Burt: The Backup and Recovery Tool

Eric Melski

November 23, 1999

Abstract

Burt is a freely distributed parallel network backup system written at the University of Wisconsin, Madison. It is designed to backup large heterogeneous networks. It uses the Tcl scripting language and standard backup programs like `dump(1)` and GNUTar to enable backups of a wide variety of data sources, including UNIX and Windows NT workstations, AFS based storage, and others. It also uses Tcl for the creation of the user interface, giving the system administrator great flexibility in customizing the system. Burt supports parallel backups to ensure high backup speeds, and checksums to ensure data integrity. The principal contribution of Burt is that it provides a powerful I/O engine within the context of a flexible scripting language; this combination enables graceful solutions to many problems associated with backups of large installations. At our site, we use Burt to backup data from 350 workstations and from our AFS servers, a total of approximately 900 GB every two weeks.

1 Introduction

In the past several years, network installations have become increasingly large and complex. These networks consist of more workstations, and those workstations, particularly in research environments, run a wider variety of operating systems. In addition, the recent dramatic decrease in the price of hard disk drives has made it feasible for users to store large amounts of data on their workstations. Many users also want the best of both the *datafull* (all data stored on local disks) and the *dataless* (all data stored on fileservers) workstation models. Finally, system administrators are under constant pressure to decrease the amount of time during which backups run. Together, these points make backups more difficult than ever.

As an example, the University of Wisconsin, Madison, Computer Sciences Department has over 350 workstations, each of which runs one of seven different flavors of UNIX. Some of these workstations have 8 gigabytes of data or more stored locally. In addition, we have over 500 gigabytes of data stored on our AFS [4] fileservers. We are given only six hours each night during which to perform backups.

Burt, the "Backup and Recovery Tool," was developed to address these issues. We had several specific goals in mind when we developed Burt, some of which are common to many backup systems:

- Store data to long-term media
- Provide a means by which to track stored data
- Retrieve stored data
- Ensure the integrity and reliability of stored data, to the degree allowed by the storage technology
- Ensure the speedy backup and recovery of data, to the degree allowed by the storage technology
- Provide a mechanism to automate the backup of data
- Secure backups, to a reasonable degree, from likely attacks
- Support a variety of storage technologies

- Support network backups
- Provide an easy-to-use interface

Other goals were more specific to our needs. These related to the quantity and distribution of data at our site, and the heterogeneity of our installation:

- Support the backup of large aggregate amounts of data, on the order of hundreds of gigabytes, and gracefully scale to accommodate growth.
- Support the backup of large amounts of data from a single source; in particular, allow for the backup of atomic sources that are larger than the capacity of a single tape or other storage element.
- Support the backup of data from a large number of sources, on the order of tens of thousands of sources, and gracefully scale to accommodate growth.
- Support backups of data from many different kinds of sources; in particular, support backups of multiple flavors of UNIX workstations and of AFS data.
- Allow easy integration of future kinds of data sources; in particular, support backups of all future operating systems and architectures with minimal changes to the backup system.

The backup system we had been using, DK [12], did not meet these needs adequately. We determined that the best way to address all of these needs was to construct a new backup system. We built that system in two layers, one compiled and one scripted, using C for the compiled language and Tcl [10] for the scripted language. This model has been used in many modern applications, such as word processors and other office software. We found that it could be applied to backup software as well, where it gave us both performance and flexibility. In particular, the Burt I/O engine and supporting scripts provide:

- Support for the backup of a wide variety of data sources, and easy integration of new types of sources, by using standard backup programs like `dump(1)` and others
- Fast backups, often at or near the maximum speed of the storage device
- Support for large atomic sources and large aggregate amounts of data
- Support for user-interfaces tailored to a particular site
- Easily extended functionality, through the Tcl/Tk core and extensions

This paper discusses the design of Burt and its features as seen by a system administrator. We will begin with a brief overview of what a backup system is, in section 2. Sections 3 and 4 describe the Burt Architecture and user interface. Section 5 contains comparisions to other backup systems. Section 6 describes our experiences with Burt at the University of Wisconsin, Madison, over the past two years. Finally, Section 7 discusses our future plans.

2 Overview

A *backup system* consists of many subsystems. At the coarsest level, those subsystems may be grouped into a hardware component and a software component. We will largely ignore the hardware component for the remainder of this paper.

The software component may be further divided into an agent responsible for moving data to and from the hardware, a mechanism for tracking what data is on each backup media volume, and a user interface. By itself, the Burt engine is only the first of these: a data moving agent. The other pieces are Tcl scripts that a system administrator creates to suit the needs of a particular site. These scripts handle all the other software portions of a backup system, including tracking data and providing a user interface.

With this in mind, Burt is designed to enable a system administrator to backup a large collection of data, from a large number of dissimilar data sources to a single output destination. In the context of this paper, *data source* includes disk partitions on workstations, AFS volumes, individual files, databases, and any other unit of data for which a data stream can be created. The output can be written to any location, including files, sockets, and tape drives.

3 The Burt Architecture

Burt consists of two components: a compiled component, and Tcl script based *backup types*, which are bound to the compiled component at run time. We chose Tcl as the scripting language for Burt because it is easy to extend, and we felt that it was easy to learn. In addition, the availability of the Tk graphical toolkit extension assured us that we would be able to easily make graphical user interfaces for our system.

