distcc User Manual

Martin Pool mbp@samba.org

$Date: 2002/05/17 05:45:54 $, for distcc-0.2

Contents

Chapter 1

Introduction

"Speed, it seems to me, provides the one genuinely modern pleasure." – Aldous Huxley (1894 - 1963)

1.1 Overview

distcc is a program to distribute compilation of C code across several machines on a network. distcc should always generate the same results as a local compile, is simple to install and use, and is often significantly faster than a local compile.

Unlike other distributed build systems, distcc does not require all machines to share a filesystem, have synchronized clocks, or to have the same libraries or header files installed.

Compilation is centrally controlled by a client machine, which is typically the developer's workstation or laptop. The distcc client runs on this machine, as does *make*, the preprocessor, the linker, and other stages of the build process. Any number of "volunteer" machines help the client to build the program, by running the C compiler and assembler as required. The volunteer machines run the `distccd` daemon which listens on a network socket for requests.

distcc sends the complete preprocessed source code across the network for each job, so all it requires of the volunteer machines is that they be running the `distccd` daemon, and that they have an appropriate compiler installed.

distcc is designed to be used with GNU make's parallel-build feature (`-j`). Shipping files across the network takes time, but few cycles on the client machine. Any files that can be built remotely are essentially "for free" in terms of client CPU.

1.2 Author

distcc was written by Martin Pool.

distcc was inspired by Andrew Tridgell's ccache program.

1.3 Licence

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

The author understands the GNU GPL to apply to distcc in the following way: you are allowed to use distcc to compile a non-free program, or to call it from a non-free Make, or to call a non-free compiler. However, you may not distribute a modified version of distcc unless you comply with the terms of the GPL: in particular, giving your users access to the source code and the right to redistribute it, and clearly identifying your changes.

If you find distcc useful, I would appreciate you writing an email to tell me.

1.4 Security Considerations

distcc should only be used on networks where all machines and all users are trusted.

The distcc daemon, `distccd`, allows other machines on the network to run arbitrary commands on the volunteer machine. Anyone that can make a connection to the volunteer machine can run essentially any command as the user running `distccd`.

distcc is suitable for use on a small to medium network of friendly developers. It's certainly not suitable for use on a machine connected to the Internet or a large (e.g. university campus) network without firewalling in place.

`inetd` or `tcpwrappers` can be used to impose access control rules, but this should be done with an eye to the possibility of address spoofing.

In summary, the security level is similar to that of old-style network protocols like X11-over-TCP, NFS or RSH.

1.5 Getting Started

Four straightforward steps are required to install and use distcc:

1. Compile and install the `distcc` package on the client and volunteer machines.
2. Start the `distccd` daemon on all volunteer machines.
3. On the client, set the `DISTCC_HOSTS` environment variable to indicate which volunteer machines to use.
4. Set the `CC` variable or edit Makefiles to prefix distcc to calls to the C compiler.

Chapter 2

Using distcc

2.1 Invoking distcc

distcc is prefixed to C compiler command lines and acts as a wrapper to invoke the compiler either on the local client machine, or on a remote volunteer host.

For example, to compile the standard application program:

```
distcc gcc -o hello.o -c hello.c
```

Standard Makefiles, including those using the GNU autoconf/automake system use the **$CC** variable as the name of the compiler to run. In most cases, it is sufficient to just override this variable, either from the command line, or perhaps from your login script if you wish to use distcc for all compilation. For example:

```
make CC='distcc gcc'
```

NOTE: You cannot just set `CC=distcc`, because distcc needs to know the name of the real compiler.

2.2 Options

distcc only accepts a single option, `--help`, which causes it to print a usage message and exit, as does invocation with no arguments. All other options and arguments are understood as the name of a compiler, followed by arguments and options for the compiler.

2.3 Environment Variables

The way in which distcc runs the compiler is controlled by a few environment variables.

NOTE:

Some versions of make do not export Make variables as environment variables by default, and also that assignments to variables within the Makefile may override their definitions in the environment. The most reliable method seems to be to set `DISTCC_*` variables in the environment of Make, and to set `CC` on the right-hand-side of the Make command line. For example:

```
$ DISTCC_HOSTS='localhost wistful toey'
$ export DISTCC_HOSTS
$ make CC='distcc gcc' all
```

`DISTCC_HOSTS`

Space-separated list of volunteer hosts.

