Western Kentucky University
TopSCHOLAR®

Honors College Capstone Experience/Thesis Projects

Honors College at WKU

Fall 2007

Back-up Server for Computer Science Department

Victoria Gaylord
Western Kentucky University, victora.gaylord@wku.edu

Follow this and additional works at: http://digitalcommons.wku.edu/stu_hon_theses

Part of the Data Storage Systems Commons, Other Computer Engineering Commons, Other Computer Sciences Commons, and the Systems Architecture Commons

Recommended Citation
Gaylord, Victoria, "Back-up Server for Computer Science Department" (2007). *Honors College Capstone Experience/Thesis Projects.* Paper 116.
http://digitalcommons.wku.edu/stu_hon_theses/116

This Thesis is brought to you for free and open access by TopSCHOLAR®. It has been accepted for inclusion in Honors College Capstone Experience/Thesis Projects by an authorized administrator of TopSCHOLAR®. For more information, please contact connie.foster@wku.edu.

Introduction

Before this project was initiated, the responsibility of file preservation lay in each individual faculty member. Some opted to maintain a backup of important information, but others chanced that their data would remain safe. This latter method of data storage placed valuable research data, reports, papers, and other information at risk from hard drives failing due to viruses, hardware malfunctions, and other technological problems. Furthermore, a computer, functioning flawlessly, is susceptible to vandalism; a strong example given, on April 23, 2006, when a part of Cherry Hall was destroyed by fire. Professors lost years of work because of ruined hard drives and no secondary backup. The back-up server I helped to set up offers a solution to this critical problem by providing a means of redundantly storing information in a secure location.

The server runs a program called Bacula on a Unix operating system. Bacula makes nightly incremental backups, and it completes a full backup of all hard drives once a month – all of which are stored on tapes. Bacula makes no discrepancy between operating systems on the client machines. It was designed to support all major operating systems (Unix, Windows, Linux, Mac, etc), a major factor in the decision to use Bacula. Because of the value of the information kept on the server, a firewall script was written, using IPTables, to protect the server from network attacks. The server is tailored specifically for the Computer Science Department, but it can be accommodated for other networks.

There are three major sections to this project: hardware specifications, securing the server, and customizing Bacula to meet the department's needs.

Hardware

The server consists of the following specifications:

- Intel Pentium IV processor at 2.5 GHz
- 1 gigabyte RAM
- 200 gigabyte RAID 1 hard drive (The data is still salvageable in case of a hard drive failure.)

Securing the Server and Data Storage

In preserving information, it is vital to protect the server from security threats and technological failures. Though it is impossible to completely secure the server, fail-safes have been added that will help to guard it. There are three methods that have been used to protect it: the design of the firewall, the design of the backup schedule, and provisions made in Bacula itself.

Firewall (Full Document in Appendix A)

The firewall was written using IPTables script. This script provides mechanisms for controlling the flow and type of traffic to and from the server by specifying accepted/rejected ports, IP addresses, packets, etc.

Note: Because of security reasons, it is not possible to include all details in the IPTables script.

The IPTables script begins by refusing to allow any data to be accepted by, or forwarded through, the server. However, there are no restrictions on output because, in preventing access *to* the server, the server is then protected from being used to send rogue information out to other computers. (This was a design decision on my part. The most common IPTables script sets OUTPUT to DROP.) The following three rules implementing this are listed below:

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT
```

The rest of the commands in the script are the exceptions to the first two explicit rules. (In essence, the exceptions "poke holes" in the script that allow certain traffic into the server.)

The second most important part of the firewall is its recognition of IP addresses known to be spoofed and would most likely be used with malicious intent. (Spoofing an IP address fools a computer into believing that it is a legitimate connection and then tricks it into opening itself to a dangerous host.) If such an IP number is used, the firewall would automatically block it from the server and record its actions in a log for administrative review. Below is one such example (Bauer 84):

```
iptables -A INPUT -s 255.0.0.0/8 -j LOG --log-prefix "Spoofed source IP!"
iptables -A INPUT -s 255.0.0.0/8 -j DROP
```

Other IPs that the firewall will block and log are 10.0.0.0/8, 172.16.0.0/12, and 192.0.0.0/8. These IPs are not deliverable over the Internet, so if the server encounters them, it is safe to assume that these are spoofed (Bauer 84).

The server is also protected from someone or something exploiting its own IP address, which then could be used to create a feed-back loop and flood the server with its own packets, causing it to crash. (For security reasons, the server's IP address has been omitted from this report.)

