Burt

Backup and Recovery Tool

Eric Melski & Dean Jansa

August 25, 1998

Abstract

Burt is a freely distributable parallel network backup system designed with flexibility, scalability and speed as its primary concerns. It is an extension to Tcl/Tk which adds a handful of commands to the standard Tcl/Tk library. As such it allows a high degree of flexibility in its implementation. Burt is capable of backing up data from virtually any source. Support for the parallel backup of many systems directly to tape ensures high backup speeds, often as fast as the tape drive is physically capable, and a degree of insulation from slow individual systems. Data written to tape is checksummed and, once a backup run is complete, read from tape and checksummed again to verify the integrity of the data. At the UW-Madison Computer Sciences department, Burt is used to backup approximately 500 GB of data on 350 workstations and 13 AFS file servers to a collection of DDS-DC 4mm tape drives.

1 Background

Burt was born out of the need for a extensible backup and recovery tool. We were faced with the need to backup hundreds of workstations of various flavors of Unix and filesystem types. In addition, we needed to be able to easily support backups for our substantial AFS filespace. Whatever system we decided upon needed to be capable of dealing with future hardware, software, and networking challenges. Finally, the system had to be fast. We backup approximately 55 gigabytes of data every night, and those backups have a window of only several hours to run.

For several years, we had been using DK from Tektronix [9]. This tool had many good features, including support for multiplexing, or the parallel backup of several systems simultaneously to a single tape drive; some flexibility in terms of what systems could be backed up; and some general concepts, such as the idea of the system as an engine or tool to achieve backups.

However, DK also had a number of shortcomings. The multiplexing support was inflexible and inadequate for our growing installation – it could only handle backing up a few systems simultaneously, and it required that systems be "hardwired" to backup in a particular order. Although it had some flexibility, that too was limited – all supported systems had to look the same to DK. Finally, DK had a disturbing tendancy to hang when hosts being backed up died, and the code, which was written in the mid 1980's, was inordinately difficult to maintain. The most modern operating system we could run DK on was SunOS 4.1.3, and we wanted to move to something more modern.

1.1 Why not use *(insert backup system here)*?

Initially, we tried to find an existing system that would meet our requirements. None were found.

Commercial systems had a number of problems from our standpoint. Many of these systems use proprietary client software on the computers to be backed up. This limits the range of systems that can be supported – only those for which the maker of the software has released a client can be backed up. In addition, it introduces an inevitable lag between the emergence of a new type of system and the introduction of backup support for that system. We needed to be able to support essentially any system, as soon as that system was available.

Not all commercial systems use propietary client software. However, other problems prevented us from using those systems, including prohibitive costs and in some cases inadequate performance for our needs. In addition, none of the systems we examined seemed to quite fit our ideas about how backups should work. For example, some required that the backup host run Windows NT, but at that time we had no such computers.

We also considered freeware solutions, in particular Amanda, a backup system developed by the University of Maryland [1]. However, we felt that Amanda did not quite fit our needs for several reasons. First, though it does support multiplexing to a temporary storage location, it does not support multiplexing directly to the tape [2]. This is problematic because it requires a hard disk large enough to hold the largest backup image in order to enable multiplexing. Unfortunately, our largest backup image occupies about 4 gigabytes, and when we were considering Amanda in mid-1995, hard disks were really too expensive for this to be practical at our site. In addition, this method of multiplexing negatively impacts overall performance, as it introduces an intermediary stage to which data must first be written and subsequently read, instead of simply reading once (from the original source) and writing once (to the tape drive).

Another problem we perceived in Amanda was its inability to gracefully handle backup images larger than the backup media. At present, we use DDS2 tape drives with 90 meter tapes; the nominal capacity of these tapes is 2.0 gigabytes. Assuming an optimistic compression rate of 2:1, that capacity can be increased to 4.0 gigabytes. As noted, our largest backup image occupies about 4 gigabytes. With Amanda, we would be unable to backup that filesystem unless we used something like GNUtar and split the filesystem in to smaller pieces. However, we felt and still feel that GNUtar is no backup program. We prefer to use the native backup programs of the filesystem in question.

Amanda also had some security issues at the time we considered it, and it is only now starting to implement support for more secure networking protocols. It uses some client side software which we felt was unnecessary and inconvenient, although not as troublesome as the proprietary software used by commercial systems. It supported the definition of new types of systems to backup, but that definition didn't allow enough flexibility for our tastes.

1.2 Burt is born

We finally decided that the best solution for our problem was to write a new tool which would incorporate all the features that we wanted: unlimited flexibility, multiplexing support, scalability, and high performance. That tool was called Burt, short for Backup and Recovery Tool.

