Implementation and Analysis of Win32 Native Distributed Compilation System - *WinDistcc*

Kyongjin Jo[1], Kwanghoon Choi[2], Jongkook Kim[1] and Seon Wook Kim[1]
[1]School of Electrical Engineering, Korea
1, 5-ka, Anam-dong, Sungbuk-ku, Seoul 136-701, Korea
[2]Mobile Handset R&D Center, LG Electronics
327-23, Gasan-dong, Gumchon-gu, Seoul 153-802, Korea
E-mail: [1]{seon}@korea.ac.kr

Abstract: Many software vendors are suffering from heavy compilation overload because the size of their software product is getting bigger and bigger. One of the promising solutions to reduce the compilation time is to use a distributed compiler. It allows us to compile multiple files on several machines concurrently. However most of distributed compilers can't deliver ideal performance due to many undesired overheads such as communication overhead, lack of resources, load imbalance, file dependence, and so on. In order to study the detailed performance matrices, we developed the Win32 prototype of a distributed compiler based on *distcc* (GNU distributed compiler) [1], [2], called *WinDistcc*. *WinDistcc* contains additional features based on *distcc*'s basic functions. The compiler supports two kinds of compilation modes: a local preprocessing or a remote preprocessing. We measured the performances in both cases and identify reasons of performance degradation in normal distributed compilers. Based on the performance study, we could understand the design requirement for ideal distributed compilers.

1. Introduction

In large-scale software projects such as software development on mobile phones, the heavy compilation time is one of serious bottlenecks in terms of productivity. In order to resolve the compilation problem, we developed the Win32 native distributed compilation system, called *WinDistcc* which is based on *distcc* (GNU distributed compiler) [1]. An original distributed compilation process consists of several steps such as local preprocessing on a local server, sending the preprocessed files from the local server to remotes, compilation of the sent file on remotes, sending an object file to the local server from remotes and linking objects and building an executable code at the local server. The local server means a server computer which has source files and a remote does the remote host computer which is connected with a local server for distributed compilation.

These processes work well in a light project which has small amount of preprocessing jobs. However, if a project has large amount of preprocessing jobs, then a local server suffer from overhead of local preprocessing jobs and managing remote machines. To avoid this problem, *WinDistcc* can support a remote preprocessing that preprocesses the source file from remote hosts. In order to support the remote preprocessing, we made a group data structure by analyzing dependencies between source files and header files. This structure provides the dependence information between files that should be sent to remote hosts for remote preprocessing. Through our approach we could reduce the compilation time and with a remote preprocessing we could exploit more parallelism.

This paper consists of three parts: Section 2.explains an architecture of *WinDistcc* and *distcc*. Section 3shows the performance evaluation results, and Section 4makes conclusion of this paper.

This work has been supported by Global LG Track Program of LG Electronics.

2. *WinDistcc*

In order to develop *WinDistcc*, we investigate *distcc* first, because it's the parent of *WinDistcc*. In this chapter, we'll explain the architectures of *distcc* and *WinDistcc* both. Based on the knowledge, we'll explain overall details of *WinDistcc* implementation issues.

2.1 Architecture of *distcc*

distcc is the representative open-source distribute compiler made by GNU society [1], [2]. It consists of two parts, a server and a remote host. The server manages overall distributed compilation jobs such as the remote host selection, a preprocessing, a result collection and so on. The server processes are invoked by *make* utility [3], [4] and are working during given compilation jobs. The remote host performs compilation jobs which are requested by the server process.

The basic concept of *distcc* were described in Figure 1. It uses only a local preprocessing scheme. The server process receives compilation command from *make* utility and chooses a non-busy remote host from the list of remote hosts. The server connects with the remote host and performs preprocessing jobs on the local server. After the preprocessing job is done, the server sends the preprocessed file and the compilation command to the remote host. The remote side is waiting for the server's requests and preprocessed files until whole distributed compilation jobs completely finish. The remote receives the preprocessed file and compilation command from the server. After that, the remote host side performs compilation and sends a compiled code such as an object file to the server. Since the dependence between source files is required for preprocessing, it is the easiest way to performance the preprocessing on the local server. However, the preprocessing jobs can't be processed in parallel (i.e. serial execution is increased) so this scheme results in serious performance degradation for distributed compilation.