`DISTCC_VERBOSE`

If set, distcc produces explanatory messages on the standard error stream. This can be helpful in debugging problems. Bug reports should include verbose output.

`DISTCC_LOG`

Log file to receive messages from distcc itself, rather than stderr.

2.4 Which Jobs are Distributed?

Building a C program on Unix involves several phases:

- Preprocessing source (`.c`) and headers (`.h`) to a preprocessed file (`.i`)
- Compiling preprocessed source (`.i`) to assembly instructions (`.s`)
- Assembling to an object file (`.o`)
- Linking object files and libraries to form an executable, library, or shared library.

distcc only ever runs the compiler and assembler remotely. The preprocessor must always run locally because it needs to access various header files on the local machine which may not be present, or may not be the same, on the volunteer. The linker similarly needs to examine libraries and object files, and so must run locally.

The compiler and assembler take only a single input file, the preprocessed source, produce a single output, the object file. distcc ships these two files across the network and can therefore run the compiler/assembler remotely.

Fortunately, for most programs running the preprocessor is relatively cheap, and the linker is called relatively infrequent, so most of the work can be distributed.

distcc examines its command line to determine which of these phases are being invoked, and whether the job can be distributed. The command-line scanner is intended to behave in the same way as gcc. In case of doubt, distcc runs the job locally.

In particular, this means that commands that compile and link in one go cannot be distributed. These are quite rare in realistic projects. Here is one example of a command that could not be distributed:

```
$ distcc gcc -o hello hello.c
```

2.5 Running Jobs in Parallel

Moving source across the network is less efficient to compiling it locally. If you have access to a machine much faster than your workstation, the performance gain may overwhelm the cost of transferring the source code and it may be quicker to ship all your source across the network to compile it there.

In general, it is even better to compile on two or machines in parallel. Any number of invocations of distcc can run at the same time, and they will distribute their work across the available hosts.

distcc does not manage parallelization, but relies on Make or some other build system to invoke compiles in parallel.

With GNU Make, you should use the `-j` option to specify a number of parallel tasks slightly higher than the number of available hosts. For example:

```
$ export DISTCC_HOSTS='angry toey wistful localhost'
$ make -j5
```

2.6 Choosing a Host?

The `$DISTCC_HOSTS` variable tells distcc which volunteer machines are available to run jobs.

distcc uses a simple locking heuristic on each client to keep track of which volunteer machines are likely to be busy. distcc prefers to distribute jobs to machines that are not already running a job from this client, and prefers machines occurring earlier in the list of hosts.

distcc does not explicitly coordinate jobs injected from multiple users or client machines.

If only one invocation of distcc runs at a time, it will always execute on the first host in the list. (This behaviour is not guaranteed, however.)

2.7 Diagnostic Messages

distcc prints a message when it runs a command locally or remotely. For more information, set `$DISTCC_VERBOSE` and look at the server's log file.

By default, distcc prints diagnostic messages to stderr. Sometimes these are too intrusive into the output of the regular compiler, and so they may be selectively redirected by setting the `$DISTCC_LOG` environment variable to a filename.

2.8 Exit Code

The exit code of distcc is normally that of the compiler: zero for successful compilation and non-zero otherwise. Error messages from local or remote compilers are passed through to diagnostic output on the client.

If distcc fails to distribute a job to a selected volunteer machine, it will try to run the compiler locally on the client. If that fails, distcc will return exit code 1.

distcc tries to distinguish between a failure to distribute the job, and a "genuine" failure of the compiler on the remote machine, for example because of a syntax error in the program. In the second case, distcc does not re-run the compiler locally.

2.9 distcc with ccache

distcc works well with the ccache tool for caching compilation results. To use the two of them together, simply set

```
CC='ccache distcc gcc'
```

2.10 File Metadata

distcc transfers only the binary contents of source, error, and object files, without any concern for metadata, attributes, character sets or end-of-line conventions.

distcc never transmits file times across the network or modifies them, and so should not care whether the clocks on the client and volunteer machines are synchronized or not. When an object file is received onto the client, its modification time will be the current time on the client machine.

Chapter 3

The distccd Server

The distccd server may be started either from a super-server such as `inetd`, or as a stand-alone daemon.

distccd does not need to run as root and should not.

distccd does not have a configuration file; it's behaviour is controlled only by command-line options and requests from clients.

