Local Disk Depot - Customizing the Software Environment

Walter C. Wong
Carnegie Mellon University

Local Disk Depot – Customizing the Software Environment

Walter C. Wong – Carnegie Mellon University

ABSTRACT

The **depot** model, developed at Carnegie Mellon University, provides a method for managing third-party and locally developed software. **depot** uses an object-oriented approach to managing software; each software package is managed as one or more logical objects. Yet, from the perspective of a user, the multiple ''software objects'' appear as a single, integrated, software environment.

Local disk depot **(ldd)** is an extension to the **depot** framework. **ldd** facilitates the management of environments that ''inherit'' software from the ''master'' software environments. The inherited software environment is formed by taking software and configuration information from the master software environments, and integrating that with local software and configuration information. The most common use of **ldd** is to have an inherited environment on the local disk of a workstation. This allows the workstation administrator to locally cache software in order to improve performance and availability of critical software in the event of server or network failure.

The inherited environments, however, can be used for more than saving local copies of remote software. Workstation administrators can introduce customizations to the software environment, as well as add additional software, even from other software environments. Developers can easily test new or updated applications on their own machines in an environment that is otherwise identical to the released environment.

Introduction

The software release management model currently in use by the Andrew system at Carnegie Mellon University is designed around a tool called **depot**[Coly92B]. In the **depot** model, each component of a software environment is separated into logically managed software objects called *collections*. An example of a software environment would be `/usr/local`, and a sample collection would be all the files and directories that make up the application **gnu-emacs**.

There is a separate directory for each collection. Each of these directories contain the files and directories that make up the collection. The organization of these files and directories reflect how the application should look when exported to the environment. For example, the `info` files for **gnu-emacs** would be stored in the `info` directory of the **gnu-emacs** collection. Similarly, the binary for **gnu-emacs** would be in the `bin` directory, the library files in `lib`, etc.

The separation of software objects is primarily for administrative purposes. The users do not see separate pieces of software, rather they see the union of all the individual software objects, the software environment. Thus, even though the binary for **gnu-emacs** is located in a separate directory from the **X11** collection, the users would still access the **gnu-emacs** binary in the environment's `bin` directory, such as, `/usr/local/bin`, along with the **X11** executables, and all the other executables in the environment.

Up until very recently, the software environments supported by the Andrew system were exported to client workstations via AFS, a distributed file system [Saty85]. The `/usr/local` path was just a symbolic link to the AFS path where the software was stored and managed. The most obvious benefit of this scheme was central maintenance of the software environments. The administrative burden of software installation, customization, and maintenance was removed from the local workstation owner. Local hard drive space was expensive and it was more cost effective to buy larger hard drives and place them on the servers. Finally, there wasn't that much software available for UNIX systems.

Today, the administrative issue is still a strong argument for central maintenance of software. However, disk space is cheap, and there are also many compelling reasons to move software to a local workstation's disk:

- *Local is faster than remote*. Even with AFS, a caching distributed file system, file access is much faster locally than from the distributed file system, even if the application is already cached [Stol92]. Additionally, the fileserver and network are shared resources so your workstation is competing with all the other workstations using those resources.

- *Local has greater availability*. In the case where the applications resides on the local workstation, the only fault that can prevent software from being accessible is a failure on the local machine (assuming there are no license server or other similar dependencies). However, when using a distributed file system, not only will a local problem stop you from getting to the software, you are now also susceptible to network problems as well as any problems that may occur on the servers you are using.
- *Local allows customization*. Software that is centrally maintained limits the options the local workstation owner has in customizing and configuring it to suit his needs. This does not have to be the case, but as the user to administrator ratio keeps increasing, there is even less time that an administrator can allocate to do something specifically for one user or one machine.

depot, by itself, has the ability to create a software environment on the local disk, based on a software environment on a distributed file system. What **depot** lacks the ability to merge local customizations with those from the central services. This is where local disk **depot (ldd)** comes in. **ldd** allows central maintenance of the software environment to co-exist with local configurations of the software environment. In this manner, new software can be introduced to the system, current software updated, and old software removed by the central services without direct intervention from the workstation administrator. Furthermore, any configuration options set by the local administrator will remain in effect throughout any of the central changes.

Implementation

The implementation of local disk **depot**, comprises of three parts. The first is a preprocessor. The second is the directory layout, or tree structure, of the software environment. The final component is a simple shell script wrapper that provides a simple interface to updating the environment.

