A Portable History

Erik Quanstrom
quanstro@coraid.com

ABSTRACT

History from the dump filesystem has traditionally been locked into the fileserver on which it was created. I describe the method used for transferring our history from the old fileserver, Plato, to the new fileserver, Kibbiee, without making any modifications to Plato.

Introduction

Earlier this year the dump filesystem on our main fileserver, Plato, surpassed 75% usage. Since Plato is running a 32-bit fileserver on four-year-old hardware, we decided to build a new fileserver instead of adding new disks to an old machine. Usually starting with a new on-disk layout means that history cannot be transferred to the new machine and the old fileserver must be down while a snapshot of the current filesystem is taken. Even if the same on-disk format is used, both fileservers must be offline during the transfer and there is the risk of damaging the original filesystem due to operator error. I outline the method used to copy history from the 32-bit Plato to the 64-bit Kibbiee while Plato continued to serve files.

Method

Both fileservers were placed (silently) into allow mode. Kibbiee's date was initially set to Midnight on June 11, 2004, the first day of Plato's dump. For each historical day of the dump, *updatedb*(1) was used to generate a list of changes to the filesystem from the previous day, a database and a log file. For the first day of the dump, the preceding day was set to the first day of the dump and the database was empty. Thus all files from that day were added. The *C*(1) command was used to set the date on the fileserver to midnight Standard Time on the day of the dump. Changes to `adm/users` were applied first so files created would have the correct owners, then other changes were processed, both with *cphist*(1). Finally a dump was forced on Kibbiee. This process was repeated for each day of the dump. 1024 dumps were processed. Each dump took approximately ten minutes to process.

Fileserver Changes

No changes were made to Plato. Kibbiee's kernel required changes to PCI enumeration and interrupt handling. The time functions were updated to support Daylight Savings Time in any timezone using the data files from the Plan 9 C library. Support was added for reading `nvram` from a Plan 9 `9fat` partition by reading the partition table *9load*(8) leaves in core. This facilitates booting a cpu kernel on the fileserver for maintenance operations such as loading a new kernel. This can't be done easily on another machine because the fileserver boots from a flash disk.

In addition, new drivers were written or ported for the Myricom Lanai z8e 10 gigabit ethernet adapter, the Intel i82563 PCIe-based gigabit ethernet adapter and the Marvell 88SX[56]xx hotpluggable SATA controller.

Device Copy

Since the Marvell controller supports hotpluggable SATA drives, we are able to copy the worm onto drives that are physically moved to an offsite backup fileserver. The same process for copying history as outline here could be used. However this seems error prone and using allow mode at predictable times presents a security problem. To allow for online backups, the `devcopy` command was added. The following example will copy blocks 0 through 183140625 from `[m2m3]` to `[m7m8]`,

```
devcopy start [m2m3] [m7m8] 0 183140625
```

The final two arguments are optional parameters. The command is executed in the background by a single, dedicated process. Console control returns immediately. There can only be one active device copy at a time. With no arguments, `devcopy` prints out its progress. The arguments `pause` or `resume` apply to the last copy started.

It is expected that the operator will either arrange that no dump is taken while a device is being copied, or will have an offline process to erase the inconsistant data at the end of the dump filesystem.

QID relationships

In order for the history to work correctly on the new server, we need to insure that files are appended or deleted and recreated exactly as they were on the original. Just inspecting the mode bits is insufficient. For example an mbox has the append bit set. However each time it's edited the old file is deleted and a completely new file is written.

In order to transfer history correctly, we require that following properties hold. Whenever to files on Plato have the same `qid.vers`, they must also have the same `qid.vers` on Kibbiee, although the absolute value on either is not important. We also require that whenever two files have the same `qid.path` on the Plato that they have the same `qid.path` on Kibbiee.

The new program *cphist*(1) does just this. When files are listed as changed or created, the `qid.path` and `qid.vers` are carefully inspected to decide if the file should be deleted and copied or if only the modified blocks need to be copied. The last-modified user is also set. This is sufficient for all cases except when a file is renamed. In this case, the old file is always deleted and a new file is created. This case resulted in Kibbiee's fake WORM using about 20% more blocks than Plato's.

Issues

Due to a spam problem in 2006, there were two directories on Plato that had over 2 million entries. Scanning these directories took approximately 45 minutes each. This increased the time needed to transfer one day's dump from 10 minutes to 100 minutes. This problem was overcome by binding an empty directory over the two large directories before running *updatedb*(1).

Two errors were encountered when a non-empty directory was deleted and replaced by a file. This problem was encountered because *updatedb*(1) does not emit deletes first. This problem was addressed in an *ad-hoc* fashion.

After Kibbiee became the active filesystem, it was discovered that a missing line in the 9p code in the fileserver kernel prevented the `muid` from being set. Thus *history*(1) output lists incorrect users. Since `adm/users` changed during the process of loading history, the mapping of users to the `uid` of channel loading the history changed as history was being loaded.