2 Design concepts

In order to achieve our goals, we decided to make Burt an extension to Tcl/Tk, a full featured scripting language produced by Sun Microsystems originally, and Scriptics Corporation later. We divided the problem of backups into primitive operations:

- Specifying what is to be backed up
- Putting data onto a tape
- Checking what is on a tape
- Reading data off a tape
- Obtaining status information

These primitives were implemented as new Tcl commands which comprise the Burt engine. These commands can be thought of as a toolkit; it is up to the administrator to decide what to build with those tools. An actual Burt installation consists of the Burt engine in conjunction with Tcl/Tk scripts that make use of the engine to make backups happen. The end application is written in Tcl, and can take advantage of all the power that Tcl/Tk offers, as well as any other extension loaded into Tcl.

At the simplest level, Burt is really nothing more than a multiplexor. It takes data from any number of input streams and combines those streams into one output stream. Input streams can come from any source, though the intended application is to use Burt to multiplex data from programs such as `dump(1)`, `cpio(1)`, or `tar(1)`. Figure 1 shows broadly how Burt fits into a backup solution.

A few other concepts are critical to the design and functioning of Burt – backup types, schedule items, channels, and filemarks.

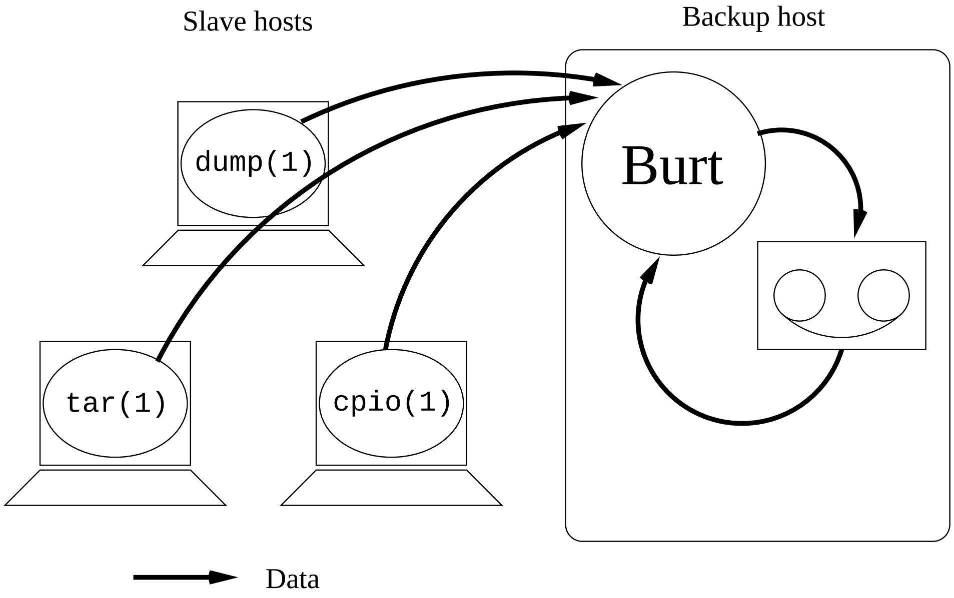

Figure 1: Data flow in Burt

2.1 Backup types

Backup types are the key to Burt's flexibility. Fundamentally, a backup type is a definition of how to backup something. For each type of filesystem or backup, the system administrator creates a Tcl procedure which initiates the backup and then returns a data stream from which Burt reads backup data. There is no limit to the complexity or simplicity of this procedure. The only requirement is that the procedure accept a few arguments which specify what particular item is to be backed up, and that it return a readable data stream. Once the procedure has been written, it is associated with a specific backup type via a call to `register_backup_proc`. Then, when Burt needs to backup an item of that type, it calls the appropriate procedure to do so. An example is the `unix_dump` procedure, which uses `rsh(1)` to connect to a remote host and `dump(1)` to backup data from that host:

```
proc unix_dump { host atom level } {
    # determine a logfile name
    set logfile /tmp/BURTlog.$host.$atom.$level

    # get a file descriptor for the dump data
    set dumpfd [open "|/bin/rsh $host -n \"/etc/dump ${level}uf -
        $atom\" 2> $logfile" {RDONLY NONBLOCK}]

    # get a file descriptor for the standard error data
```

```
    set stderrfd [open $logfile r]

    # return the file descriptors
    return [list $dumpfd $stderrfd]
}
```

Similarly, a backup type includes a definition of how to restore something. Again, the administrator creates a Tcl procedure which returns to Burt a data stream to which data from a tape is written. This procedure is associated with a specific type via a call to `register_recover_proc`. An example of a restore procedure is the `unix_restore` procedure, which just writes restored data to a file:

```
proc unix_restore { host atom level } {
    return [open "$host.$atom.restore" w]
}
```