Rather than making **depot** a ''kitchen sink'' type tool, we have another program to perform the preprocessing tasks. The preprocessor, **dpp**, takes the local environment configuration file, `custom.depot.proto`, runs it through a macro preprocessor, **mpp**, which expands all the variables and incorporates all of the specified library files. Then, **dpp** merges the configuration information from all the given environments produces and produces `custom.depot`, the configuration file used by **depot**. For example, a minimal `custom.depot.proto` for the `/usr/local` environment, would contain the following lines:

```
%include /afs/andrew.cmu.edu/wsadmin/depot/src/depot.include
*.searchpath: ${local}
```

The `%include` directive is an **mpp** command which incorporates the centrally maintained customization information, including the definitions of various variables. For example, **${local}** gets translated to the proper AFS path for each supported architecture. On a DECstation using the default released software, **${local}** expands to the path `/afs/andrew.cmu.edu/pmax_ul4/local/depot`. The `*.searchpath` line is normally used by **depot** to locate the software collections that will be integrated into the environment. **dpp** also uses this option to locate and include configuration information. In the event that multiple searchpath elements are given, the `custom.depot.proto` has precedence for configuration information, and then the precedence depends on the order in which the searchpaths are listed. The earlier on the `searchpath` list, the higher the precedence.

The second component is the tree structure. The collections that are released for general use are located under `${local}`, When a new software application is released, the old version is not removed from the environment. Rather the new version is placed in `${local}` with a higher **depot** version number.[1] Our choice for an ever-increasing version number is the UNIX time value, that is, seconds since 1970. For example, for in the case of the `splus` collection, under `${local}` we have:

```
splus.714684343  splus.745275952
splus.746318854  splus.747006152
```

Thus, `splus.747006152` contains the files that are actually available in the environment.

Every 30 days, the environment is examined to see what collections may be removed. The 30 days is an arbitrary limit within which we expect that a **ldd** update will occur on every workstation. At this 30 day interval, any version that is older than 30 days is removed, unless it is the only version out there.

This organization is necessary because files on the local disk of the client workstation may be linked to files that exist on AFS. In order to ensure that these symbolic links continue to point to valid files, one either has to update every client workstation whenever the software environment changes, or one needs to provide a way for the local environment to stay consistent until an update can occur. Updating all the workstations at once did not seem to be a scalable or reliable solution, so the tree organization described above was developed. This mechanism also provides the local workstation owner the ability to set the update time and

[1] When **depot** is run, if uses the collection with the highest version number.

frequency to best suit his needs, as long as it is within the 30 day interval.

The third component is a front end. It has two primary functions. The first is logging and notification. It centrally records the configuration file so that the central facilities have some idea of what machines are out there and what their configuration is. Additionally, this provides a backup of the configuration information. The front end also notifies both the local administrator and the central administrator in the event that an error occurred during an update. The second purpose of the front end is to simplify the interface to the **ldd** system. For example, the front end replaces the need for the workstation administrator to know all the command line arguments that have to be passed to both **dpp** and **depot**. With this interface, all the user needs to do is type in

```
/etc/dodepot  <environment>
```

where `<environment>` is the path to the environment that should be updated, for instance `/usr/local`.

Usage

Copying Files

The most common usage of **ldd** is to move applications that normally reside on the distributed filesystem to the local drive of workstations, and ensure that the applications do not get out of date. A good deal of flexibility is provided for the workstation administrator. He can specify that files, directories or entire collections are to be copied. Alternatively, entire directory hierarchies can be marked as 'link only' in order to conserve local disk space. For example, the workstation owner can specify that all the files (and directories) in `bin` and `lib` are to be copied while all the files in `include` and `man` are to be linked. Thus the workstation administrator can have locally exactly what he wants locally. When any of the software is updated or changed, new versions are automatically copied over.

Testing

ldd has proven to a convenient mechanism for testing applications before they are released to the environment. Developers can incorporate a new collection to be tested by just adding the following line to the `custom.depot.proto` in the appropriate environment:

```
app.path:${destroot}/${sys}/local/app/004
```

In this case, the files for the application `app` would come from that path listed above, even if that collection already exists in any of the directories specified by the searchpath. When the developer is done testing, he can remove the line, run **ldd** and the software environment would be restored to the previous state.

