The Design and Implementation of the Network Backup Control Language

John C. Orthoefer (Internet: jco@beach.cis.ufl.edu)
University of Florida
Department of Computer and Informational Sciences

ABSTRACT

Dumper is the implementation of a language to simplify the process of doing backups on a networked computer system. The language, called Network Backup Control Language (NBCL), permits the user to relate UNIX† mount points to backup devices, on a single TCP/IP network, and executes these relations on specified days. With backup schedules put forth in a formal language, regular backups can be done efficiently and completely.

INTRODUCTION

Currently, backups are performed either at irregular intervals or by specialized personnel at most installations. Now, by putting forth a schedule in a formal notation that can be programmatically interpreted backups can be done by non-specialized personnel.

In the past, computers had small file systems and contained their own secondary storage devices. With such a system, backups of the entire file system could be performed by a single person in little time. As computer systems' grew, their file systems capacity also increased, causing a corresponding increase in the amount of time needed to make effective backups. One of the steps taken to reduce the amount of time needed to execute back-ups was to introduce incremental backups. Incremental back-ups reduce the amount of data duplicated on multiple backup tapes.

An incremental backup is accomplished by noting when the last backup of a file system was done, and then only placing the new data on the secondary backup device. The new data is called a forward delta. The method that the Berkeley `dump` [1] program uses to achieve incremental backups is to assign each backup a number and store that number (also referred to as a level) with a date-time stamp and the file system to which the date-time and level apply. When a backup is begun, a level is stated and the program then looks up the date of the previous backup at the next lower level, so that the range of dates of files to be backed up at this level will be known.

For example, the method Berkeley suggest, for doing incremental backups is to do a level 0 every month. During a week, a modified Tower of Hanoi sequence is used: Level 3 on Sunday, 2 on Monday, 5 on Tuesday, 4 on Wednesday, 7 on Thursday, 6 on Friday, and 9 on Saturday. It may also be desirable to do a level 9 at the end of every day.

While an incremental backup system reduces the amount of data that needs to be backed up, the complexity of the schedule is increased. To further complicate a backup plan, many computers are now part of a network with a combination of disked workstations and mainframes which contain both user and system files. But, not all machines have a secondary backup device and the backup devices are not necessarily all identical.

`Dump` [1] was modified by Berkeley to work with networks. The network version is called `rdump`. The operation of `rdump` is very similar to `dump` with the following exceptions in file names and tape control. All file names, for files on remote machines are written as `hostname:/path/file name`. The path should be given from root. `Rdump` also starts up a process on the machine that contains the secondary backup device. This process is used to control the tape device.

Currently, in a medium to large computer system, several file systems on different machines need to be backed up on a daily basis. The problem is that there is no formal way to keep the schedule information in a central location. Normally there are only a few people in the system administration that know what backups are to be done and when. Also, with the backups being done by hand, mistakes can be made.

Network Backup Control Language (NBCL) is a way to formalize the scheduling of file systems to be backed-up. `Dumper` is the name of the interpreter that will control Berkeley's `rdump` [1] command to do backups across a Transmission Control Protocol/Internet Protocol (TCP/IP) network.

TECHNICAL OVERVIEW

In the discussion that follows it is assumed that the reader has some familiarity with TCP/IP, Network File

† UNIX is a trademark of Bell Laboratories.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
©1990 ACM 0-89791-356-6/90/0400/0329 $1.50

System (NFS), and BSD 4.3. The relevant parts of BSD 4.3 are the command `rdump`, and UNIX device files and mount points.

`Dumper` was developed to execute the `rdump` command. The `rdump` program is part of the BSD 4.x release, therefore it is easily accessible to all UNIX installations. `Rdump` requires a raw device file with which to operate. Raw devices are not exported by the NFS mounting of the drive. Therefore, `rdump` is required to be executed on the machine that the file system is physically mounted on. On the machine that supports the secondary storage device, normally a tape backup unit, `rdump` executes the `rmt`[1] command. `Rmt` is the Remote Magnetic Tape control program which lets `rdump` read and write the tape.

When `rdump` is finished with a tape, anyone who has access through the machine to the tape drive could write to the tape. To prevent this from affecting any backups, when `rdump` gets done, dumper immediately sends an *offline* command to the tape drive through the `mt` command.

Dumper also needs access to the `rsh` command, so that it can start up remote shells over network links. `Mt` must be executed on the machine that the tape drive is physically mounted on, while `rdump` must be executed on the machine that the disk is physically mounted on. This also introduces the requirement that the machine and account that dumper is run from must be *trusted*. That is to say, the machine must let dumper login without a password.

Dumper was designed to be as general as possible with regard to the size of the network to be backed-up. Therefore, a macro facility and conditional execution capability were added. This is done by piping the NBCL source file through the C preprocessor, `cpp`. When dumper does this it passes a predefined value for `MACHINENAME` to cpp. This value contains the internet name. The machine's internet name in capitals is also defined, eg. beach would be passed as `BEACH` with a value of 1.

With the conditional execution it is possible to build a prototype file, that is then distributed to several different subnets. Each subnet will then do their own backups. This gives a system administrator central control and distributed execution of backup schedules. With distributed resources and decentralized execution one can add redundancy to a backup scheme. Hence, if some of the machines in a network go down, backup capability is not lost.