Finally, a backup type includes a procedure which interprets the standard error output from the backup of an item. Each line of standard error output is passed to this procedure; it must parse those lines and determine whether or not an error has occured. The error checking procedure is associated with a type via a call to `register_backupmonitor_proc`. An example is the `unix_check` procedure:

```
proc unix_check { host atom level line } {
    return [regexp {error|Unknown|EXITED|ATTENTION|abort} $line]
}
```

These are simple examples, but they demonstrate the ease with which backup types can be defined. Burt leverages the inherent power of Tcl and the wide variety of extension modules available to support a practically unlimited array of systems. This is what really distinguishes Burt from other backup systems. It gives Burt a distinct advantage over those others, particularly those which uses proprietary client software. Some systems, like Amanda [1], do allow for the creation of new backup types, but the process is fairly restricted since those systems lack a language as expressive as Tcl with which to define new types. With Burt there is no restriction.

2.2 Schedule items

Schedule items are the unit of backups in Burt. Schedule items consist of a *host*, an *atom*, a *level* and a *type*. Of these, the *type* field is the most important, as it specifies to Burt which backup type to use to backup the schedule item. The specific meaning of *host*, *atom* and *level* is dependant on the definition of the type. They are *typeless*, to borrow a programming term. Typically, however, *host* refers to the hostname of the workstation being backed up; *atom* is the partition being backed up; and *level* is the level at which to back up that atom on that host. Normally we interpret *level* as the BSD `dump(1)` level of the backup, but it could be anything.

2.3 Channels

Channels are critical to Burt's backup performance. A channel represents an input stream that is being multiplexed to the tape. Any number of channels can be used simultaneously, which has been shown to be crucial to backup performance [6, 2]. In this respect, Burt is similar to a number of backup systems which allow some form of multiplexing, including Legato Networker [4], Amanda [1], and others.

Some authors argue that writing multiple channels directly to tape is a mistake, because it results in a tape that cannot be read without the use of the software used to create the tape. These authors instead prefer systems that produce a tape which can be read with standard utilities such as `dd(1)`. However, producing such a tape comes at the cost of performance, as those systems must either reject parallelism altogether, or run backups in parallel to an intermediary location, typically a hard disk, and then write the backup images to tape one at a time. The former case is unacceptable for any installation with even a modest amount of data; the latter introduces an unneccessary middleman into the backup process, the cost of which is evident in the speeds attained by these systems. In addition, in the case of Burt, the tape format is so simple that a trivial wrapper around `dd(1)` would be sufficient to read back the data on the tape.

2.4 Filemarks

Filemarks are critical to Burt's recover performance. Most modern tape devices support writing filemarks to the tape. The drive can then position the tape at any of those filemarks at very high speeds. Burt makes use of this feature by writing a filemark before the first block of data from each item backed up onto a tape. Then, when there is a need to recover something from the tape, the user can specify to Burt the number of filemarks to skip before looking for data. Thus the user need not read all the data on the tape, a time consuming operation.

3 How it works

The primitive operations described before can now be given names:

- Specifying what is to be backed up: `schedule`
- Putting data onto a tape: `backup`
- Checking what is on a tape: `readtape`
- Reading data off a tape: `recover`
- Obtaining status information: `status`