Local Customization

With **ldd** it is possible to install departmental or machine specific customizations to the software environment. For example, we compile **X11** as it is distributed from the X Consortium. Any local changes are made in the collection called `x11config`. The files in `x11config` overrides, or replaces, the appropriate files in `x11r4`. If individual workstation owners or departmental workstation managers wanted to have a different `x11config` collection, they could simply use **ldd** to specify a different path to `x11config`, and thereby tailor **X11** without having to compile it themselves.

Another customization example would be the installation of software packages that aren't normally available in the central software environment. The primary benefit of installing software in the same location as the central software is the users would access the new software just as if they were access software provided by the central services. No paths or other configuration information would have to be changed in the user's home directory.

The only danger in customizing the software environment is that the users may get confused. Before **ldd** was available, all Andrew workstations shared the same software environment (e.g. `/usr/local`, and `/usr/contributed`). With **ldd** users may see different defaults and different software depending on if they are using a ''departmental'' Andrew workstation or a ''public'' Andrew workstation. So far, this has not been a problem, neither do we foresee it to be one.

Software Sharing

Software sharing under the **depot** model is very straightforward due to the organization of the software components. In the event a distributed file system is not available, then one could simply **tar** up the collection, transfer it to the remote machine, and then extract it. If the remote machine did not run **depot**, one could just extract the collection in the appropriate directory (such as `/usr/local`).

A more sophisticated example would be to use **ldd** to enhance the departmental software environment. Some departments on campus wish to minimize the dependency on the central services. For example, if the central fileservers or the network to the central services went down, they want to still be able to get their work done. As a result, packages that are considered ''critical'' are compiled are exported via NFS. AFS is still used but only for ''non-critical'' applications, as such, `/usr/local` is a symbolic link to the central software repository and another directory is allocated for the mission critical software. With **ldd** the software environment could still be exported via NFS, if so desired, but the mission critical software could be copied to the local disk of the NFS fileserver, using the methods described previously. This potentially saves the departmental administrator a good deal of time compiling and installing software that has already been compiled and installed. Software that isn't provided by the central services, could, again, be installed locally.

Mobile Computing

ldd provides a simple way of supporting mobile computing and disconnected operations via the copying mechanism previously described. Software used when disconnected can be copied to the local disk and updated each time the workstation is connected via a high bandwidth network, such as Ethernet. At this time, all the linked software is also available, as usual. When the workstation is disconnected, then only the copied applications are available. When the workstation is connected via a low bandwidth network, like SLIP, then the other applications are still available, but access would be considerably slower.

The component nature of **ldd** also facilities mobile computing by making a simple GUI possible. For example, there could be a 'point and click' options for what to copy and what to link as well as buttons for 'connect' and 'disconnect.'

Problems

The first problem is that there is a reasonable amount of wasted resources between **dpp** and **depot**. Both these programs contain code that perform virtually the same functions. This is being addressed with the second major version of **depot**. The new version of **depot** provides a programming interface, so any action that **depot** performs can be accessed internally from another program. Work on rewriting **dpp** to use these libraries is about to commence.

As with any system that automatically updates software, there exists the problem of updating applications that are currently running. As with many other systems, we ignore the issue. Traditionally, this problem is addressed by doing the update when the machine reboots. In general, we have been able to ignore this issue since it has not been a problem.

The use of version numbers to maintain local disk consistency potentially wastes a significant amount of disk space, especially when you increase the update window. The only way around this is to force all the workstations to update when the central environment is updated. Given those choices, we decided to sacrifice the disk space.

Finally, version numbers tend to increase the complexity of the system and give the administrator more work to do. In small sites, it may not be worth using the version numbers for this reason and thereby sacrifice a degree of consistency. For larger sites, this may not be acceptable and what we have is a set of support tools that handle much of the work. An overview of these tools can be found in [Coly92A].

Future Directions

The work on the next major release of **depot** is nearing completion. The two major changes in this release of **depot** include significant performance enhancements and a programming interface. As discussed previously, **dpp** will be rewritten to use these programming libraries, especially since the file format for the configuration file has changed in the new version of **depot**.