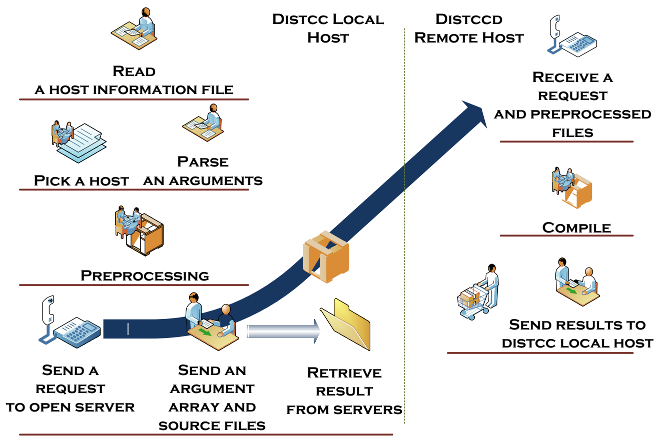

Distcc Local Host
Distccd Remote Host
Read a host information file
Pick a host
Parse an arguments
Preprocessing
Send a request to open server
Send an argument array and source files
Retrieve result from servers
Receive a request and preprocessed files
Compile
Send results to distcc local host

Figure 1. *distcc*’s basic operation.

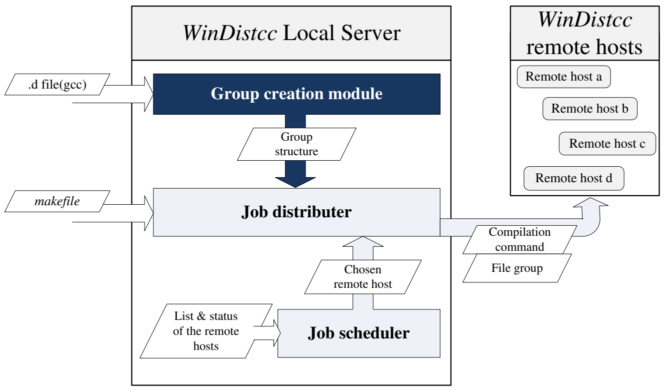


Figure 2. *WinDistcc* architecture.

2.2 Architecture of *WinDistcc*

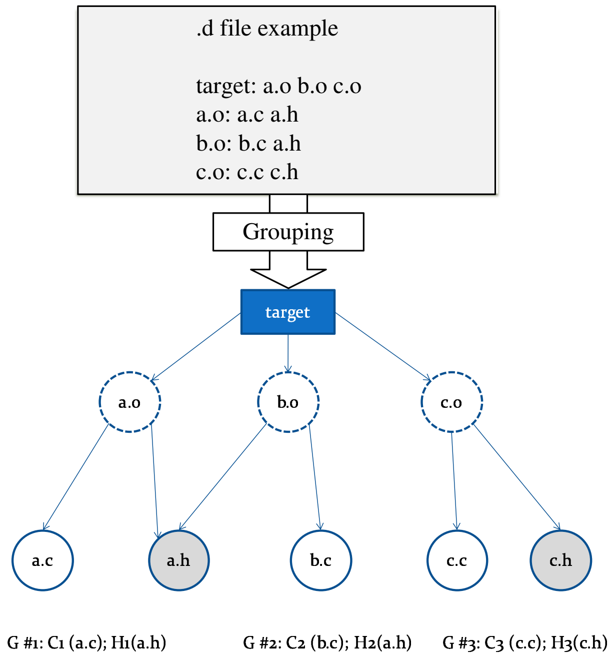
WinDistcc is based on *distcc*, but more smart modules to exploit higher parallelism. *WinDistcc* works on a local server and several remote hosts, as shown in Figure 2. Each remote host has a very simple structure to consist of a network handler, a thread handler and a compiler. A local server has the following functionalities: a group creation module, a job distributer, a job scheduler and a network handler. Except the group creation module, other modules are already available in the original *distcc*.

Before launching distributed compilation process, *WinDistcc* executes *gcc* [5], [6] first to get dependency information file. The group creation module gets a *.d* file from the previous operation. The module analyzes the contents the file and makes group structures from it. Simple grouping scheme is depicted in Figure 3. The job distributer considers *Makefile* and list of remote hosts, and chooses the remote host who will perform a compilation job. With group information and *Makefile*, the job distributer sends compilation command and group associated files to chosen remote hosts. After the compilation job is finished in remote hosts, each remote hosts sends an object file to the local server.

2.3 Implementation of *WinDistcc*

We performed three processes for *WinDistcc* implementation. First, we migrated the Linux version of *distcc* to Win32 native version. In order to migrate the compiler to Win32 version, we implemented a thread based job preprocessing system and a winsock style communication system.

Second, we implemented grouping operation for the remote preprocessing. The simplest way to extract dependency information is parsing *Makefile*. However this approach can’t give all file dependencies such as a source file and its whole related header files. To resolve this problem, we extracted dependency information from ’.d’ file from *gcc*. With this information, we made several groups which contain dependency related files inside them.



```
.d file example

target: a.o b.o c.o
a.o: a.c a.h
b.o: b.c a.h
c.o: c.c c.h
```

Figure 3. Grouping scheme.

