NetBuild (version 0.02)

Technical Report UT-CS-01-461

Keith Moore, Jack Dongarra
Innovative Computing Laboratory
Computer Science Department
University of Tennessee, Knoxville
{moore,dongarra}@cs.utk.edu

Abstract. NetBuild is a system for making it easy for programmers to incorporate standardized function libraries into their programs, by freeing them from the need to have such libraries installed locally. NetBuild accomplishes this by allowing users to link in function libraries which reside on network servers as if they were installed locally. This report describes NetBuild's goals, an initial implementation, and issues with installation, configuration, user interface, librarian interface, and security.

1. Introduction

NetBuild is a system for making it easy for programmers to incorporate standardized function libraries into their programs, by freeing them from the need to have such libraries installed locally. Instead, the libraries reside on network-accessible servers. When NetBuild is invoked at compilation time (in lieu of the normal compiler or linker), it determines which of the requested libraries are non-resident, downloads those which are missing, and (using the system's normal compiler or linker) links them in along with resident libraries.

NetBuild was designed with the following goals in mind:

- It should be easy to learn and to use, requiring few changes from the interfaces to which the programmer is already accustomed. Ideally existing Makefiles, scripts, etc. should be usable with few or no changes.
- It should free the programmer from the burden of configuring, compiling, installing, and maintaining standardized function libraries on the programmer's target platforms, while providing equivalent results. It should therefore automatically select the most appropriate version of a function library for the target platform when such is available, and ideally, be able to automatically build the appropriate version if it is not available.
- Similarly, it should be able to use the most up-to-date version of a library available (in order to obviate the need for local maintenance of those libraries). However, it should also be able to ensure repeatable results across multiple compilations over time.
- It should have the ability to perform this function even for high-performance computing applications which may need libraries (e.g. ATLAS [1]) which are finely tuned to the characteristics of a particular target platform.
- It should not introduce new security threats to the user's data, compilation environment, or runtime environment.
- It should make efficient use of network resources so as to avoid unnecessary delay in compilation
- It should be easy to install and configure, even for a non-privileged user.
- It should work in most Internet-connected environments, even those which require that external URL access be made via a proxy, firewall, or NAT.

This report describes an initial implementation of NetBuild, and issues identified in the course of that implementation effort.

2. Installation

There are two ways to install NetBuild - the automatic way and the manual way. They do essentially the same thing but the manual way makes it more obvious what is happening.

Here's the automatic way:

1.Download http://www.cs.utk.edu/~moore/netbuild-installer.sh and save it in a writable directory.

2. Type

```
./netbuild-installer.sh
```

to run the installer script. This does everything you need to configure, compile, and install NetBuild.

Here's the manual way:

1. Download the netbuild tar file from ftp://cs.utk.edu/pub/moore/NetBuild-0.02.tar. Be sure to use binary mode.

2. Extract the contents of the tar file:

```
tar xvfp NetBuild-0.02.tar
```

3. Run the configure script to determine local system parameters:

```
cd NetBuild-0.02/src
./configure
```

Note: by default, configure expects that you want to install NetBuild in a subdirectory of your HOME area named NetBuild. If you want it installed somewhere else you need to specify this on the comnmand-line. For example:

```
./configure --prefix=/usr/local
```

4. Compile NetBuild by typing:

```
make
```

5. Install NetBuild by typing:

```
make install
```

Note: NetBuild needs to have the locations of various files compiled-in, so if you want to change the directory where NetBuild is installed after you have configured NetBuild, you must re-run configure, re-compile, and re-install as follows:

```
make distclean
./configure --prefix=/new/prefix
make install
```

6. For most systems this will suffice to install NetBuild. On some hosts or platforms it may be necessary to modify NetBuild's configuration files (see below) so that NetBuild will understand peculiarities of the local system's compilers, linkers, or run-time environment.