```
iptables -A INPUT -s XXX.XXX.XXX.XXX -j LOG --log-prefix "Tried to use host IP!"
```

iptables -A INPUT -s XXX.XXX.XXX.XXX -j DROP

As with the spoofed IP addresses, this incident will be recorded into the logs for later review (Bauer 85).

There is another situation where the firewall must guard the server against attack: the TCP half-scan, more commonly known as a stealth scan. It takes advantage of the TCP three-way handshake by sending a rogue packet masquerading as a SYN-packet to a computer and then using the SYN-ACK reply to open a connection. This connection gives the attacker a back door into the system. This particular method is referred to as a stealth scan because these activities are not logged by the host system until the attacking machine sends a SYN-ACK reply, which is not done until it is finished exploiting the host (IBM Security Service Website). To protect the server from this, the following rule was implemented (Bauer 85):

IPTABLES –A INPUT –p tcp ! --syn –m state --state NEW –j LOG --log-prefix "Stealth scan attempt?"
IPTABLES –A INPUT –p tcp ! --syn –m state --state NEW –j DROP

This rule forces the firewall to ensure that packets coming in are actually SYN packets.

For general added security, a more specific forwarding rule was added to the firewall that will force the it to log and drop all incoming packets destined anywhere but to the server:

iptables –A FORWARD –j LOG --log-prefix "Attempted FORWARD?:"
iptables –A FORWARD –j DROP

The final important sections of the firewall are the exceptions that open incoming and outgoing ports. (A port is a number at the beginning of a data packet that directs the packet to a specific process running on the computer.) Recall that the default setting is to allow nothing into the server. Again, for security purposes, the ports have been omitted from the following example.

iptables -A INPUT -p tcp -j ACCEPT --dport XXXX -m state --state NEW
iptables -A OUTPUT -p udp --dport XXXX -m state --state NEW -j ACCEPT

Back-up Scheduling

There are three types of backups that Bacula can perform.

1. Full: all information on client machines is backed up
2. Differential: all files that have been altered since last full backup are backed up
3. Incremental: all files that have been altered since the last incremental back up are backed up

In order to have an efficient back-up system, it is important to have more than one copy of the information downloaded from client computers. The scheduling policy is left to the administrator's discretion. For the Computer Science Department, Bacula is scheduled to run incremental backups six days a week. Once a month, it will perform a full backup of the client machines. There is also a differential backup completed every week except for the time when it performs a full backup. Multiple tapes are then stored, and they are rotated out on a four-month (120 days) basis or when full. For further redundancy, Bacula's catalog, which maintains an index of the files stored, is copied to tape at the end of *each* back-up session. This method ensures that if any or all of a tape is destroyed, there is still a chance, depending on the backup date of the file, that data can be restored.

Bacula's Security Provisions

The backup process is initiated by the server. It contacts the client machines using the IP addresses listed in its director-configuration file and waits for the client to answer. Both Bacula and the client machines have a password that must match before backup can begin. (These passwords are stored within the daemons so that only the machines ever see them and the user is not responsible for knowing them, adding further security.) If passwords do not match, Bacula will refuse the connection. Also, if the client does not answer within a specified time (23-30 minutes), then Bacula will move on to the next client.

Customizing Bacula (*Configuration Files in Appendix B)*

Why Bacula?