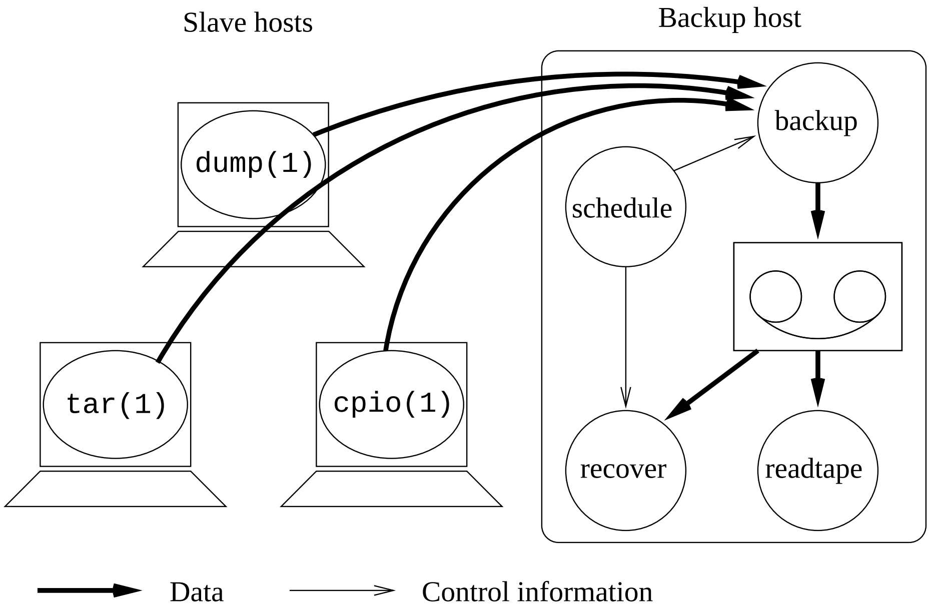

Figure 2: Data flow in Burt – detailed view

The details of how each command works need not be discussed here, but it is instructive to see how they work together to perform backups and recovers. Figure 2 illustrates graphically how Burt's components work together.

3.0.1 Doing backups

The first step in performing backups with Burt is to specify what items are to be backed up. This is done with the `schedule` command, which builds in memory a table consisting of schedule items. This table is called the *schedule table*.

Next, the actual backup process starts with a call to the `backup` command; this call specifies the number of channels to be used, the output and log streams for the backup, and the label for the backup medium. For each channel, `backup` extracts an item from the schedule table and calls the procedure associated with the backup type of that item to start backing it up. As soon as any of these items finishes, `backup` extracts another item from the schedule table and starts the backup of that item, until the schedule table is empty. At the start of any one item, `backup` writes out a filemark to mark the beginning of the data for that item on the tape. As `backup` writes data to the tape, the data is checksummed and the checksums are written out with the data.

Finally, the `readtape` command is used to read back what was written to the tape and verify the itegrity of the data. First, it is used to read the tape label and verify that the correct tape is loaded. Next, it reads the data, including the checksum stored on the tape. It recomputes the checksum of the data, and finally it compares the recomputed value with the value read from the tape. If there is any discrepancy, the data is considered invalid. As it reads the tape, `readtape` creates a *table of contents (TOC)* for the tape, which is a list of all the items on the tape and the number of filemarks that preceed the beginning of each item.

3.0.2 Recovering data

Performing recovers is as straightforward as performing backups. First, the user specifies what items to recover by using the `schedule` command. Any number of items from a single tape can be schedule for recovery.

The recover process then starts with a call to the `recover` command, which specifies input and log channels for the recover, and optionally the number of filemarks to fast-forward the tape past before looking for data to recover. Once the appropriate number of filemarks has been skipped, `recover` starts reading data from the tape. When it finds data from one of the scheduled items, `recover` verifies the checksum and, on success of the checksum, writes the data to the output stream created by the restore procedure associated with the backup type of the item.

4 Backup concerns

There are some concerns – autonomy, security, accidental tape overwrites, and reliability – that are common to backup administrators, judging by the papers written on backup systems [1, 8, 5]. Here we will discuss those concerns within the scope of Burt.

4.1 Autonomy

Everybody wants their backups to run unattended. Using Burt, this is no problem. The administrator can easily make a Tcl script which schedules items for backup and then calls `backup` to write them to tape. When the run is complete, the script can send a message to the administrator with summary information about the run. The script can be run at a scheduled time using `cron(1)` or `at(1)`.

4.2 Security

Many people are concerned with backup security, and rightly so. There are two aspects to this concern. First is authentication concerns. Obviously we don't want an arbitrary party to be able to ask our computers for backup data. We must make sure that the person running the backup and receiving the data is authorized to do so. Since the administrator defines the backup types, they are free to use whatever level and method of authentication desired. We use `Kerberos V rsh` to access most of our computers, and `ssh(1)` to access others. This affords us a high degree of authentication security. In addition, we can encrypt the data stream from the remote host to the backup host in order to protect against network traffic snoopers.

Second is data security. If your tapes are stolen, you must ensure that the hostile party is unable to retrieve data from them. Again, since the administrator defines the backup types, they can easily create a backup type which pipes backup data through an encryption program of choice before forwarding the data to Burt. The data written to the tape would thus be encrypted and secure in case the tape were to fall into hostile hands. Of course, this introduces some other problems, such as key management, but those are beyond the scope of this paper.

4.3 Overwriting tapes

Many authors are quite worried about the risk of overwriting the wrong tape. The reason is clear: if the wrong tape is overwritten, a good deal of backup data could be irrevocably lost. The systems presented by those authors thus go to great lengths to verify that the tape in the tape drive is the correct one according to the particular backup policy in use.

With Burt, this is easily done. The administrator need only add a few lines of Tcl code to the backup script to make use of Burt's `readtape` command, which can return the tape label. The script could then verify that the correct tape is indeed loaded. Alternatively, the administrator could make a separate utility to verify the tape early in the day, like the `amcheck` program distributed with Amanda [1].