Third, we implemented group based distributed compilation scheme. According to the group information, *WinDistcc* sends all the files in one group to a specific remote host. Since the remote host has all files required for the compilation, the remote host doesn’t need further file transmission. This concept is described in Figure 4. Differently from the original *distcc*, the remote host in *WinDistcc* performs both preprocessing and compilation job.

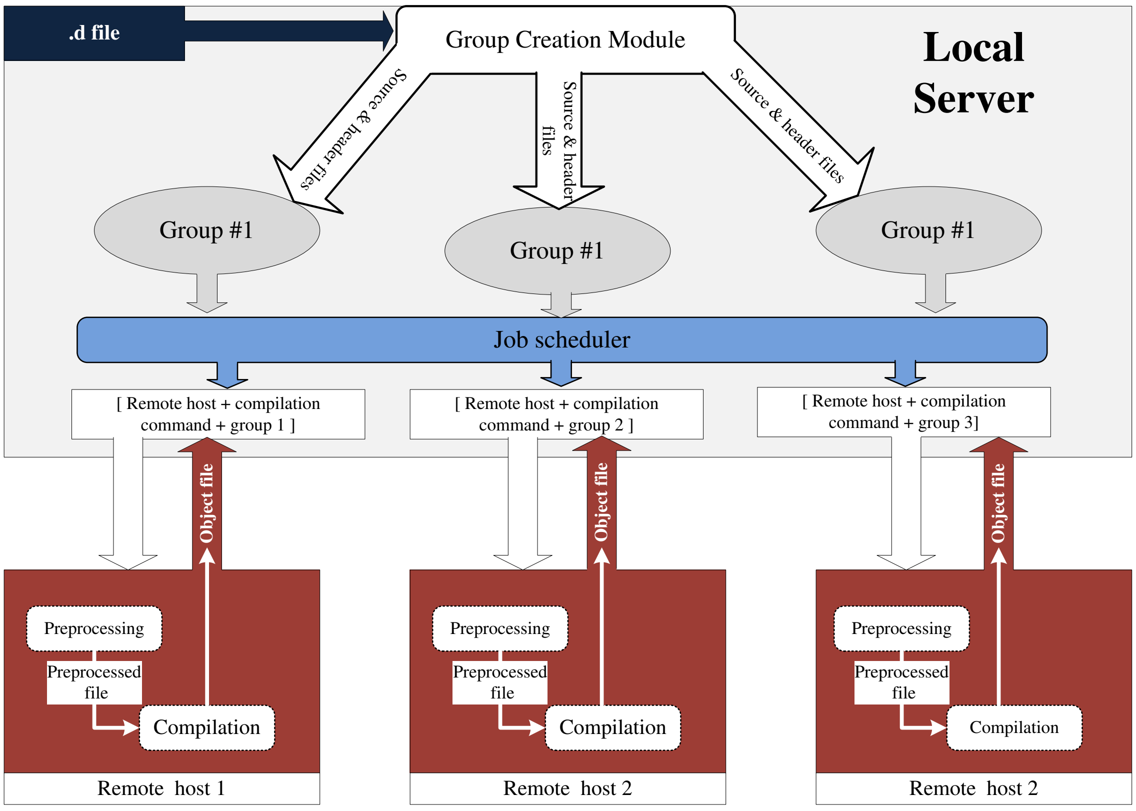


Figure 4. Group operation of *WinDistcc*.

3. Performance evaluation

3.1 Evaluation Method

In order to evaluate our distributed compiler framework, *WinDistcc*, we used two kinds of evaluation methods: with and without remote preprocessing support. Table 1 shows performance evaluation environment. We picked three projects to measure the performance, make-3.81, tcl 8.4.1 and tk 8.4.1. These three projects are Linux based, so we used cygwin for Win32 environment. Actually, we used *make* utility for distributed compilation jobs so we can set the number of compilation threads with a jobserver function in *make* utility. If the number of compilation threads is more than one, then the remote host will create multiple compilation threads simultaneously. In order to let the remote host busy during whole distributed compilation process, we set the number of threads as two. If we set more than two as the number of compilation threads, then the remote host suffers from resource contention.

Table 1. Performance evaluation environment.

Project for compilation	Make-3.81, tcl 8.0, tk 8.1	
Machine specification for compilation	CPU	Intel Core(2) Duo 1.83GHz (for the server), AMD Athlon 3200+(for the remote hosts)
	RAM	DDR2 2GByte (for the remote hosts) / 3GByte (for the server)
Network	100MB LAN without firewall	
Operating System	Cygwin on MicroSoft Windows XP Professional	
Number of compilation thread	2 threads per remote host	

3.2 Evaluation Result

Figure 5 shows average speedup of *WinDistcc* with and without (w/o) remote preprocessing support with respect to server-only compilation. In case of w/o remote preprocessing, it shows good performance improvement from 1 remote host to 2 remote hosts, and the speedup is almost linear. However, with more than 3 remote hosts this architecture doesn't get scalable performance. The main reason of this behavior is overhead about multiple preprocessing and remote managing jobs at the local server. In case of with remote preprocessing