Bacula is incredibly flexible. It supports a number of back-up devices, including tape drives, DVDs, CDs, and hard drive storage. It is also easy to add and remove clients, which is important because it is common for departmental personnel to change offices, transfer, and to hire new personnel. From a hardware perspective, new computers are also being cycled in as old ones are removed. Bacula allows an administrator to update the client's information (usually only the IP address) without the need to configure a new client in the daemon files. Bacula also works across all operating systems while occupying a very tiny portion of space in the client's computer. It also requires minimal user intervention, which helps reduce the possibility of a user-created error. (These two reasons in particular were the two most important reasons for selecting Bacula to handle the back-up jobs.) Finally, Bacula is freeware, making it easier to tailor the software specifically to the administrator's needs.

Listed in Bacula's documentation (36) are some other advantages over other systems. A sample few include:

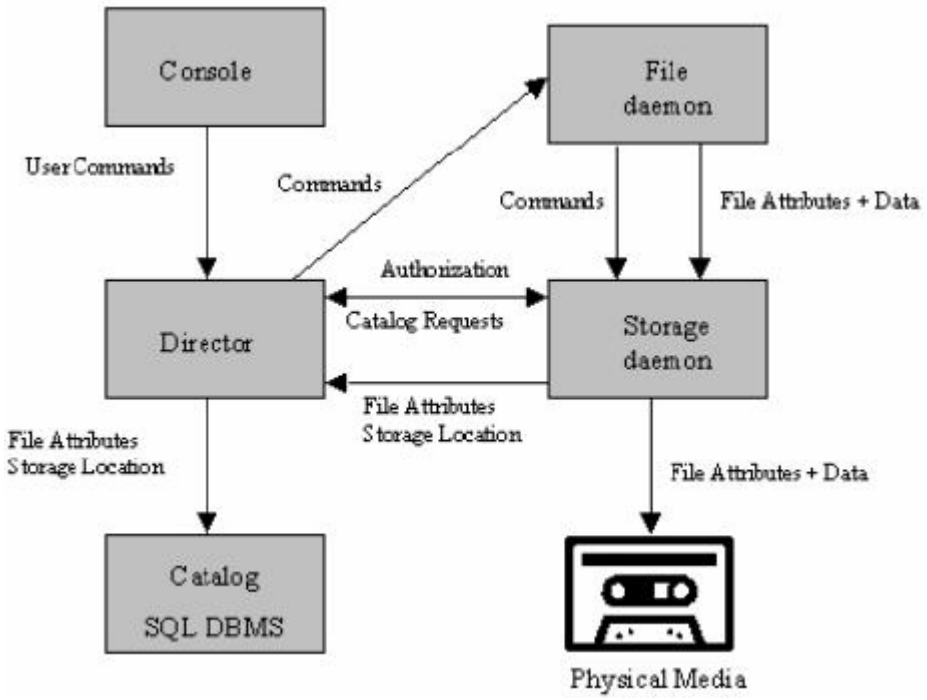
- Automatic pruning of the database thus simplifying database administration.
- Bacula has a built in job-scheduler.
- Bacula handles multi-volume backups.

An Overview of Bacula

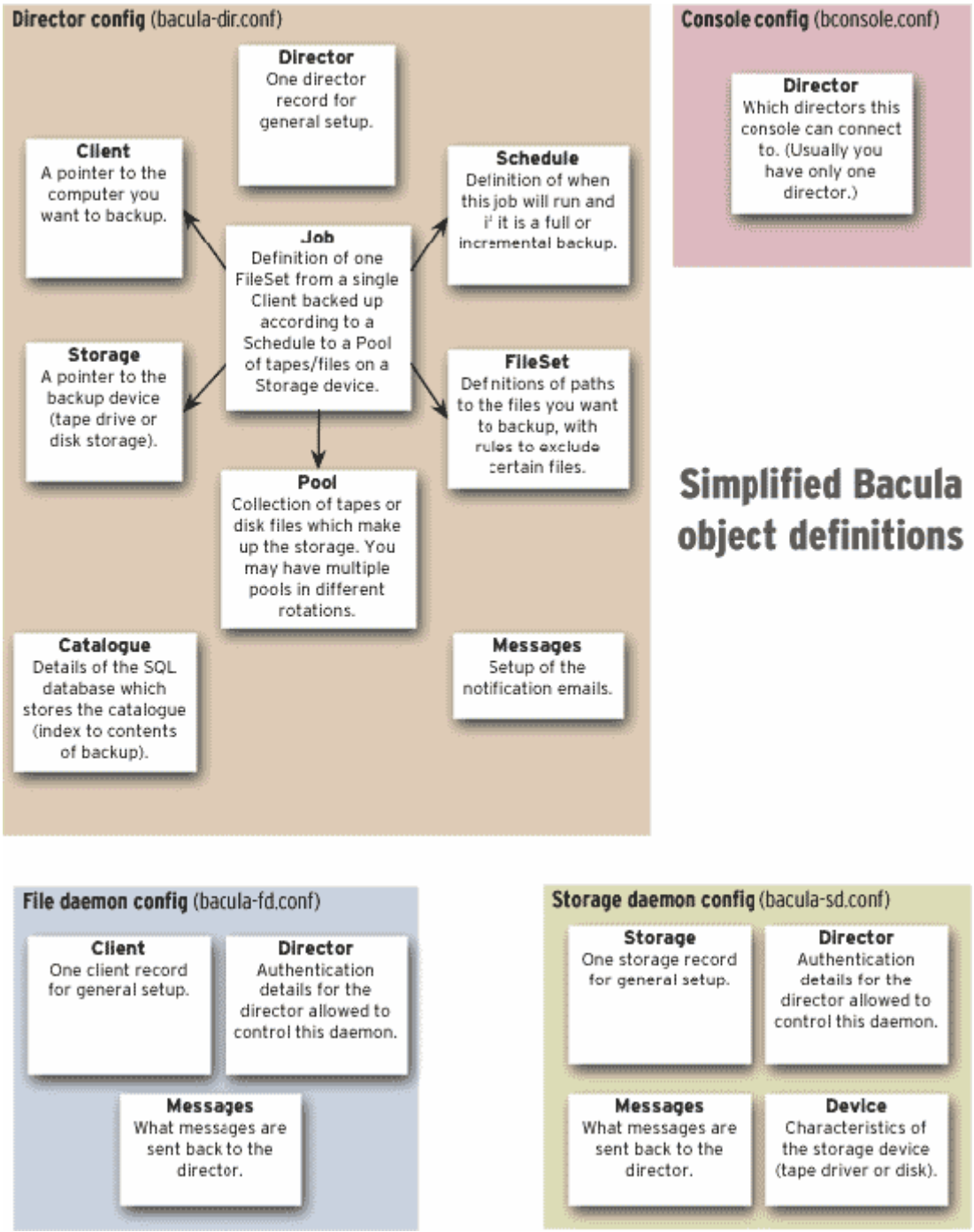
Bacula is divided into five main parts: the Director Daemon, the Console, the Catalog, the Storage Daemon, and the File Daemon. (A daemon is Unix terminology for a process running in the background.) Below is a brief list of the terminology used frequently in this document. (Bacula: The Network Back-up Solution 28-31)

> File Daemon (also known as the Client Service or File Service): runs on the client computer to be backed up. It streams data from the client machine to the server as the server requests it.
>
> Note: The server is also a client in this particular case because its files are also backed up. Therefore, the File Daemon also resides on the server.
>
> Storage Daemon: the code that writes the attributes and data to a storage Volume (usually a tape or disk).
>
> Catalog: used to store summary information about the Jobs, Clients, and Files that were backed up and on what Volume or Volumes. For this server, MySQL was used.
>
> Console: allows the user to interact with the Director Daemon.
>
> Director Daemon: the main Bacula server daemon that schedules and directs all Bacula operations.

The following is a basic diagram of the parts of Bacula and their relationships to each other (Bacula: The Network Back-up Solution 33).



Before continuing, it is important to define key terms used in Bacula. To do this, I have included a diagram that includes the necessary definitions for convenient reference (Bacula: The Network Back-up Solution 27).



The Console

The Console, accessed by the command *bconsole*, allows the user, or, in this particular case, the administrator, to interact with the Director. Though it does have a GUI interface, one was not installed for this case. Instead, *bconsole* is called through the Unix prompt. However, it should be noted that Bacula also provides a means for the client machines to use *bconsole,* but it was decided not to install it on the client machines. It has numerous commands, but some of the more common ones used include

- run – run a job immediately
- messages – display messages from the Director Daemon
- status – status of clients and daemons
- reload – reload configuration files
- restore – restore an archived file from the tape to the client machine (more about this later)

The Catalog

The Catalog is kept in a MySQL database and is updated every time Bacula runs a back-up job. By default, it is kept on the hard drive, but for an added redundancy, the catalog is also written to tape at the end of each run. The Catalog is vital to the restore process because Bacula uses it to find the most recent backup of a particular file, which could be at any place on a tape or on any tape. Bacula cannot function without it.

The File Daemon

As stated above, the File Daemon is installed on all client machines. The Director Daemon contacts the File Daemon and must have the same password as the File Daemon. (The password is different for each File Daemon.) The File Daemon is relatively simple in configuring as shown below. (Comments denoted by "#".) Again, for security reasons, actual names and passwords have been omitted.