NBCL STRUCTURE

There are four pieces of information that dumper requires to operate: devices to accept back ups, file systems to be backed up, which levels on what days, and a set of relationships for the previous items. Since each item is independent of the others they are placed separate sections of the NBCL source file. Each section is proceed by a percent sign (`%`) and the name of the section. The sections are named `devices`, `schedules`, `filesystems`, and `dumps`.

The first section of the source file is the `device` section. In this section the user gives symbolic names to sets of devices. The syntax of a symbolic list is a name, a colon, and a list of comma separated devices and/or a list of previously defined symbolic names, the list is terminated by a semicolon. If you are listing a device you must specify the device name as a hostname and a UNIX device file, the density of the device, and a length. Normaly the length is the length of the tape on the device.

The `schedules` section of the source file follows the device section. This section is comprised of a semicolon separated list of the following form: a symbolic name, a colon, and a comma separated list of days and dump levels. The days that NBCL reconize are `sun` for Sunday, `mon` for Monday, `tue` for Tuesday, `wed` for Wednesday, `thr` for Thursday, `fri` for Friday, and `sat` for Saturday. The list of days is terminated by a semicolon.

The next section of the NBCL source file is `filesystems`. This section contains a semicolon separated list having the form: name, colon, and a list of file system mount points to be backed up. The file system mount points are in the format of `hostname:/path`, the path to the mount point must be from the root.

The final section is named `dumps`. The `dumps` section relates the previous three sections. This section contains a series of semicolon separated lines with a format: file system name `to` device name schedule name.
The schedule in figure 1 backs up the system files every Saturday. The user file systems get backed up 3 times a week; Monday, Wednesday, and Friday. The system files always go to the manatee tape drive, and the user files go to the first free drive.

```
%devices
tape1: device=manatee:/dev/rst8 density=800 length=900;
tape2: device=beach:/dev/rmt16 density=6250 length=2400;
alltapes: tape1, tape2;

%schedules
week_a: mon 6, wed 7, fri 8;
weekend: sat 1;

%filesystems
beach_usr_fs: beach:/cis/beach0, beach:/cis/beach1;
manatee_usr_fs: manatee:/usr
sysfs:  beach:/etc, manatee:/etc;

%dumps
sysfs to tape1 weekend;
manatee_usr_fs to alltapes aweek;
beach_usr_fs to alltapes aweek;
```

FIGURE 1

SECURITY AND BACKUPS

Doing backups over a network can appear to cause several problems with system security. Dumper must be able to login to an account with the same login id on every machine on which backups are to be done. Likewise, `rdump` must be able to login in to the machine that has the tape device mounted.

When machines are going to be logging in without passwords, one might want to secure the account. To secure accounts currently under UNIX a system administrator will create a small root file system with a minimum of commands in it. The login would then be setup to do `chroot`, then the login directory would be the highest point in the directory structure to which the account could get.

While this would work to lock up the account, `rdump` must be able to get to raw devices, this also can be placed in the captive account. However, since the raw devices can be read, and written, without going thru the operating system, making a captive account for dumper will not increase security.

A possible way to set-up dumper to minimize security risk is to make its password entry a star (*). The encryption algorithm used by UNIX, for password verification, will not map any plaintext string to a single star. The login would be accomplished through the use of `.rhosts` files. A `.rhost` file is a list of accounts logging in from particular machines that may login without passwords.

In addition to the *trusted* login. Dumper must be able to execute `rdump`, which is done by placing the dumper account in the same group as the owner of the `rdump` program, the group is often named operator. `rdump` must also be able to set its uid to root so that `rdump` can get raw access to the disk drive.

SOFTWARE ENVIRONMENT

Dumper utilizes several existing Berkeley Standard Distribution (BSD) 4.3 commands. The commands Dumper executes are `rdump`, `mt`, `cpp`, and `rsh`. By using these existing commands, rather than duplicating their functionality, it is possible to modify the networking protocols with few or no changes to the interpreter.

The programming languages chosen for this project were Flex[2], Bison[3], and C. All three are available from the Free Software Foundation. Flex generates a lexical analyzer. Bison generates the parser. Finally, C was needed to compile the lexer, parser, and the actions that are needed to tie the whole program together. Also available from the Free Software Foundation is cpp which provides macro and conditional interpretation in NBCL.

CONCLUSION

Dumper is currently in daily use at the University of Florida. While there is some room for improvement, mostly in the area of intermachine communications, the program works well currently. The major goal of simplifying backups on networked computer systems was met totally.

ACKNOWLEDGMENTS

I would like to thank my senior project director, Dr. Joseph Wilson, for helping me through the rough spots. A big thank you also goes to Andy Wilcox who helped me get NCBL off the ground.

REFERENCES

[1] *UNIX System Managers Manual*, Virtual VAX-11 Version. Bell Laboratories, Modified by The University of Califoria, Berkeley, California. April, 1986.

[2] Paxson, V. *FLEX*. Cambridge, MA: Free Software Foundation, Inc, 1989.

[3] Donnelly, C. and Stallman, R. *BISON, The YACC-compatible Parser Generator*. Cambridge, MA: Free Software Foundation, Inc, 1989.