3. Using NetBuild

In order to use NetBuild the directory containing the `netbuild` command must be in the user's PATH. A shell script named `netbuild` is installed in *`prefix`*`/bin` (which is equivalent to `$HOME/NetBuild/bin` if the default prefix is used). A user of sh, ksh, or bash would type

```
PATH=$HOME/NetBuild/bin:$PATH
export PATH
```

while a csh or tcsh user would type

```
set path = ( ~/NetBuild/bin $path )
```

The file `$HOME/NetBuild/bin/netbuild` is a shell script which may be used on different platforms (as long as each one has NetBuild installed in the same location), and may be copied to other directories. However if the other components of NetBuild are reinstalled in a different directory, copies of the `netbuild` shell script must also be updated.

The `netbuild` command creates a special compilation environment which contains substitute versions of the system's normal compilers or linkers. These substitutes (we call them shims) parse the comand-line options that are passed to the compiler, determine which libraries are being requested, determine whether any of these libraries is not resident on the system, and attempt to download any missing libraries. Finally the real compiler or linker is invoked with suitable options to cause it to link in the libraries that have been downloaded in addition to any locally installed libraries.

The shims are stored in a special directory (normally `$HOME/NetBuild/` *`platform`*`/lib`). Typing

```
netbuild command
```

causes *`command`* to be executed in a modified enviroment where the PATH environment variable has been updated to contain that directory prior to any other directory. The netbuild command thus creates an environment where any attempts to invoke these compilers are intercepted by NetBuild. *This modified PATH is **only** used when compiling something under NetBuild.*

So for instance

```
netbuild f77 xyzzy.f -llapack -lblas
```

would invoke `f77` in a NetBuild compilation environment. NetBuild might then realize that the requested libraries *lapack* and *blas* were not resident on the system and download appropriate versions, before invoking the real `f77` compiler. Similarly,

```
netbuild make
```

would invoke the `make` command in the special compilation environment which would in turn be inherited by any compilers invoked by `make`. This allows NetBuild to be used without changes to existing Makefiles.

4. Demo programs

Under the demos subdirectory are the source codes for two demo programs - dgeev and dgesv. To compile either of these programs using netbuild, cd to the appropriate directory and type

```
netbuild make
```

Since these programs use the lapack and blas libraries, NetBuild will download the lapack and blas libraries if they are needed. If you want to try compiling them without NetBuild for comparison, type

```
make clean; make
```

At the present time there are two libraries installed for use by NetBuild - lapack and blas. These libraries are available for three platforms: alphaev6-dec-osf5.0, i686-pc-linux-gnu, and sparc-sun-solaris2.7. Other libraries can be added.

5. Implementation

The heart of NetBuild is a C program that runs on several UNIX-derived and UNIX-like platforms. It works as follows:

- The program checks the name by which it was invoked. If that name ends in `netbuild`, it adds the directory containing NetBuild's shims to the PATH and treats the remainder of the command-line as a command to be invoked with the modified PATH.
- Otherwise, NetBuild parses the command-line arguments as if it were the compiler or linker, identifying options that specify libraries to be linked
- For each of these libraries, NetBuild determines whether those libraries are already installed on the local system.

- For each of the libraries that are not installed, NetBuild consults one or more network servers in an attempt to find libraries which match the characteristics of the target platform. When it finds such a library it will download it to the local system. Previously downloaded libraries are cached so they are not downloaded again if they have not changed.
- The authenticity and integrity of the libraries is verified, and if valid, the libraries are installed in local directories which are private to NetBuild.
- The system compiler or linker is then invoked with extra options to force the newly-downloaded libraries to be linked in along with the resident ones.

The current implementation is characterized as prototype rather than production code. It is intended as a proof-of-concept and a testbed for new features rather than a code base for use by ordinary users. A production version of NetBuild would need to pay much more attention to security and robustness issues.

Details of the implementation are discussed below.