```
Director
{
  #name of the director permitted to contact the client machine
  Name = NameoftheDirector

  #Director Daemon must match this password for a connection
  Password = ClientPassword
}

FileDaemon
{
  #name of the File Daemon on the client machine
  Name = FileDaemonName

  #Port number that Bacula uses to communicate with Client
  FDport = 1234
```

```
  #client listens for Director Daemon
  WorkingDirectory = /bacula/working
  Pid Directory = /bacula/working
}

Messages
{
  #All messages except those regarding skipped files will be
  #sent back to the director.
  Name = Standard
  director = NameoftheDirector = all, !skipped
}
```

An important note: when installing a File Daemon on a Windows machine, remember to open the Windows firewall to the port number listed in Bacula. Otherwise, the firewall will block the Director from contacting the Client machine.

The Storage Daemon

By default, the Storage Daemon is set to handle many types of storage devices. Because of the volume of data expected to be handled, it was determined that a tape drive would be most appropriate for the Computer Science Department. Though somewhat slow to read and write as compared to most types of storage media, tapes are cost-efficient, easy to store, and can hold large amounts of data. A sample configuration is included, but actual names and passwords have been omitted. (Comments denoted by a "#".) The code was modified from the Bacula Documentation (184-186).

```
Storage
{
  Name = NameofStorageDaemon

  #Port number Director uses to communicate with Storage Daemon
  SDPort = 1234


  #Storage Daemon listens for Director Daemon
  WorkingDirectory = "/root/bacula/bin/working"
  Pid Directory = "/root/bacula/bin/working"

  #Number of Jobs that can be written at once
  Maximum Concurrent Jobs = 20
}

Director
{
  Name = NameoftheDirector
  Password = PasswordtoContactStorageDaemon
}

# Restricted Director, used by tray-monitor to get the
# status of the storage daemon
Director
{
```

```
  Name = NameofRestrictedDirector
  Password = PasswordtoContactStorageDaemon

  #permitted to check status
  Monitor = yes
}

#Tape Drive
Device {
  Name = NameofTapeDrive
  Media Type = DLT
  Archive Device = /dev/nst0

  # when device opened, read it
  AutomaticMount = yes;
  AlwaysOpen = yes;

  #If yes, this supports removable media - CDs, tapes, etc
  RemovableMedia = yes;

  # If Yes, the archive device is assumed to be a random access
  # medium which supports the lseek facility (DVD, USB, and
  # fixed files)
  RandomAccess = no;