The most interesting work is the use of the preprocessor **mpp**. **ldd** takes a step forward in doing workstation administration via libraries and via a more modular/object oriented approach to managing all the files on a workstation. For example, the local administrator will be able to issue specific commands to enable (or disable) specific functionality, such as, a single command to enable anonymous FTP, or a command to disable **fingerd** from running. The end goal is to let the local workstation owner manage as much of the software as he wants in a simple and effective manner, and let the departmental and/or central management services fill in the gap in such a way to maximize the local, departmental, and central resources.

Conclusion

Originally, **ldd** was designed only to move applications off of AFS and thus speed up day to day work. However, as work progressed we saw that the benefits were not limited to just that. It has proven to be a a simple and easy method for testing, customizing, and sharing software. **ldd** has been in use at our site for over a year now, and we foresee it as tool to help us in our mobile computing plans as well as in our general workstation management strategy.

Availability

dpp is available from export.acs.cmu.edu (128.2.35.69) in the `/pub/depot` directory. **dpp** is currently in use on HP/UX 9.x, Ultrix/RISC 4.2A, SunOS 4.1.3 platforms. The code is in ANSI C.

An internet mailing list for **depot** and **ldd** exists. To subscribe, send email to info-cmu-depot-request@andrew.cmu.edu.

References

[Coly92A] Colyer, Wallace; Held, Mark; Markley, David, and Wong, Walter. ''Software Management in the Andrew System.'' AFS *User's Group Proceedings.* Spring 1992.

[Coly92B] Colyer, Wallace, and Wong, Walter. "Depot: A Tool for Managing Software Environments." *LISA VI Proceedings* 1992. pp. 153-162.

[Held92] Held, Mark, and Neuhart, Dawn. *Software Management in the Andrew Distributed UNIX System at CMU.* Computing Services, Carnegie Mellon University. 1992.

[Saty85] Satyanarayanan, M.; Howard, J. H; Nichols, D. A.; Sidebotham N., and Spector A. Z. ''The ITC Distributed File System: Principles and Design.'' *Proceedings of the 10th ACM Symposium on Operating System Principles.* 1985.

[Stol92] Stolarchuk, Michael T. ''Faster AFS'' AFS *User's Group Proceedings.* Spring 1992.

Author Information

Walter Wong graduated from Carnegie Mellon University with a B.S. in Cognitive Science in 1991 and promptly joined the Andrew Systems Group performing various programming and system administration tasks. His current focus is on workstation and software administration issues. Feel free to contact him electronically at wcw+@cmu.edu. Alternatively, U.S. Mail can be addressed to: Computing Services; Carnegie Mellon University; 5000 Forbes Avenue; Pittsburgh, PA 15213-3890.

Appendix A: Sample custom.depot.proto

The following is an ''interesting'' **`custom.depot.proto`** from a DECstation 5000 client running AFS.

```
%define beta
# This changes the ${local} variable to point to the
# 'beta' software tree (e.g.
# /afs/andrew.cmu.edu/system/beta/pmax_ul4/local/depot
%include /afs/andrew.cmu.edu/wsadmin/depot/src/depot.include

*.searchpath:/afs/andrew.cmu.edu/usr13/ww0r/devel/depot/local, ${local}
usemodtimes: true
# use the modification time to determine whether or not a copied file
# should be updated.

# create a variable for my convenience
%define destlocal ${destroot}/${sys}/local

# Software currently being tested. Use the software for these
# collections located at the given path, rather than the files
# for the collections found in the searchpath
depot.path:${destlocal}/depot/020
dpp.path: ${destlocal}/dpp/021

# symlink these files/directories under /usr/local
linktarget:root.client,root.server,man,lib/X11/XP
linktarget:bin/sas,lib/sas/sas

# copy these all the files/directories under these directories
copytarget: bin/rlogin,bin/rsh,fonts/andrew,lib/X11
copytarget: lib/sas/sasexe/base,lib/sas/sasexe/graph,lib/sas/sasexe/stat
copytarget: bin/lpr

# copy all of these collections
wcw.mapcommand: copy
graphon.mapcommand: copy
afs.mapcommand: copy
frame.mapcommand: copy
gnu-emacs.mapcommand: copy
x11r4.mapcommand: copy
x11config.mapcommand: copy
perl.mapcommand: copy
telnet.mapcommand:copy
rlogin.mapcommand:copy
tools.mapcommand:copy

# ignore these directories/files under /usr/local
specialfile: lost+found,tmp
```