3.1 The Engine

The compiled component, which we will refer to as the engine, extends the scripting language Tcl [10] and consists of roughly 7,500 lines of C. A special effort was made to adhere to POSIX.1 standards [6] to ensure that the engine could be easily ported to many UNIX systems; we have successfully ported Burt to Solaris, Linux, and SunOS. When loaded into a Tcl interpreter, the engine adds a few commands that serve as an API for controlling the engine:

- `backup`, for initiating backups
- `recover`, for initiating recoveries (restores)
- `schedule`, for scheduling data sources for backup or recovery
- `readtape`, for verifying checksums on tape and building a list of data sources on tape
- `status`, for obtaining runtime status information

The engine has a number of significant features, but two are particularly important. First is has multiplexing capabilities, and second, it is "dumb."

3.1.1 The multiplexing engine

A key feature of the Burt engine is its multiplexing capability: it receives as input some number of data streams, packetizes and checksums the data, and outputs a single stream containing those packets (Figure 1). Each packet contains a header that indicates the origin of the data, the sequence number of the packet, and other information; and a block of data from an input stream.

The ability to multiplex is important because it enables the system to achieve much higher overall performance than is generally possible when using a single input stream. Other authors have quantified the performance benefit [2, 11]; one paper showed a better-than-linear speed increase as the number of input data streams increased [2]. Essentially, the speed increase comes from the system's ability to compensate for slow input streams by reading data from other input streams – the more input streams in use, the more likely it is that any one of them will have data waiting to be read at a given instant. Multiplexing does add complexity to the system, but we feel that the performance benefits far outweigh the cost of that complexity.

Of course, the engine has demultiplexing capabilities as well. When reading a tape written by Burt, the engine receives a data stream composed of interleaved data from many data sources. The engine only extracts data for those data sources that an operator is interested in recovering from tape (Figure 2).

3.1.2 The "dumb" engine

The second critical feature of the engine is that it is "dumb." It does not know where the input data is coming from, nor where it is going to. The knowledge about where the data is coming from is encapsulated in *backup types*, the second component of the Burt engine. The knowledge about where the data is going to is managed by the user interface, and is specified by the operator or administrator. This separation allows Burt to support a wide variety of data sources and storage devices, because the administrator need not change the compiled portion of the system to add support for any type of data source or storage device. The administrator only needs to extend or update the Tcl based backup types or user interface, as described later.

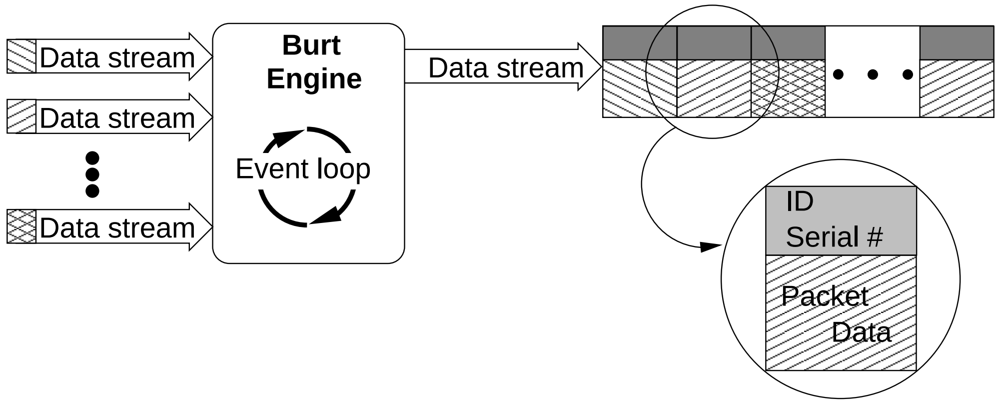

Figure 1: Data flow in Burt during backup, as seen by the engine.

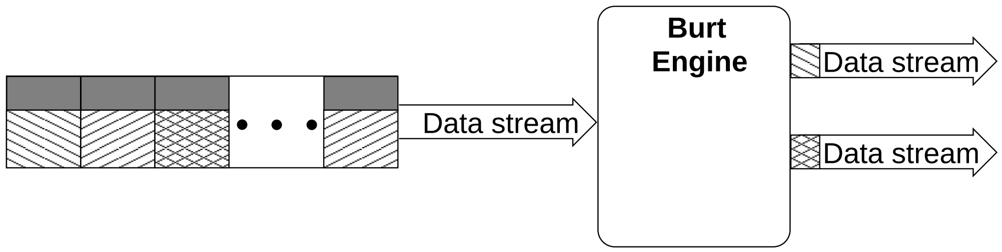

Figure 2: Data flow in Burt during recovery, as seen by the engine.

3.1.3 Other important features

A number of other engine features are worthy of mention. One such feature is that it writes directly to the tape, instead of writing to an intermediary storage location first, as some systems do [2]. This allows Burt to reduce significantly the total time required to perform a backup.

The engine computes a checksum for every packet written to tape, and writes the checksum out with the packet. The checksum enables Burt to verify that the data read from the tape is identical to the data written to the tape. This provides protection from media corruption. If the engine finds an incorrect checksum when reading the tape, it notifies the operator. The engine uses a 32-bit extension of the Fletcher checksum [3], which we chose because the algorithm is fast, highly reliable, and easy to implement.

The engine allows backups to span tapes. This is particularly important for data sources that are larger than the capacity of a single backup tape. It also allows the operator to append data to a tape previously written by Burt. This can be useful at sites that have aggregate data sizes smaller than the capacity of a single backup tape. Those sites can put backups from several successive days on a single tape, rather than wasting the additional space.

