution for pruning the remaining ones.

Conclusion

Increasing disk capacities and network bandwidth allow us to implement disk-based backup mechanisms. An important aspect of a disk-based backup system is the employed retention schedule. The prune tool allows you to rationally specify and automatically manage the retention schedule to suit your needs. An exponential schedule with an integer base or a Fibonacci-based schedule can be easily understood by unsophisticated users, while a schedule with a normal distribution and an appropriately set standard deviation is more likely to reflect your true file-retention requirements.