5.1. Option parsing

Since NetBuild is invoked as if it were the normal system compiler or linker, NetBuild needs to be able to understand options that vary from one compiler or linker to another. NetBuild therefore has a configurable parser for command-line options. The parser can be configured on a per-host, per-platform, and per-compiler basis.

The parser need not understand the syntax and semantics of each option. It needs to know which options require additional arguments (so that subsequent arguments beginning with a hyphen are not treated as separate options), which options specify libraries to be linked, and which options specify local directories which should be searched. In the future, it may also need to be aware of options which specify static or dynamic linking (the current version only supports static linking), and options which specify variants of the compiler's target platform (so that it can use the correct libraries if the specified target is different than the default one).

5.2 Searching for local libraries

NetBuild must search local directories to determine whether some of the requested libraries are already resident. Since these directories vary from one target platform to another and from one compiler to another on the same platform, the list of directories which NetBuild consults is configurable. In addition, any directories specified on the command-line are also consulted. Finally, since naming conventions vary from one platform to another, NetBuild can be configured to understand the file naming conventions for libraries on the local platform. For instance, library "foo" might be matched by any of `libfoo.a`, `libfoo.so,` or `libfoo.so.1.2.`

5.3 Identifying target platform characteristics

NetBuild currently uses the GNU `config.guess` program to determine a canonical name for the target platform. This is a string which is generally of the form *`CPUtype-vendor-OSname`* where *`OSname`* also contains a version number. This is nowhere nearly precise enough for NetBuild to meet its goals, but this naming scheme is more-or-less consistently used by a large number of software packages, and it is useful as a starting point. Eventually NetBuild will want to detect many more features, including compiler and version, number of CPUs, CPU instruction set extensions (and whether those are supported by the operating system), cache sizes, etc., to allow for more effective matching of available libraries.

5.4 Identifying available network-resident libraries

NetBuild consults an external web server to identify which libraries might be available for a particular platform. In order to find which versions of library foo are available, NetBuild downloads a file named *BASEURL/foo*. That file consists of lines of the form

```
platform URL
```

which indicate that the library named *foo* for platform `platform` can be found at *URL*. Eventually this format will need to be extended to allow constraints to be specified (e.g. only for use with CPUs supporting MMX extension), and perhaps also to allow preferences to be specified (library X is better than Y if data cache size exceeds Z).

Currently NetBuild only supports downloads over HTTP, thus the URLs must all be `http` URLs.

5.5 Library container file format

NetBuild currently expects downloaded libraries to be in gzip-compressed tar format. The actual library archive is expected to be a component of the tar file. The URL which is specified above includes a "fragment identifier" (#suffix) which contains the name of the component of the tar archive to be extracted. The archive may also contain other components. After downloading the tar file the archive is extracted, renamed as necessary, and copied to the cache directory.

We are using this container file format in order to make immediate use of pre-compiled libraries on the netlib software repository [3,4,5]. In the future, it may be necessary to change the format or to change NetBuild to support multiple container file formats.

5.6 Caching

NetBuild caches files that are downloaded from network so that they are not downloaded again unless necessary. The cache is currently maintained on a per-user basis due to security concerns associated with maintaining a shared cache. Libraries downloaded from servers are stored in a directory whose name is derived from a hash of the (canonicalized) URL from which the library was obtained; a separate metadata file contains the last-modified date of that URL. Subsequent attempts to download that file use the HTTP "if-modified-since" directive which causes the file to be downloaded only if it has been changed. Note that the last change date of the container file which is downloaded may be different than the last change date of the actual library.

5.7 Authenticity and Integrity verification

A module which performs authenticity and integrity verification has been implemented but not yet incorporated into NetBuild. This module uses GNU Privacy Guard (GPG) [6] to verify digital signatures. The signatures can be created with GPG or any of several PGP variants. Because the trust model for netbuild libraries is different from that of normal PGP signatures (just because you trust a signature on a library to be authentic does not mean you trust that it’s okay to execute that library on your computer), the signatures used by NetBuild are kept on a separate key ring in a separate directory.