Finally, the engine uses filemarks on tape to signify the start of data from each data source. These filemarks can be used later to fast-forward the tape directly to the point at which the requested data is located. This can greatly improve the time required to recover a particular data source from tape, because the engine can skip all of the data preceeding the relevant data rather than reading it. This is merely an optimization; without the filemark information, the engine can still recover all the data from the tape, but it will take more time to do so.

3.2 Backup Types

The second component of the Burt engine is the backup types. These are Tcl functions that serve as an interface between the engine and a particular type of data source. A site will have one backup type for each type of data source. For example, we use eight backup types: one for each flavor of UNIX used at our site, and one for AFS. In the terminology of object oriented programming, Burt has a single backup type interface, and each type of data source has a unique implementation of that interface. Figure 3 illustrates the concept.

A backup type consists of several Tcl functions that encapsulate the knowledge about how to backup a type of data source. Burt requires that a backup type implement three functions: backup, monitor, and recover. Each function is *registered* with the engine, meaning that it is associated with a particular type of data source, and is then called by the engine as needed at runtime. For reference, the *solaris* backup type, which we use to backup the majority of our workstations, is included in Appendix A.

3.2.1 The backup function

The backup function is responsible for connecting to and initiating backups of a particular data source. When the engine is requested to backup a data source, it invokes the backup function that is registered for the type of that data source. The engine passes parameters to the backup function that indicate the particular data source to be backed up. The function initiates a backup, typically by exploiting the native backup program of the data source. For example, the backup of a UNIX workstation could use `rsh(1)` to connect to the workstation, and `dump(1)` to perform the backup of a disk partition. The backup function also prepares the standard error output from the native backup program for processing by the engine. Finally, it returns two data streams to the engine, one along which the backup output from the native backup program is transmitted, and one along which the standard error output is transmitted.

3.2.2 The monitor function

The monitor function is responsible for examining the standard error output. As the backup of the data source proceeds, the engine feeds all standard error output received into the monitor function registered for the type of the data source. If the monitor function indicates that an error has occurred, the engine

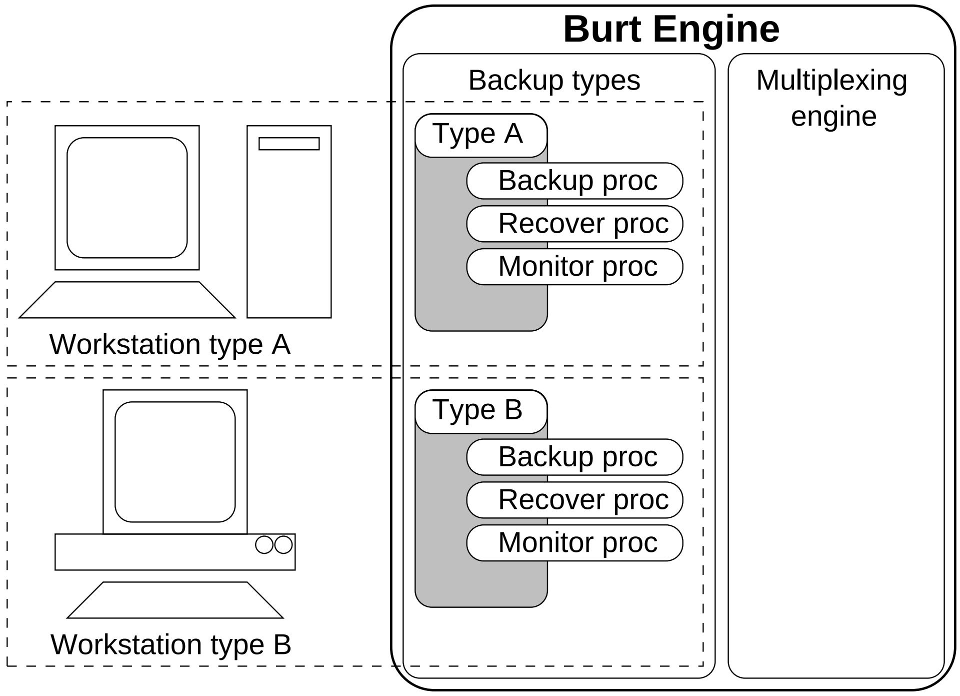

Figure 3: Each type of system has a unique backup type implementation, but every backup type presents the same interface to the engine

terminates the backup of the data source in question and logs that action along with the error message.

3.2.3 The recovery function

The recovery function is responsible for creating a data stream to which recovered data will be written. When the engine is requested to perform a recovery of previously backed up data from a particular data source, it calls the recover function registered for the type of that data source. The engine passes the recovery function information indicating the original origin of the data being recovered, and the recovery function creates a data stream to which the engine can write recovered data. As the engine reads data from the backup media, it writes each packet that is from the requested data source to the data stream created by the recovery function.

Typically, the data stream will be a standard UNIX pipe directly into the native recovery program of the data source from which the recovered data was backed up. Alternatively, the data may be written to a file, which is useful when the operator needs to browse the data and does not want to reread the tape.

3.2.4 Flexibility of the backup type architecture

The backup type architecture is one place where the benefits of separating the system into a compiled and a scripted component are clear. Because the backup types are not compiled, they are easy to modify and extend, and can even be altered at runtime. Certain features of Tcl make it very suitable for use in this context; in particular, the language's sophisticated process control capabilities and extensive string and regular expression operations make it easy to construct backup types for a wide variety of data sources.