4.4 Reliability

Like security, reliability is a multi-layered problem. First is the problem of getting reliable data from the client backup program. As other authors have discovered, many common backup programs, including `dump(1)`, have serious problems dealing with some situations [10]. Since Burt is dependant on external backup programs to do the work of backing up items, the data it puts on tape is only as reliable as the data obtained from those programs.

Second is the problem of ensuring that the data written to tape can be retrieved from tape. Burt deals with this problem by computing checksums on all backup data written to tape and then verifying those checksums whenever the data is read from the tape. If Burt uses the Fletcher checksum.

5 A Simple Example

Burt can seem overwhelming at first glance, but setting it up can be very easy. Here is an example of how Burt might be set up at a small site that backs up only a few computers, all of the same type.

5.1 The unix backup type

First, we must define a backup type. We wil call it the `unix` type, because we will use it to backup Unix systems. The procedure definitions for this type will be exactly those shown during the discussion of backup types: `unix_dump`, `unix_check` and `unix_restore`.

5.2 The backups.tcl script

Next, we must create a Tcl script to use the Burt commands to schedule and backup items. This script, which we will call `backups.tcl`, might look like this:

```
#!/usr/local/bin/tclsh

# Load the Burt library containing the new Tcl commands and
# the script library containing the unix type definition
package require Burt

# associate the unix_dump procedure with the unix type
register_backup_proc unix unix_dump

# associate the unix_check procedure with the unix type
register_backupmonitor_proc unix unix_check

# schedule some items to be backed up
# (usage: schedule add host atom level type)
schedule add pigpen.cs.wisc.edu / 0 unix
schedule add gazoo.cs.wisc.edu / 0 unix
schedule add stall.cs.wisc.edu / 0 unix

# open the tapedrive and logfile
set tapefd [open /dev/rmt/0b w+]
set logfd [open /tmp/backup.log w]

# start this backup with 3 channels and the label "full backups"
# (usage: backup start channels outputchannel logchannel label)
backup start 3 $tapefd $logfd "full backups"

# wait for the backup to finish
vwait burt_complete

# close the tape drive to rewind it, then reopen for readchecks
close $tapefd
set tapefd [open /dev/rmt/0b r]

# do readchecks on the tape and put the resulting TOC in the log
# (usage: readtape toc tape log)
```

```
puts $logfd [readtape toc $tapefd $logfd]

# close the tapedrive and logfile
close $tapefd
close $logfd

# mail the log to the operator
exec /usr/ucb/mail -s "Backup log [clock format [clock seconds]]"
    operator < /tmp/backup.log
```

5.3 Putting it all together

Now to do full backups of the three items scheduled in this script, the operator need only run `backups.tcl`. Of course, this is only a simple example; a real system would be more elaborate. For example, we would want to add error checking and clean-up code, and we would want to store the logfile somewhere other than /tmp. Nevertheless, this gives an idea of how the pieces of Burt fit together to make Burt work.

6 Show me the money! – Burt in practice

Since August 1, 1997, we have been using Burt as our backup system at the University of Wisconsin, Madison, Computer Science Department (UWCS). We backup about 500 GB of data from 350 workstations and 13 AFS file servers to a fleet of 29 4mm DDS-DC tape drives, which we have named A through CC.

We use a two week epoch cycle, so that everything is guaranteed to be fully backed up every two weeks. The tapes containing epoch data for the first epoch cycle of each month are pulled out and stored for one year. Tapes for the first epoch cycle of the first month after the end of each semester are pulled out and stored for two years. All other tapes are recycled monthly.

In addition to epoch backups, we do *compositional* backups. This means that every day, we either do a full backup of a given item or we do a backup of the data that has changed since that item's last full backup. In BSD `dump(1)` terminology, we do level 1 dumps every day that we don't do a level 0 dump. This leads to a considerable amount of redundancy in terms of what data is actually on tapes. However, that redundancy insulates us from the loss of individual incremental tapes, and it greatly simplifies restores: at most we need only read two tapes to fully restore anything, a fact which our operators greatly appreciate.

We will detail our implementation of Burt and provide some performance statistics.