3.1 Invoking distccd

3.1.1 Common options

These options may be used for either inetd or standalone mode.

`--help`

Explains usage of the daemon and exits.

`--version`

Shows the daemon version and exits.

`-N`, `--nice NICENESS`

Makes the daemon more nice about giving up the CPU to other tasks on the machine. *NICENESS* is a value from 0 (regular priority) to 20 (lowest priority). This option is good if you want to run distccd in the background on a machine used for other purposes.

Chapter 4

distcc Bugs

4.1 Reporting Bugs

If you think you have found a bug, please check the manual and the `HACKING` file to see if it is a known restriction. If not, please send a clear and detailed report to Martin Pool `mbp@samba.org`. (For a clear and detailed description of "clear and detailed", see Simon Tatham's advice on reporting bugs, <http://www.chiark.greenend.org.uk/~sgtatham/bugs.html> .)

4.2 Known Bugs and Restrictions

There are no known cases where distcc will produce incorrect code, but they may exist. There are some restrictions on distcc, and some possible optimizations that are not yet implemented.

- Server-side errors are not directly visible to the client. (The user needs to look at the server's log file.) The server's error messages should be passed back to the client.
- distcc waits for too long on unreachable hosts.
- There is no way to specify a non-default TCP port.
- Attackers can cause arbitrary damage if they can connect to the volunteer's port, or impersonate a volunteer. distcc should therefore only be used on trusted networks. Running over `ssh` or some other security mechanism might be possible, but would be slow.
- 4200 is not a registered port; I just picked it out of my hat. If distcc proves popular, it ought to get a proper IANA-allocated port.
- distcc probably has portability bugs on systems other than GNU/Linux, but they should be accidental, not fundamental.
- distcc ought to work with compilers other than GNU cc, but it has not been tested.
- If a Makefile contains race conditions that make it unsafe for parallel execution then distcc will lose in the same way as a local compiler. Limitations of the Make language mark it hard to write some parallel rules correctly.
- If the Makefile hardcodes the name of the compiler rather than using `$(CC)` then it may have to be updated to work properly. ccache handles these situations by allowing itself to be installed

in place of `gcc`. It examines the name under which it was invoked and decides to run another compiler. It may be possible for `distcc` to piggy-back on that.

- distcc could easily handle C++ programs, but it does not yet recognize their file extensions.
- There are probably some valid compiler command lines that distcc will fail to understand, and which will therefore be run locally rather than distributed. `cc` argument parsing is complex and not completely standardized.
- distcc (by design) can't handle compilers that need to read other files from the local filesystem. This might be a problem with such things as profile-directed optimizers.
- distcc's protocol and file IO would probably have trouble with source or object files over 2GB in size. I've never heard of a file that large, and I rather suspect gcc would not handle them well either.

Chapter 5

Results

TODO (see HACKING)

Chapter 6

distcc Internals

6.1 Protocol

distcc uses a simple, application-specific protocol running directly over a TCP socket. A new request socket is opened for each job.

The request and response begin with a magic number and version number, allowing incompatible versions or misconfigurations to be identified. At the moment there is only one deployed protocol version, and no attempt to support backward or forward compatibility, though this could be added in the future.

The request and response consist of tagged, length-preceded elements. Each element of the request contains a four-character ASCII token, an eight-digit ASCII hexadecimal length or value, and, depending on the tag, a byte stream whose length is determined by the hexadecimal field.

The complete request is sent to the server before the reply begins. Opening the TCP socket is performed concurrently with execution of the preprocessor on the client.

The request from the client contains

1. Magic number and version
2. Compiler command line
3. Preprocessed source code

The response from the server contains

1. Magic number and version
2. Compiler exit code & status
3. Compiler error messages
4. Compiler stdout
5. Object file (if any)

Consult the source for more information.

6.2 Working files

distcc stores working files in a subdirectory of `/tmp`. These include synchronization files, and compiler input/output temporary files.

Temporary files should normally be cleaned up when the program exits. If distcc misbehaves, these files may be useful in tracking down the cause. Any that remain can be removed by the system's temporary file reaper, or by hand.

6.3 Lock files

distcc uses lock files to allow each client to balance its jobs across available volunteer machines. For each volunteer host, a zero-length file is created. Clients using that volunteer hold a `flock` lock on the file while running.