  #dummy directory - see below
  Spool Directory = "/tmp"
}
```

The Spool Directory is too important for a mere comment. This directory on the server's hard drive holds the data before streaming the files to the tape drive. This method keeps that tape moving smoothly rather than constantly stopping and starting (saving wear and tear on the tape), which would happen otherwise. Another function of the directory is to hold a record of the jobs yet to be completed in the event that the server is unable to follow its back-up schedule. This was tested successfully when a full tape was not replaced for nearly four days and the server hung up that entire time. Once the tape was replaced, Bacula then performed the four-days worth of jobs in succession and resumed its schedule that night.

The Director

The Director Daemon is the boss of the system. It communicates with all other daemons and coordinates the backup process. It is responsible for knowing its clients, the back-up schedule, what messages to send and to where, the storage device(s) used, etc. Because of this, it is also the most complex and took the longest to set up. It was also modified several times because of the number of design decisions required, and the most efficient use of Bacula could not be determined until it was put in action.

Note: By default, much of the Director's configuration file was already complete. Only that which needed to be modified or added is included in this report. As with the other daemons, any sensitive information has been omitted from this document.

Before any clients can be added, the Director itself must be named and defined.

```
Director {

  #name the director
  Name = NameoftheDirector

  #where director listens for connections
  DIRport = 1234

  QueryFile = "/root/bacula/bin/query.sql"

  #director listens for other daemons
  WorkingDirectory = "/root/bacula/bin/working"
  PidDirectory = "/root/bacula/bin/working"

  #number of jobs Director can run at once
  Maximum Concurrent Jobs = 1

  #Console password
  Password = "Console Password"

  Messages = Daemon
 }
```

The JobDefs resource is the default settings for the job. By defining this, only the exceptions to this default need to be defined later on. In other words, it saves typing.

```
JobDefs {

  #Name of JobDefs
  Name = "DefaultJob"

  #Type of Job
  Type = Backup

  #Type of Backup - Differential, Incremental, Full
  Level = Incremental

  #Director - remember that the server is also a client
  Client = NameoftheDirector

  #FileSet knows the files to be backed up
  FileSet = "Full Set"

  #Name of Schedule to be used
  Schedule = "WeeklyCycle"

  #by Default, it was set to write to file, but it was changed to
  #a removable device
  Storage = DLT

  #What messages to send and to where
  Messages = Standard

  #Name of pool to use - the default defined by Bacula sufficed
```

```
  Pool = Default

  Priority = 10
}

# List of files to be backed up
FileSet {
  Name = "Full Set"
  Include {
    Options {
      signature = MD5
 }
```

Defining an appropriate back-up schedule is essential. If back-ups are performed too far apart, then the most current file modifications will most likely not be up-to-date when a restore is needed. If backups are too close together, then storage space will be wasted with too many redundant files. Some factors to take into account when deciding an appropriate schedule include

- how often the client machines are used – The more often files are modified, the greater the likelihood that the files backed up will be out of date
- how many clients exist – If there are many clients, it may become necessary to space the back-up jobs out over multiple days
- amount of data to be stored – A tape has limited space, and if it runs out of space, someone must be there to change the tape. If a tape is full, Bacula will continue to spool data to the hard drive (until full) until the tape is changed.
- when back-ups can take place – Bacula cannot back up files actively being modified, so a design decision was made to run the back-up jobs at night when computers will be dormant.
- likelihood that a restore will have to be performed – Naturally, the most recent data will be the most desirable.

After taking all of these factors into consideration, the backup schedule looked like this:

```
#times are based on a 24-hour clock
Schedule {
  Name = "WeeklyCycle"
  Run = Full 1st sun at 23:05
  Run = Differential 2nd-5th sun at 23:05
  Run = Incremental mon-sat at 23:05
}

# How often catalog is backed up.
# It starts after the WeeklyCycle.
Schedule {
  Name = "WeeklyCycleAfterBackup"
  Run = Full sun-sat at 23:10
}
```

The Client has three parts: *Client, Job,* and *FileSet.* While *Client* and *Job* must be defined for each client, there is a default *FileSet* that will back up the My Documents

folder. If the Client has other folders that need to be backed up, then a separate *FileSet* will have to be defined for that client.

```
Client {

  #Name of File Daemon to contact
  Name = FileDaemonName

  #IP address to contact File Daemon
  Address = XXX.X.X.XXX

  #Port to connect
  FDPort = 1234

  #Catalog to store information about files stored
  Catalog = MyCatalog

  #Password on Director Daemon must match File Daemon or
  #connection is refused
  Password = "Client Password"

  #How long file records stay in catalog
  File Retention = 30d

  #How long job records stay in catalog
  Job Retention = 180d

  #Catalog will remove old file and job records if past retention
  #period
  AutoPrune = yes
}

Job {

  #Name of the Job
  Name = "NameofJob"

  #Type of job
  Type = Backup

  #which client job is for
  Client = FileDaemonName

  #what files to back up
  FileSet = "This Set"

  #When job will be performed
  Schedule = "WeeklyCycle"

  #Storage Device to copy files
  Storage = DLT

  #Default is Standard; for convenience this was changed - more
  #about this later
  Messages = TheseMessages
```

```
  #Type of pool
  Pool = Default

  #done so restore will know which tape holds file
  Write Bootstrap = "/root/bacula/bin/working/Client2.bsr"

  #write to disk and then to tape - saves wear and tear on tape
  SpoolData = yes
}
```

For most of the clients, the default FileSet was not used. Rather, each client had their own unique FileSet. However, rather than list the files to be included and excluded within the FileSet, it was easier to create a separate file with the list of files. In this way, there would be no need to search the Director configuration files to make any later changes, but instead one would only have to go to a separate file and make the changes there.