GPG is used in NetBuild prototypes because it is easy to interface to, readily available, and presumably free of patent issues. However the current implementation requires that GPG be installed in addition to NetBuild. To make it easier for the user to install NetBuild, it would be preferable for the signature verification code to be incorporated directly. A later version of NetBuild might use a different signature format, or support multiple formats.

5.8 Server requirements

NetBuild is designed to use ordinary HTTP servers. It currently requires no CGI or other active content on the server, nor does it depend on server-native file naming conventions. We currently use Apache servers, but any other HTTP server should also work.

6. Future Directions

In the future, we intend to add the following features to NetBuild:

- a more flexible system of matching object libraries to target platform characteristics, sufficient to allow (for example) the best available ATLAS library for the target platform to be selected.
- support for using NetBuild to create libraries to be used by NetBuild, with automatic tagging and cataloging of those object libraries with the proper attributes to allow for effective matching, and (perhaps) support to invoke compilers so that they generate maximally portable and/or maximally efficient code, as needed.
- support for dynamic libraries, including the ability to bind to a library at run-time rather than at link-time.
- ability of NetBuild to know about, and make use of, inter-library dependencies
- support for automatically compiling missing libraries from source code
- support for verification of libraries using digital signatures
- caching via a shared cache directory

7. Acknowledgements

Matt Smith wrote an initial implementation of NetBuild which influenced the current version. Susan Blackford, Eric Grosse, and Piotr Luszczek are gratefully acknowledged for helping to fine-tune this version.

8. References

[1] R. Clint Whaley, Jack Dongarra. ‘‘Automatically Tuned Linear Algebra Software’’. Proceedings, Supercomputing 1998 conference. http://www.supercomp.org/sc98/TechPapers/sc98_FullAbstracts/Whaley814/INDEX.HTM

[2] R. Whaley, A. Petitet, Jack Dongarra. ‘‘Automated Empirical Optimization of Software and the ATLAS Project’’ *Parallel Computing*, 27 (1-2) p. 3-25, 2001.

[3] Jack J Dongarra and Eric Grosse. ‘‘Distribution of mathematical software via electronic mail’’ *Communications of the ACM* v 30, n5 (May. 1987), pp 403 - 407. http://www.acm.org/pubs/articles/journals/cacm/1987-30-5/p403-dongarra/p403-dongarra.pdf

[4] Shirley Browne, Jack Dongarra, Eric Grosse, and Tom Rowan. ‘‘The Netlib Mathematical Software Repository,’’ *D-Lib Magazine*, September 1995. Electronic journal, http://www.cnri.reston.va.us/home/dlib/september95/09contents.html

[5] Netlib - http://www.netlib.org

[6] Gnu Privacy Guard. http://www.gnupg.org/

Appendix - NetBuild file layout (and how to uninstall)

The NetBuild installation process stores files in the following directories relative to *prefix*. By default *prefix* is $HOME/NetBuild.

- *prefix*`/bin` - contains a shell script named netbuild which determines the current platform and invokes the correct binary netbuild for that platform. this allows NetBuild users who access multiple platforms to put a single platform-independent directory in their PATH.
- *prefix*`/lib` - contains the config.guess shell script, which determines the current platform
- *prefix*`/`*cputype-vendor-osname*`/bin` - contains the netbuild binary for that platform
- *prefix*`/`*cputype-vendor-osname*`/lib` - contains symlinks pointing to netbuild for each of the compilers on that platform
- *prefix*`/`*cputype-vendor-osname*`/etc` - contains configuration files for that platform

In addition, NetBuild caches library files in `$HOME/.netbuild/cache` and writes temporary files (when downloading) in `$HOME/.netbuild/temp.`

NetBuild may therefore be uninstalled by typing:

```
rm -r $HOME/NetBuild $HOME/.netbuild
```

and by removing `$HOME/NetBuild/bin` from your `PATH.`