The philosophy of exploiting the native backup and restore programs of a particular data source is also essential to Burt's ability to support many kinds of systems. This type of setup enables the administrator to leverage existing tools rather than requiring them to create or maintain programs for performing backups. In addition, it reduces the amount of time between the introduction of a new type of data source and the introduction of backup system support for that data source. The administrator does not need to install anything on the client, provided it already has a native mechanism for performing backups.

In practice, it has proven to be easy to add support for new kinds of data sources. When we originally began using Burt in August 1997, we supported only five varieties of UNIX at our site; since that time, we have added support for two other varieties. In both cases, adding support consisted of less than twenty minutes of work creating a new backup type, a few tests to verify the proper function of the new backup type, and then bringing the backup type "online." An administrator who is reasonably comfortable programming in Tcl should have no difficulty creating or modifying backup types.

4 The User Interface

In a sense, the user interface is what brings it all together: the engine, the backup types, and the operator. The engine adds a few commands to a standard Tcl interpreter; the user interface is responsible for making use of these new commands to make backups and recoveries happen. This is the second area where the separation of the system into a compiled and a scripted componant has proven to be a great benefit. Because the interface is composed of scripts, it can be easily altered. Perhaps even more importantly, it is easy for each administrator to customize the interface to the particular needs of their installation. Tcl, along with the Tk graphical toolkit, is a good choice for this task. It is easy to create sophisticated graphical user interfaces with Tcl/Tk, a fact which is demonstrated by the wide variety of programs that use Tcl/Tk to create their user interface.

As with backup types, an administrator who is reasonably comfortable with Tcl should have no trouble making user interface scripts. At minimum, these scripts should have some means of starting backups of some list of items, some means of starting recoveries of some list of items, and some means of finding the tape on which a particular item is stored. At our site, we have chosen to follow the UNIX model of breaking a system into many small components.

Thus we have separate scripts for performing backups, recoveries, and searches. We also have a variety of "utility" scripts, including runtime status displays and easy-to-use schedule list editors. These scripts range from simple batch-oriented programs to fancy graphical, interactive user interfaces. In total, we have written roughly 3,000 lines of Tcl/Tk code.

4.1 The backup script

A typical backup script begins by determining what data sources are to be backed up to a given tape. At our site, this is accomplished through statically generated *group* files, which are simply lists of data sources. The operator or batch processor specifies which group file should be read. The script then schedules each data source with the engine, via the `schedule` command. Next, it must register the backup type functions for each type that has been scheduled. Then, the script must open an output stream, typically a tape drive; again, at our site, the operator or batch processor specifies which tape drive to use. Finally, the script instructs the engine to begin backups, via the `backup` command.

Once the backups of all scheduled data sources have completed, the script must make a record of the data sources that have been written to the tape, in order for the administrator to be able to locate data from particular data sources. The normal way of doing this is via the `readtape` command, which builds a list of all the items on the tape, as well as tests the checksums for every packet on the tape to verify that the data has been correctly stored. The list is then stored in a database for future reference. Finally, the backup script will mail the administrator with information about the run. Pseudo-code for this entire process might look like this:

```
call queue_data_sources()
call register_backup_types()
output = call open_output_stream()
call start_backups(output)
wait while backups are not complete
backup_statistics = call get_backup_stats()
table_of_contents = call verify_tape()
call close_output_stream(output)
call write_to_database(table_of_contents)
call mail_operator(backup_statistics)
```

Our backup script can be found on the World Wide Web at [7].

4.2 The recover script

A typical recover script, like a backup script, begins by determining what data sources are to be recovered. We use an interactive script that allows the operator to enter data sources to recover. The script then schedules each of these data sources with the engine, via the `schedule` command. Next it registers the backup type functions, particularly the recovery functions, for each type that has been scheduled. Then, the script must open an input stream previously written by Burt, typically a tape; the operator specifies which tape drive to use. Finally, the script instructs the engine to begin recoveries, via the `recover` command.

The engine will first fast-forward past as many filemarks as it is instructed to, and then begin to read each packet on the tape. The use of filemarks is optional in this case; if they are used, they can dramatically reduce the amount of time required to recover data from the tape. If a packet for a scheduled data source is found, the engine writes the data in the packet to the output stream created for that data source by the recovery function. Our recover script can be found on the World Wide Web at [8].

4.3 The search script

The search script need only provide the administrator with a means for determining what tape contains the data from a particular data source. We store that information in a simple text database; accordingly, our search script is basically a wrapper around `grep(1)`.

Our search script provides limited additional functionality, such as allowing the operator to view the log associated with the creation of a particular tape, and determining the physical location of a particular tape. Our search script can be found on the World Wide Web at [9].

4.4 Other scripts

We use a number of other scripts in addition to the backup, recovery, and search scripts. For example, we have a script that reads only the Burt label from a tape and displays it; we have a script that displays the current speed and percentage complete of a running backup; and we have a script that coordinates the start of nightly backups in batch mode. Most of these scripts are between 10 and 300 lines of Tcl/Tk code.

5 Comparisons to other systems

There are several backup systems that aim to fill the same needs as Burt. This raises the question, why did we create something new rather than using an existing solution?

The answer is plain: none of the systems we examined filled all of our needs, or they did not fill our needs as well as we would have liked. The primary contribution of Burt is that it meets the needs established earlier, and we feel it does so better than other systems. With respect to each system, Burt is more flexible, more extensible, more scalable, or a combination of these. It is also less costly than commercial systems, a factor that has a significant impact at some sites.