6.1 UWCS backup types

As noted before, Burt uses the notion of types to determine how to backup different systems. At our site, we have defined several backup types: AFS, HPUX, Linux, NeXT, UFS, Ultrix, and XFS. Each of these represents a different kind of system that we back up. The two most commonly used are AFS, which we use to dump our AFS space volume by volume; and UFS, used to dump our Solaris workstations. The Tcl/Tk code defining the UFS backup type is:

```
##################################################
# Function definitions for ufs backup type
##################################################

proc ufs_dump {host atom level} {
    global sessionID tmpdir
    set hostlogfile $tmpdir/BURTlog.${sessionID}.$host
    append hostlogfile [info cmdcount]
    set dumpfd [open "|/s/std/bin/rsh $host -n \"/etc/dump \
        ${level}uf - $atom\" 2> $hostlogfile" {RDONLY NONBLOCK}]

    set stderrfd [open $hostlogfile r]
    return [list $dumpfd $stderrfd]
}

proc ufs_check {host atom level line} {
    return [regexp {error|Unknown|EXITED|ATTENTION|abort} $line]
}

proc ufs_recover {host atom level} {
    set filename "$host"
    if { [string compare $atom "/"] == 0 } {
        append filename ".root"
    } else {
        regsub -all {/} $atom "." newatom
        append filename "$newatom"
    }
    append filename ".$level.burt"
    return [open "$filename" w]
}
```

`ufs_dump` defines the procedure called by Burt to begin dumping an atom of type `ufs`. It uses `Kerberos V rsh` to connect to a host, and `dump(1)` to get backup data from the host. When an operator wants to restore a `ufs` item from a Burt tape, Burt uses `ufs_recover` to do so. In this case, we just write the data from tape into a file on disk, though this function could be more elaborate. For example, it could open a pipe directly to the restore program and bypass the disk altogether. We opted for the method shown in order to avoid problems associated with accessing a remote tape drive.

As mentioned, we also have a substantial amount of filespace in AFS. The `afs_dump` procedure uses the `vos(1)` command, which can dump an AFS volume to standard output, to backup that filespace volume by volume.

6.2 Groups

We use the concept of dump groups to organize the things that we have to dump and to keep them manageable. Groups are lists of items to be backed up to a single tape. Each group can be classified based on two criteria: whether it is an epoch (level 0) or an incremental (level 1) backup; and whether it is made up of AFS or workstation dumps. As a general rule we avoid mixing backup levels on a single tape, because it reduces the number of tapes we have to pull out and store each month. The exception is the daily redo group, which is made up of miscellaneous items which failed during the nightly backups or which are in need of special consideration. Groups are named for the tape drive on which they normally run, one of A through CC, and for the date on which they are supposed to run.

AFS groups are load balanced each night by a custom-made Perl script, and an effort is made to backup entire partitions of each server to one tape, and entire servers to only one group letter (ie, all partitions from the server dogbert will be epoched on the various AA groups). This has some pros and cons. In its favor is the good locality of data that it gives us – if a partition from a server crashes, we read a minimal number of tapes to restore it. However, since all the data for a given group is coming from only a couple of physical disks, backup throughput for these groups plummets. Note that this low speed is not due to any shortcoming in Burt. Rather, it is a result of our particular setup. Minimal load balancing is done by hand on the workstation groups.

6.3 Scripts

There are numerous scripts that we make use of to implement Burt. We will only detail the most important ones.

6.3.1 burtcentral

`burtcentral` starts multiple Burt backup processes on various hosts and tape drives at different times. At around midnight every night, `burtcentral` starts via `cron(1)` and loads a default schedule file. Each line of a schedule file lists a starting time in 24-hour format, a tape drive, and a group name. Our default schedule file reads in part:

```
0100    A       A
0101    B       B
0102    C       C
```

```
.
.
.
0400    CC       CC
```

If the number of the group is not explicitly specified, `burtcentral` assumes the groups meant to be run today. After loading the schedule file, `burtcentral` waits for the starting times. As each time is reached, all groups listed are started on the specified host and drive, via a `Kerberos 5 rsh` to the appropriate host machine.

This scheme does creates a single point of failure – if the machine on which `burtcentral` is meant to run crashes, any unstarted backups will not be started. However, having tried other methods for automatically starting each of the 29 backups each night, we have found that the ease with which we can now manage the backups outweighs the risks.

6.3.2 burtbackup

Each backup is run using the `burtbackup` script, which first obtains a unique session ID to identify the tape and backup session. This session ID consists of the value of clock seconds followed by the process id of the `burtbackup` process. While this doesn't quite guarantee a unique session ID, it works well enough in practice – since we recycle tapes on a monthly basis, the session ID need only avoid conflicting with other session IDs from the same month. The session ID is written to the tape, and the log for the run is named with the session ID. This allows us to easily match tapes to logs in the event that the physical label on the tape is lost or incorrect.