```
FileSet {

  #Name of FileSet
  Name = "This Set"

  #Files to back up
  Include {
    Options {
      signature = MD5
    }
    #directory to find list of files to be backed up
    File = "<filesets/incclient1"
  }
  #Files to be ignored
  Exclude {
    #director to find list of files to be ignored
    File = "<filesets/execclient1"
  }
}
```

In the two files, *incclient1 and excclient2,* the files to be included and excluded (respectfully) are listed.

```
incclient1.txt

C:/Documents and Settings

execclient1.txt

C:/Documents and Settings/wkuuser/My Documents/thisfile
C:/Documents and Settings/wkuuser/My Documents/thatfile
```

An excellent example of Bacula's flexibility is in the *Messages* resource. Each client can have its own variation of *Messages* without having to use the default. This way, the administrator can determine who should be e-mailed about warnings, errors, etc. Also, the administrator can determine which messages from the director should be included in

the e-mail. On one hand, an e-mail may be generated for everything that goes right and wrong with the back-up job, or very little messages at all. In a design decision, it was determined that, as well as the administrator, the client user should be aware of any problems occurring during the job. That way, the user will be aware that the most recent modification of files is not on tape and any restore would retrieve an earlier version of their file. However, if the job terminated successfully and there were no errors or warnings, then there will be no e-mail generated. For our implementation, the e-mails were most often generated from errors resulting from faculty shutting down their computers at night. If the computers were not on, then the server would be unable to reach the client. An e-mail would be sent to the faculty member notifying them that their files were not backed up during the last session.

```
Messages {
 #Message Name
 Name = TheseMessages

 #mailcommand - used to specify exactly HOW to send the mail
 #uses bsmtp Unix commands
 #client
 mailcommand = "/root/bacula/bin/bsmtp -h localhost -f
 \"\(Bacula\) john.smith@wku.edu\" -s \"Bacula: %t %e of %c %l\"
 %r"

 #administrator
 operatorcommand = "/root/bacula/bin/bsmtp -h localhost -f
 \"\(Bacula\) bacula.administrator@wku.edu\" -s \"Bacula:
Intervention
 needed for %j\" %r"

 #send e-mail in case of error, but not files skipped,
 #saved normally, general information, or that terminated
 #normally
 #send the message to the email addresses that are given as a
 #comma separated list in the address field.
  mail = john.smith@wku.edu, bacula.administrator@wku.edu = all,
!skipped, !saved, !info, !terminate

 #Volume mount or intervention requests from the Storage daemon
  operator = root@localhost = mount
  console = all, !skipped, !saved
  append = "/root/bacula/bin/working/log" = all, !skipped
}
```

Finally, the Director Daemon must also know about the storage device(s). This is relatively simple as shown in the code below.

```
Storage {

 #Name of Storage Device
 Name = DLT

 #Address to contact storage device - done through director
  Address = XXX.XX.XX.XXX
```

```
  #Port to listen
  SDPort = 4567

  #password for Storage daemon
  Password = "StoragePassword"

  #must be same as MediaType in Storage daemon
  Device = DLT

  Media Type = DLT

  #Autochanger device? Yes/no
  Autochanger = no
}
```

Tape Rotation

Because the tapes can hold a large amount of data (80 GB uncompressed) and the number of client machines is still minimal, tapes are rotated out only when they become full. However, on a larger system, or when the number of clients using the server or backup jobs increase, the rotation scheme may have to be changed. It could be possible to use a separate tape for incremental, differential, and full, or put separate clients on separate tapes. In a case where many tapes are required, Bacula also supports an auto-changer, but one was not needed for the Computer Science Department. A good indication of when to reconsider the tape rotation scheme is when the back-up schedule changes.

Tape Expiration

Tapes are set to expire within one year of their initialization. This time frame can be changed in the Director Configuration. However, this does not mean that a tape cannot be reused. When using an expired tape, it must be reinitialized, and then new data can be written.

Restore

There is very little point in having a back-up system if the files cannot be restored. It is for this that the Catalog is so vital because the *restore* uses the Catalog to find the location of the files. If anything happened to the Catalog, *restore* could not function. (Fortunately, the Catalog is also backed up at the end of every nightly session.)

In order for *restore* to work, there must be a Job defined in the Director configuration files that defines the *restore* command. Only one Restore need be defined regardless of the number of clients. A default version is given, so only one change needed to be made.