In this section, we examine other backup systems, highlighting specific similarities and differences. This is not meant to be a comprehensive comparison of backup systems; we have simply chosen a few systems that we believe other administrators will be familiar with. A more complete comparison of backup systems can be found in [1], though that comparison predates Burt. The systems we consider here are

- Amanda, from the University of Maryland [2, 1]
- Legato Networker [5]

5.1 Common features

All of the systems have some features in common. For example, they each support backups of a range of types of data sources, and they each support backups of several data sources in parallel. However, the implementation of these features varies greatly from one system to the next.

5.1.1 Support for different types of data sources

One feature that is implemented very differently by the systems is their support for different types of data sources. Although they each support backups of a range of types of data sources, that range is significantly different in each product due to the implementation. Legato Networker uses proprietary backup programs for each type of data source. Consequently, if Legato has not yet created a backup program for a particular type of data source, Networker cannot support it. Presently, Networker's support for various types of data sources is fairly broad, including Windows clients and a variety of flavors of UNIX, but there is no support for AFS backups, which is critical for our site. This implementation can lead to delays between the introduction of a new type of data source and the introduction of support for backups of that type of data source. It should be noted that the use of custom backup programs has at least one important advantage: it allows Networker to create very detailed indices of the contents of each backup tape. These indices can be made at the file level, which makes it easier for administrators to locate a single file on a backup tape.

Amanda uses standard backup programs for each type of data source, including programs like BSD dump and GNUTar. This leaves the system well positioned to support a wide range of systems, provided that they use BSD dump-like or GNUTar-like backup programs. Of course, this includes the majority of systems that an administrator might want to backup, including Windows clients and various UNIX clients. There does not seem to be direct support for AFS backups. We believe that it would be possible to add such support to the system, although it would require editing Amanda's C source code and recompiling.

Burt also uses standard backup programs for each type of data source, but it is not limited to BSD dump-like and GNUTar-like programs. Burt allows the use of any program that writes to standard output as a backup program. Consequently, Burt can backup not only Windows clients and various UNIX clients, but also AFS, and even more exotic types of data sources, such as World Wide Web sites via HTTP sockets. Part of this flexibility comes from the manner in which the backup type layer is implemented. It provides a sufficiently generic interface to the engine that a very wide range of data sources can be used. The other part of the flexibility comes from the use of Tcl as the implementation language for the backup types.

By using standard backup programs, both Amanda and Burt limit themselves, in a sense, to the capabilities of those programs. For example, Networker's proprietary backup programs allow the creation of backup indices that enumerate every file backed up, rather than just listing the paritions or hosts that were backed up. In the case of Amanda and Burt, the creation of such indices is possible only if the underlying backup programs used in the backup system support that sort of index. BSD dump and derivative programs, for example, do not, but GNUTar and others do.

5.1.2 Parallel backups

The systems also vary in their implementation of parallel backups. Each supports the backup of several systems in parallel, in order to increase backup throughput. Legato Networker and Burt both multiplex backup data directly to the tape drive, as described above. This allows very high backup speeds to be achieved — often at or near the maximum speed of the tape drive — providing that a large enough number of systems are backed up in parallel. However, the tapes created by such a system may be difficult to read without using the backup software to do so.

Amanda, by contrast, uses a "holding disk" to which backups are written in parallel; from that holding disk, the backup data is written serially to tape. This implementation has some advantages. First, it is faster than serially backing up systems directly to tape, as demonstrated in [2]. And it creates a tape that can be read easily with standard UNIX programs like `dd(1)`. However, this implementation can be significantly slower than multiplexing directly to tape. This is primarily due to the addition of a "middleman" to the data path. Each block of backup data received by Amanda must be written to disk, then later read from disk and written to tape. Burt and Networker have only one write, directly to tape, for each block of data received.

5.2 Unique features

Besides their common features, each system has some unique features as well. Amanda's most notable feature is the high degree of automation that it provides to the administrator. It will automatically schedule backups and perform load balancing on backup tapes. It can also protect against accidental tape overwrites, and as noted above, produces tapes that can be read easily using standard UNIX utilities.

Burt's most notable feature is its high degree of flexibility in terms of supported types of data sources and user interface. We have not created a general purpose, graphical user interface for Burt; by design, such an interface is not a part of Burt. We have created a fairly sophisticated administrative interface, but it is highly site-specific, and probably of little use to other sites, except as a demonstrative example. In any case, we feel that the creation of the user interface is best left to the individual administrator, who is best qualified to know what features are needed at their site. Tcl certainly provides the tools to make an impressive user interface. Similarly, we have not created a general purpose backup scheduler, though we have created a scheduler for our AFS backups, and are in the process of developing a scheduler for our workstation backups. In addition to this flexibility, Burt has the ability to distribute backup data over multiple tapes, and it uses checksums to help ensure data integrity.

Networker's most notable feature is its sophisticated user interface. In addition to providing access to the administrative functions of the system, Networker's user interface provides users the ability to request restores from the system. If the backup media is stored in an automatic loader, such as a tape changer or tape library, Networker can perform user requested recoveries without administrator intervention. Networker also has the ability to distribute backup data over multiple tapes.

6 Experience and Performance

We began using Burt as our primary backup system in August 1997. After a few initial problems due to bugs in the implementations of our backup types, everything has gone smoothly.