After obtaining a session ID, `burtbackup` reads a group file and schedules each listed item. After everything is scheduled, `burtbackup` calls the Burt command `backup`, passing it file handles for an opened tape drive and log file, the number of channels to use during the run, and the label for the tape. During the course of the run, a good deal of information is output to the log, showing the starting and ending times of each item that is backed up, the number of bytes from each item, and all the execution dialog from the backup of each atom. A typical log looks like this:

```
BURT: Starting Backup test on mach5:/dev/rmt/1hb.
    Thu Jun 11 15:22:35 CDT 1998
BURT: Started rizzo-wizz / 1 ufs [15:22:40 06/11/98]
BURT: Started pigpen / 1 ufs [15:22:40 06/11/98]
BURT: Started stall / 1 ufs [15:22:40 06/11/98]
BURT: Started 8 channels [15:22:40 06/11/98]
.
.
.
stall / 1 ufs:   DUMP: DUMP IS DONE
BURT: Completed stall / 1 ufs, 39 blocks, totaling 1212416 bytes
```

```
    [15:22:46 06/11/98]
rizzo-wizz / 1 ufs:   DUMP: Mapping (Pass I) [regular files]
pigpen / 1 ufs:   DUMP: Writing 32 Kilobyte records
```

Once all scheduled items have completed backups, `burtbackup` rewinds the tape and uses `readtape` to perform readchecks. This serves two purposes. First, it verifies the integrity of the data on the tape. Second, it generates the TOC for the tape, which we use to track what data is on which tape.

Finally, after the readchecks have completed, `burtbackup` sends mail to the operator containing summary information about the run and the output from the readchecks. If any problems were encountered during the backup, either with individual items or with the tape in general, or if errors were encountered during the readchecks, they are noted in the mail. Typical mail looks like this:

```
BURT: Backups completed Thu Jun 11 15:22:54 CDT 1998.
Runstate backup-complete
Label test.897596555.3154
Scheduled 3     Started 3       Finished 3
Tapes 1
Channels 8
TotalMegabytes 3.59
StartTime Thu Jun 11 15:22:37 CDT 1998
CompleteTime Thu Jun 11 15:22:54 CDT 1998
Bytes/Sec 221665.875000

BURT: Starting read checks [15:22:55 06/11/98]
BURT: TOC {{rizzo-wizz / 1 ufs} 1 3} {{pigpen / 1 ufs} 1 2}
{{stall / 1 ufs} 1 1}
BURT: Run Completed.  Thu Jun 11 15:23:08 CDT 1998.
```

6.3.3 burtfinder and burtrecover

When an operator needs to recover data, they first determine which tape contains the data they need by using `burtfinder`, which searches a digest version of the logs for the items needed. Once they have found the appropriate tape, they use `burtrecover` to pull the data off the tape. `burtrecover` allows the user to interactively specify items to recover and the number of filemarks to skip for each item before looking for data. `burtrecover` schedules each item for recovery using the same scheduling mechanism that `burtbackup` uses and determines the minimum number of filemarks to skip, since only one fast forward can be performed per recover session. Finally `burtrecover` calls the Burt command `recover`, passing it the filehandles for the opened tape drive and log file, and the number of filemarks to skip.