```
Job {
  Name = "RestoreFiles"
  Type = Restore
  Client=csbackup-fd
  FileSet="Full Set"
```

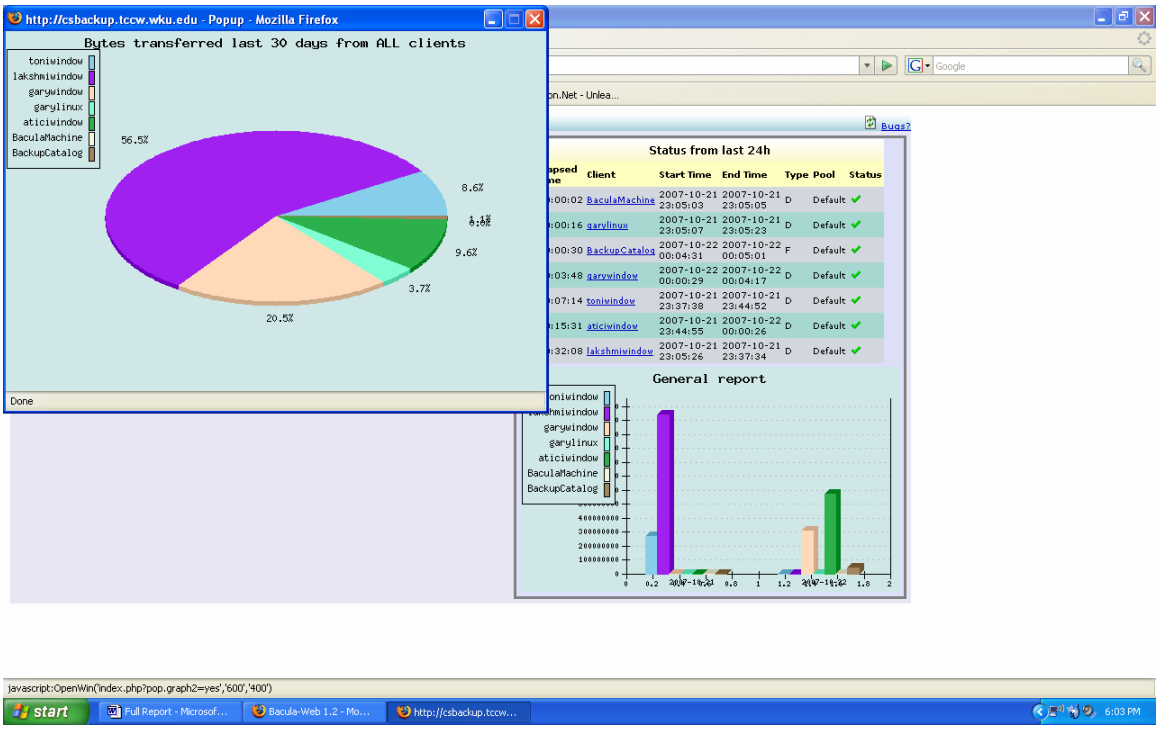
```
 #Changed Storage from "File" to "DLT" - name of tape drive
  Storage = DLT
  Pool = Default
  Messages = Standard
  Where = /tmp/bacula-restores
}
```

To run a restore, the *restore* command is entered under the bconsole command prompt. There are twelve different ways to find the file, through date written, clients, filename, the catalog, etc. Once the correct file is found, Bacula will ask for the tape holding that file to be mounted, and then it will proceed to restore the file to a temporary directory on the client machine. It WILL NOT overwrite the file of that same name on the client machine. Also Bacula WILL NOT discriminate between the client asking for the restore and the client from which the file was originally copied. This means that Client1's files can be sent to Client2. However, because the client machines were not given the power to do their own restores, only the administrator can perform a restore from the server. This adds security to the saved data by preventing one client's files from being sent to another client's machine

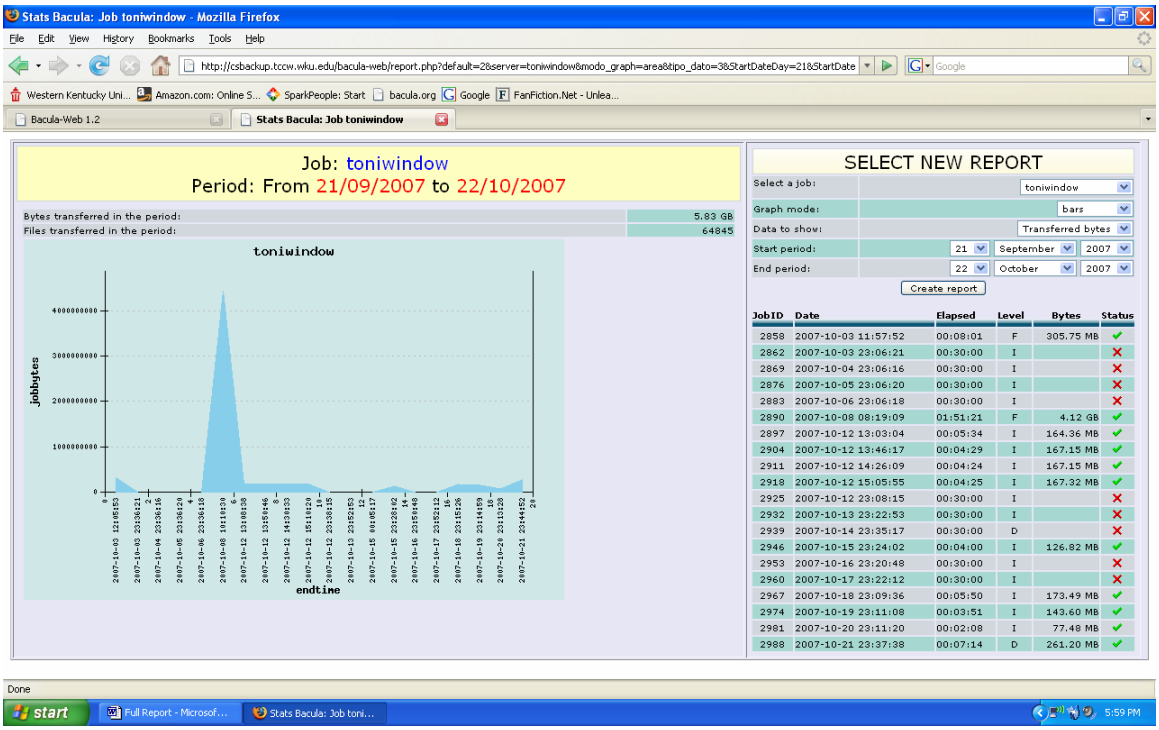
Note: Fortunately, there has not been a need to use the *restore* command, but before faculty machines were added to the back-up system, restores were completed on test machines to make sure that the information was restored correctly. There were no problems and every restore was successful.

Website to Check Server's Progress

Aside from the e-mails and server messages, there is a third way to not only check Bacula's progress but to also obtain a graphical representation of data written. On the next page are some screen shots of the website.


http://csbackup.tccw.wku.edu - Popup - Mozilla Firefox
Bytes transferred last 30 days from ALL clients
toniwindow
lakshmiwindow
garywindow
garylinux
aticiwindow
BaculaMachine
BackupCatalog
56.5%
8.6%
9.6%
3.7%
20.5%
Done
Status from last 24h
Client
Start Time
End Time
Type
Pool
Status
BaculaMachine
garylinux
BackupCatalog
garywindow
toniwindow
aticiwindow
lakshmiwindow
Default
General report
Bugs?

javascript:OpenWin('index.php?pop.graph2=yes','600','400')
start
Full Report - Microsof...
Bacula-Web 1.2 - Mo...
http://csbackup.tccw...
6:03 PM


Stats Bacula: Job toniwindow - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://csbackup.tccw.wku.edu/bacula-web/report.php?default=2&server=toniwindow&modo_graph=area&tipo_dato=3&StartDateDay=21&StartDate