6.1 Overview

We backup a total of around 900 gigabytes every two weeks, counting both full backups and incremental backups. We use a collection of 30 4mm DDS-2 tape drives for backups, with 90m DDS tapes, each with an uncompressed capacity of 2 gigabytes. We have enabled hardware compression on the tape drives to increase the capacity of our tapes. We choose to backup to a large array of relatively low capacity tape drives for three primary reasons. First, it gives us a second layer of parallelism, allowing us to increase our overall throughput. Second, it limits our loss if a single backup tape is damaged – instead of losing 35 gigabytes or more, as we might if we used DLT drives, we can only lose 4 gigabytes or so. Third, it is inexpensive to replace the drives if they fail.

Obviously we cannot fully backup everything every night. Instead, we use a two week "epoch cycle". Each night, we peform a full (BSD dump level 0) backup of a portion of the data, and incrementally backup the remainder. This totals roughly 60 gigabytes of data each night. This kind of backup policy is sometimes called *compositional* backups, and was easy to implement with Burt and Tcl.

6.2 Implementation

Each night a script is run at 1:00am (via `cron(1)`) to initiate our incremental backups. These occupy about 15 of our 30 tape drives for about 90 minutes. At 3:00am, the batch processor initiates the epoch backups that have been selected to run that day, on the other 15 tape drives. At 9:00am, an operator verifies that the nightly backups have completed successfully. If any require an additional tape, the operator loads a new tape in the appropriate tape drive. If any of the nightly backups have failed, which happens occasionally due to media errors, the operator restarts the failed backup. In addition, if any individual data sources have failed to be backed up, due to network errors or problems with the data source, the operator corrects the problem and runs a special "redo" backup for those data sources. Once all of the backups have finished, the operator loads tapes for the next nightly backup run.

6.3 Growth

Since Burt's introduction, we have significantly increased the size of our data set, in both total data size and total number of data sources. Originally, we backed up roughly 500 gigabytes every two weeks, from about 9,000 individual data sources, including AFS volumes and workstation disk partitions. Now we backup 900 gigabytes every two weeks, from about 13,000 data sources. We have had no problems scaling the system to accommodate this growth, although we did have to add additional tape drives to handle the increased data size.

6.4 Performance

We have had no trouble finishing backups in the six-hour window allotted to us each night. In fact, most of our backups finish well within that window, with the exception of the odd backup that runs over one tape. Backups that run over one tape do not finish until an operator comes in to load a second tape, and lie idle until that time. In general, backup performance with Burt has been quite good, often at or near the maximum possible rate for the drives and media we use. Network bandwidth and the I/O capabilities of the data sources being backed up seem to be the major limiting factors on our backup speed; Burt itself is not the bottleneck.

We do have some performance problems with our AFS backups, but those are due to the manner in which we schedule AFS backups. Our choices were influenced by the desire to be able to provide fast recovery from catastrophic AFS disk failures. One way to reduce the time needed to perform such recoveries is to minimize the number of tapes from which we must recover data. Accordingly, we schedule all the AFS volumes from a single partition on an AFS server to be backed up to a single tape. This minimizes the number of tapes we have to retrieve data from if that partition crashes, but it incurs a penalty on the backup speed. Typically, our AFS backups run at about half the speed of our workstation backups.

Of course, the purpose of doing backups is to be able to recover data. We have no significant difficulties recovering data, and have performed an average of one recovery a day over the past several months. Note that this number includes recoveries performed due to system crashes, system upgrades, user error, and tests to spot check our backups. One incident in particular stands out: in the summer of 1998, we suffered a complete disk crash on one of our AFS servers. We lost an entire disk containing over 300 user home volumes and several gigabytes of data. Thanks to Burt — and the administrator working on the problem — we had the server back up and running with all data restored within 24 hours. The majority of the time was spent waiting for the AFS servers to process the recovered data; reading the data off the tapes only required a couple of hours, and was performed in a single pass.

Overall, Burt has been a great success at our site.

7 Future work and considerations

Even though Burt in its present form has worked well for us, it is clear that there is still room for improvement. In particular, there are a few features that we would like to add to the engine.

First, we want to add broad support for tape changers. Many sites use tape changers to further automate the backup process. In some cases, Burt can use tape changers, but the support is not explicit, nor does it extend to the wide variety of tape changers available. This task is complicated by the fact that tape changers do not all use the same control interface, and we do not have any tape changers of our own with which to experiment. We recently received a source code contribution to provide tape-changer support for some kinds of tape changers, and will be working to integrate that code.

Next, we want to add support for the use of multiple tape drives from a single Burt process. Rather than writing all output to a single tape drive, the engine could write the output to multiple tape drives. Various writing patterns are possible: *striping*, in which writes alternate between tape drives (Figure 4); *mirroring*, in which all data is written to all tape drives to produce multiple copies of the backup tape (Figure 5); and *rollover*, in which the tape drives are used as sort of a "poor man's" tape changer (Figure 6). Each of these has its own benefits and penalties. Striping provides increased throughput, especially for slow tape drives, and increased capacity, but at the cost of an odd tape format. Mirroring provides additional backup reliability, because having two copies of a backup tape guards against media failure, at the cost of some backup throughput. Rollover provides increased capacity, with no significant penalty. Such functionality would be useful in some situations, and would further distinguish Burt from other backup systems.