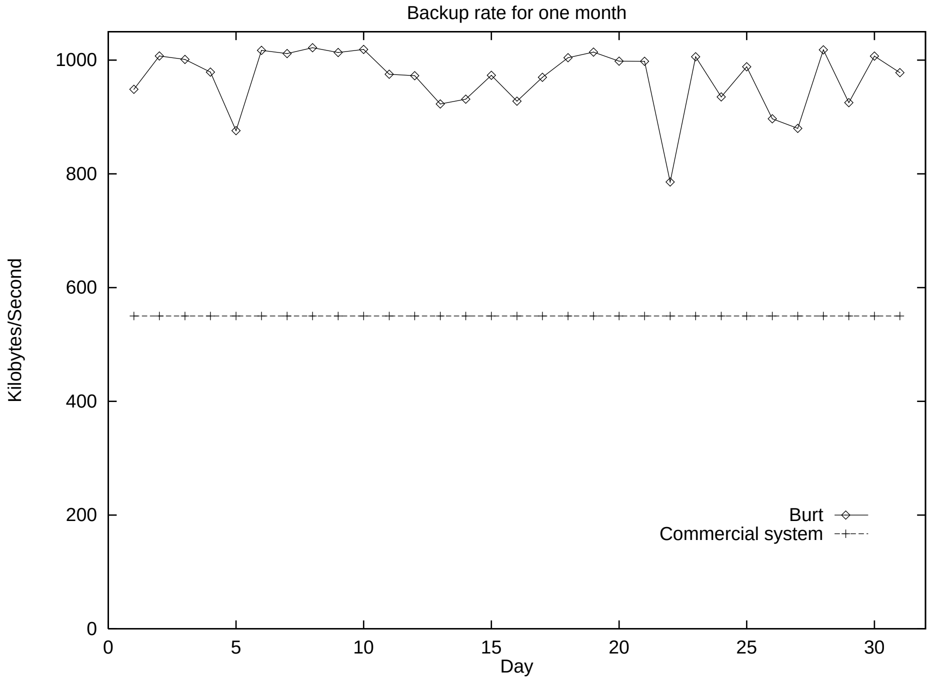

Figure 3: Performance data

6.4 Performance

Figure 3 is a plot of the rate of backup in kilobytes per second for the backup of a particular group over a period of one month. The backups were done on an HP 1533A connected to a dual processor SPARCStation 10/50 with 64 MB of RAM, a 500 MB hard disk and a 10bT Ethernet connection. The drive's native transfer rate is about 510 kilobytes/second. With hardware compression, transfer rates of rougly 1020 kilobytes per second are possible. Our tape drive was configured to use hardware compression. About 2 gigabytes of data was backed up each night.

With Burt, the drive routinely achieves transfer rates of about 850 kilobytes to 1000 kilobytes per second while doing backups. Burt clearly does a good job of keeping the tape drive streaming, even on a moderately powered backup host. In addition, there is no need for a large holding disk for speedy operation, as is required by systems like Amanda [2].

These performance results compare favorably with recent published benchmarks of some commercial systems, such as Seagate Backup Exec and and Stac Inc.'s Replica [7]. This setup used a 266 MHz Pentium II with 128 MB of RAM, two 4GB hard drives and an HP SureStore 8 DDS2 4-mm DAT drive for the backups. The tape drive used is comparable to the HP 1533A used in our own setup. As in our setup, these tape drives were configured to use hardware compression [3]. With this hardware, the fastest of the benchmarked systems clocked in at around 550 kilobytes per second.

7 Future plans

Burt fills our needs, but we feel that a few enhancements could make it even better.

First, Burt should not assume the use of magnetic tape as the backup media. There is no real reason that Burt could not be used with other types of media. However, some parts of the Burt engine assume the existence of tape-like control mechanisms on the output stream. That code could be easily changed to use more generic code. However, we feel that if we are going to support different types of media, we should exploit the features of those media which make them different from tapes. For example, if we were using disk like media, we would want to take advantage of the random-access nature of the media. Determining how best to make use of new types of media will require more research and experimentation.

Along the same lines, we would like to provide support for utilizing multiple tape drives from a single Burt backup process. This would enable us to do things like tape striping, redundant tape creation, and auto-rollover.

Finally, we would like to allow administrators to control the order in which schedule items are backed up. The pseudo-random order used currently works well when the sizes of the items are roughly equal. However, when there is a large variance in the sizes, the order in which items are backed up can have a tremendous impact on performance [2]. Being able to explicitly specify the order based on the size of the items could be a great boon. Ordering policies based on other criteria, such as the sources of the data, could be useful as well.

8 Conclusions

After using Burt in a real-world situation for almost a year, we feel that we successfully achieved our goals. We have created a backup system with flexiblity that far surpasses that of any other system we have encountered. It is fast and easy to use, and it is very scalable. We are confident that Burt will be able to handle any future backup challenges presented to us.

References

[1] James da Silva and Ólafur Gudmundsson. The Amanda Network Backup Manager. In *Proceedings of the Seventh Large Installation Systems Administration Conference*. The Usenix Association, November 1993.

[2] James da Silva, Ólafur Gudmundsson, and Daniel Mossé. Performance of a Parallel Network Backup Manager. In *Proceedings of the Summer 1992 Techincal Conference*, pages 217–225. The Usenix Association, June 1992.

[3] Rob Daly. Personal correspondence, June 1998.

[4] Legato. *Networker 5.1 for UNIX and Windows NT*, March 1998.

[5] Melissa Metz and Howie Kaye. DeeJay – The Dump Jockey: A Heterogeneous Network Backup System. In *Proceedings of the Sixth Large Installation Systems Administration Conference*. The Usenix Association, October 1992.

[6] W. Curtis Preston. Backup Techniques – Dynamic Parallelism. *SysAdmin*, February 1997.

[7] Steve Rigney. Back Up Against Disaster. *PC Magazine*, 17(6):227 – 236, March 1998.

[8] Steven M. Romig. Backup at Ohio State, Take 2. In *Proceedings of the Fourth Large Installation Systems Administration Conference*. The Usenix Association, October 1990.

[9] Dan Romike. DK I/O System, March 1987.

[10] Elizabeth D. Zwicky. Torture-testing Backup and Archive Programs: Things You Ought to Know Burt Probably Would Rather Not. In *Proceedings of the Fifth Large Installation Systems Administration Conference*. The Usenix Association, 1991.