Bacula-Web 1.2
Stats Bacula: Job toniwindow
Job: toniwindow
Period: From 21/09/2007 to 22/10/2007
Bytes transferred in the period: 5.83 GB
Files transferred in the period: 64845
toniwindow
jobbytes
endtime
SELECT NEW REPORT
Select a job: toniwindow
Graph mode: bars
Data to show: Transferred bytes
Start period: 21 September 2007
End period: 22 October 2007
Create report
JobID
Date
Elapsed
Level
Bytes
Status
Done
start
Full Report - Microsof...
Stats Bacula: Job toni...
5:59 PM

Recent Upgrade / Future Improvements

Between the time when Bacula had been installed on the server and the present, a major upgrade was released. Though there was serious concern about how an upgrade would affect the configuration files and the backup schedule, I am pleased to report that upgrading was relatively easy. Once the new version was downloaded and installed the catalog and the File Daemon on the client machines also had to be upgraded, but the configuration files were untouched and the next backup went smoothly.

Problems

There was one problem that could not be resolved. On the rare occasion when there was a glitch in the Internet connection, Bacula would stall on a client rather than moving on to the next client when it could not establish a connection. Waiting time should have been between 23 and 30 minutes, but if something happened to the connection, it would become hung up for nearly ten hours before finally moving on, pushing the other jobs into the morning when other client machines were in use. Upon checking into various blogs, it is not an uncommon problem but has yet to be fixed by the Bacula designers.

Conclusion

The server has now been running successfully for over a year, and Bacula has proven to be an efficient means of backing-up data. Though it has required several "tweaks" after initially establishing it on the server, the adjustments were usually minor and easy to implement. The learning curve is probably the most frustrating to overcome, but understanding the vocabulary helps to reach it faster. Bacula is also a work in progress, and while minor updates are a frequent occurrence, major updates are not, which is a great advantage for those who have other jobs other than acting as a system administrator. Though it has not functioned flawlessly, it has functioned efficiently, and the Computer Science Department now has a safeguard for its computers.