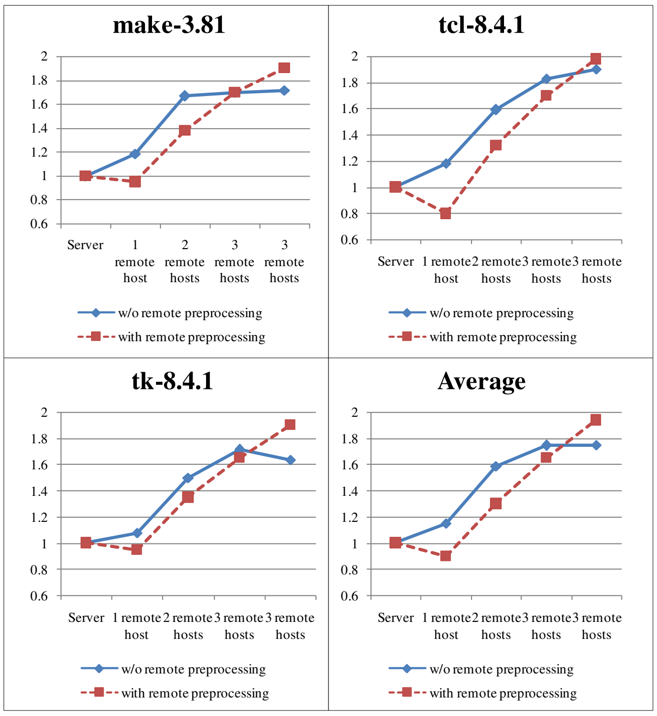


Figure 5. Speedup of *WinDistcc* with or w/o a remote preprocessing support, normalized server = 1.

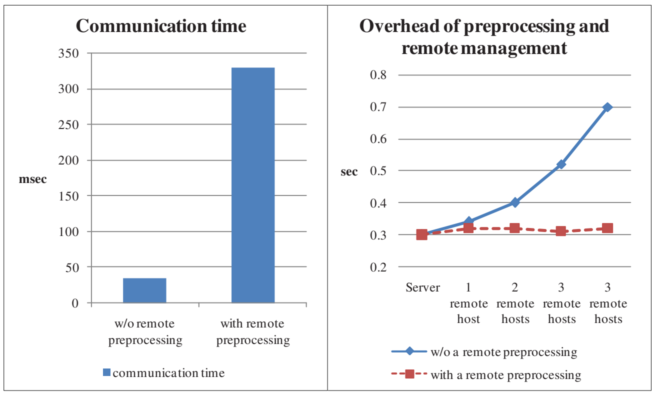


Figure 6. Execution time analysis.

support, its base performance isn't better than the local preprocessing version. Its inferior base performance is caused by communication overhead due to sending a code and its associated all headers. Nevertheless, remote preprocessing support version shows scalable performance because the server has a surplus energy to manage the distributed compilation jobs.

In Figure 5, there are evidences of a remote preprocessing version *WinDistcc*'s worse base performance and more scalable performance than w/o a remote preprocessing version *WinDistcc*. In terms of the communication time, a remote preprocessing version *WinDistcc* consumes 10 times more time than w/o remote preprocessing version, and this time consuming behavior is the reason of bad base performance. However, in terms of the overall overhead, a remote preprocessing version *WinDistcc* is free from the overhead about preprocessing and a remote management. On the contrary, as the number of remote hosts increases, the overall overhead also increases in w/o remote preprocessing version. In w/o remote preprocessing method, we can distribute compilation jobs to each remote host, but heavy local preprocessing jobs on a local server prevents remotes from connecting with the local server whenever remotes finish their compilation jobs.

4. Conclusion

In this research, we developed a remote preprocessing support Win32 native distributed compiler. By providing the remote preprocessing support we could avoid overhead at a local server caused by local preprocessing and remote management. The performance analysis showed that the remote preprocessing approach provides performance scalability. With these results, we can know the reasons of un-ideal performance of the existing distributed compilers. According to the result of this research, we can make more enhanced distributed compiler in near future. Moreover, we developed GUI for *WinDistcc* for providing convenient compilation environment to users.

References

[1] http://distcc.samba.org
[2] T.A. Jones, "distcc, a fast free distributed compiler," *linux.conf.au*, December 2003.
[3] Becker, B. "A GNU Make Tutorial", http://www.undergrad.math.uwaterloo.ca/ cs241/make /tutorial/index.html, January 1996.
[4] Stallman, R & McGrath Roland. "GNU Make", http://csugrad.cs.vt.edu/manuals/make/make_toc.html, December 1993.
[5] http://gcc.gnu.org
[6] R. Stallman. "The GNU C compiler". Free Software Foundation, 1991.