We would also like to add more explicit support for non tape-like media. In particular, we should better support disk-like media, such as ZIP disks, and various magneto-optical disks. Presently, it is possible to direct Burt's output data stream to a file on disk, but that file is treated like a sequential access construct. It would be best to exploit the features of the disk that make it different from a tape, namely its random-access nature. One simple way to exploit that feature is to replace the use of filemarks to mark the beginning of data from a particular data source with byte offsets to mark those beginnings. We have a number of ideas about how best to implement this and other uses of disk-like media, but have not yet settled on one.

Finally we would like to create an online repository of backup types into which administrators could submit backup types they have created, and download

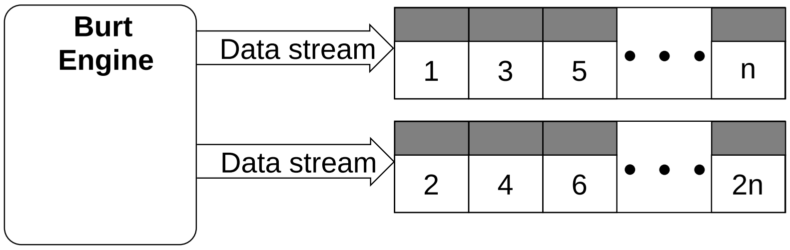

Figure 4: Sequence of writes in striping mode

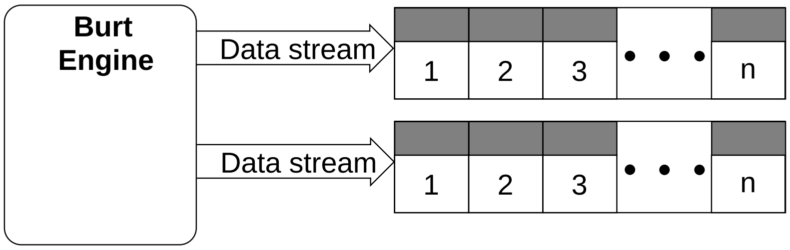

Figure 5: Sequence of writes in mirroring mode

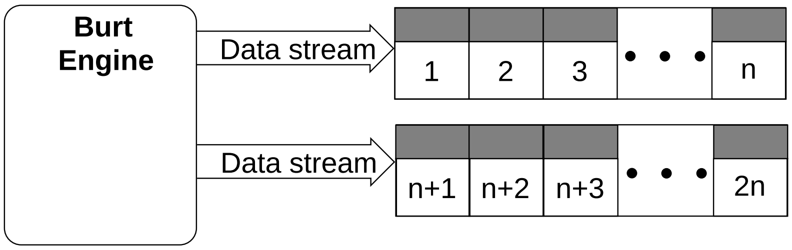

Figure 6: Sequence of writes in rollover mode

backup types that they may need. We have begun this repository with the backup types that we have created for our site, and look forward to receiving submissions from other administrators.

There is one important consideration that should be mentioned: Windows NT. Many people ask if Burt supports backups on Windows NT. The answer to that question is a qualified yes. Currently, the Burt engine cannot be run on Windows NT. We do not know what is involved with porting the engine to NT, so that capability may be long in coming. However, it is possible to create a backup type to backup data from Windows NT workstations. We know of two methods for doing so. First, the backup type could use Samba [14] to mount the NT volumes on a UNIX system, and then use SMBTAR or GNUTar to backup the data. Second, it could use any of the several Windows NT rsh services available to connect to a remote NT machine, and use GNUTar or *ntdump* (a port of BSD dump to NT that we are developing) to backup the data. We do not use either of these methods presently because neither conforms to our security policies. As soon as we find a way to reconcile that issue, we intend to begin backing up our NT workstations.

8 Conclusions and Availability

With regard to the goals previously outlined, we feel that we have been quite successful in meeting them. The principal contribution of Burt is that it provides a powerful I/O engine within the context of a flexible scripting language. In particular, Burt provides support for a wide variety of data sources; fast backups; support for data sources larger than the capacity of a single tape; support for large aggregate amounts of data; and extensible interfaces and functionality. Our own experience has shown that Burt provides all of these. We can backup all of the data in our department, including data sources larger than the capacity of a single tape, and we can do so quickly and reliably. We have been able to easily add support for new types of data sources. We have been able to scale our backup system to accommodate our significant growth in the past two years. We have been able to extend and customize our user interface as need and desired. These are features that all system administrators need from a backup system, and Burt provides them.

In addition, we feel that the model of creating software in two layers, one compiled, for performance reasons, and one scripted, for flexibility and customization purposes, is one that can be applied to a wide variety of problems. As noted, it is a model that has been used in many modern applications, of which office software is a particularly well known example. However, that use has typically been limited to minor customization and automation tasks. We found that a significantly larger portion of the application could be scripted than has been traditionally, and that doing so afforded us an extremely high degree of flexibility. We believe that hybrid applications that feature a substantial scripted component represent the future of software development, and look forward to seeing more software that is made more powerful and customizable by this approach.

Since October 1998, Burt has been available for download from the Burt homepage, `http://www.cs.wisc.edu/~jmelski/burt`. As of mid-April 1999, there have been about 6,000 visitors to the Burt homepage, and about 2,000 downloads of the Burt engine. Currently, we see about 50 downloads of the engine each week. We see about 30 downloads of the documentation each week, which is probably a better indication of the number of people actually interested in using it.

We have begun to receive source code contributions for the Burt engine source code from other Burt users. We believe that this bodes well for the future development of Burt, and look forward to receiving more such contributions in the future.

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 USENIX Technical Conference*, pages 217–225. The Usenix Association, June 1992.

[3] John Fletcher. An arithmetic checksum for serial transmissions. *IEEE Transactions on Communications*, 30-1:247–252, January 1982.

[4] John H. Howard. An Overview of the Andrews File System. In *Proceedings of the Winter 1988 USENIX Technical Conference*, pages 23–26. The Usenix Association, Winter 1988.

[5] Legato, Inc. *Legato NetWorker Administrator's Guide, UNIX Version (NetWorker for UNIX 5.5)*, December 1998.

[6] Donald Lewine. *POSIX Programmer's Guide: Writing Portable UNIX Programs*. O'Reilly & Associates, Inc., March 1994.

[7] Eric Melski. *http://www.cs.wisc.edu/~jmelski/burt/lisa99/backup.html*.

[8] Eric Melski. *http://www.cs.wisc.edu/~jmelski/burt/lisa99/recover.html*.

[9] Eric Melski. *http://www.cs.wisc.edu/~jmelski/burt/lisa99/search.html*.

[10] John Ousterhout. Tcl: An embeddable command language. In *Proceedings of the Winter 1990 Usenix Technical Conference*. The Usenix Association, Winter 1990.

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

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

[13] J. G. Steiner, B. Clifford Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the Winter 1988 Usenix Conference*. The Usenix Association, February 1988.

[14] The Samba Team. *http://samba.org*.

A The Solaris Backup Type

Following is the University of Wisconsin Department of Computer Sciences (UWCS) Solaris backup type definition as used with Burt.

```
################################################
# Function definitions for solaris dump type
################################################
```

```
################################################
# solaris_dump
#    The solaris backup type backup proc;
#    initiates a backup of the given atom on the
#    given host at the given level
################################################
proc solaris_dump {host atom level} {
    global sessionID tmpdir
    set hostlogfile $tmpdir/BURTlog.${sessionID}.$host
    if { [string compare $atom "/"] == 0 } {
        append hostlogfile ".root"
    } else {
        regsub -all {/} $atom "." newatom
        append hostlogfile "$newatom"
    }
    append hostlogfile ".$level.ufs"

    set dumpfd [open "|/s/std/bin/rsh $host -n \
        \"/usr/sbin/ufsdump ${level}uf - $atom\" \
        2> $hostlogfile" {RDONLY NONBLOCK}]

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

################################################
# solaris_cleanup
#    The solaris backup type cleanup proc; cleans
#    up temporary files created during the backup
#    of the specified host, atom and level
################################################
proc solaris_cleanup {host atom level} {
    global sessionID tmpdir
    set hostlogfile $tmpdir/BURTlog.${sessionID}.$host
    if { [string compare $atom "/"] == 0 } {
        append hostlogfile ".root"
    } else {
        regsub -all {/} $atom "." newatom
        append hostlogfile "$newatom"
    }
    append hostlogfile ".$level.ufs"

    catch {file delete $hostlogfile}
}

################################################
```

```
# solaris_monitor
#    The solaris backup type monitor proc; checks
#    the given line of dialog for keywords that
#    indicate an error
#################################################
proc solaris_monitor {host atom level line} {
    return [regexp {error|Unknown|EXITED|ATTENTION|abort|Bad} $line]
}

#################################################
# solaris_recover
#    The solaris backup type recover proc; performs
#    recoveries of the given host, atom and level
#################################################
proc solaris_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]
}
```

This backup type is used to backup disk partitions on Solaris workstations. It includes a backup procedure, a recovery procedure, and monitoring procedure, and a cleanup procedure. Each of these procedures is called by the engine as needed.

The backup procedure is responsible for connecting to the given host and initiating a backup of the given partition at the given level. It uses Kerberos V rsh [13] to make the connection, and uses the Solaris program `ufsdump(1)` to initiate the backup. The standard error output from `ufsdump` is redirected to a file, from which is it read and passed to the engine. File handles for the backup data and for the standard error data are passed back to the engine.

The cleanup procedure is an optional procedure that is used to perform any cleanup that may be required following the completion of the backup of a particular host and atom. When the engine finds that the backup of something of type *solaris* has finished, it will call the `solaris_cleanup` procedure with the particular host and atom that has finished. In this case, the cleanup procedure simply removes the temporary file that was used to store the standard error output from the backup of a particular item.

The monitor procedure is responsible for parsing a line of dialog from the standard error output of the backup of a particular item and determining whether or not the dialog indicates that an error has occurred. If the procedure determines that an error has occurred, the engine will abort the backup of the item. In this case, the monitor procedure checks the dialog for certain keywords that indicate an error has occurred.

The recover procedure is responsible for initiating a recovery of a particular item. When the engine finds that it is supposed to recover data from an item of type *solaris*, it will call the `solaris_recover` procedure with the particular host, atom, and level to be recovered. The procedure returns a writable file handle to the engine, and as the engine reads data from the backup media, if it finds a packet that is from the particular item, it will write the data to the file handle given by the recover procedure. In this case, the recover procedure just writes the data out to a file; our operators must then use the Solaris program `ufsrecover` to extract the required data from the file.

B Biography

Eric Melski graduated from the University of Wisconsin, Madison in 1999 with a BS in Computer Sciences. While at the university, he worked as a system administrator for 4 years. Following graduation, he joined Scriptics Corporation in Mountain View, California, where he is a software engineer. Reach him via U.S. Mail at Scriptics Corporation; 2593 Coast Avenue; Mountain View, CA 94043. Reach him electronically at jmelski@cs.wisc.edu.