Distributed compilation with Distcc and Icecream clusters using Gentoo, a benchmarking study case

Víctor L. Orozco[1], **José F. Lobos**[2]

[1]Centro de Tecnologia – Universidade Federal de Santa Maria (UFSM)
Av. Roraima 1000 – 97.105-900 – Santa Maria – RS – Brasil

[2]Escuela de Ciencias y Sistemas – Universidad de San Carlos de Guatemala (USAC)
Guatemala - Guatemala

Abstract. *A GNU/Linux system is the product of a collaborative process that involves the creation of a collection of software packages developed by different developers in different places. The result of that process is a big collection of code that needs to be compiled and grouped in a single distribution package; this task requires a great computational power. This work presents a benchmarking review of two of the most popular tools for distributive compilation -Icecream and Distcc-. The comparative was done by measuring the computational complexity factors proposed by Blackburn, Van Benthem and Wolter in Handbook of Modal Logic over a compilation cluster constructing a GNU/Linux distribution based on Gentoo Linux.*

1. Introduction

GNU/Linux systems are product of a collaborative process that includes selection, personalization, compilation and packaging of software applications from different origins around the world. Although every phase has its own goals and technical challenges, the compilation phase is a big computational resources demander.

In the book written by [Ifrah 2002] he describes the evolution of the computers from the abacus to the modern computers and presents an important keypoint on the evolution of computers: the fact that computers are product of a constant evolution of calculation machines. Although the functionality of the computers of our days covers many knowledge fields, the way to do the work is the same: perform calculations over information in order to produce a result. Then, when is necessary to increase the capacity of a system, is right to affirm that is necessary to improve its calculation capacity to get results faster. However the increment of computational resources is limited by the technology that exists at the time.

There are two ways to increase the capacity of a computational system, increase the computational resources on a single machine or increase the number of the computers involved in the task, techniques that are known as horizontal and vertical scalability [Michael et al. 2007]. The problem with vertical scalability is the fact that the maximum scalability is limited by the maximum capacity of the hardware. On the contrary horizontal scalability is a good option to get an scalable system if the task that demands the resources could be divided in a fixed set of computational processing units. This kind of systems are known as computational clusters, a set of computers working together in order to complete a great workload task in the lowest possible time.

Source code compilation is a perfect task to be executed in a cluster because a binary file is the result of the compilation of many source code files and that files could be distributed and compiled in many computers at the same time. Icecream and Distcc are popular tools to achieve compilation clusters construction. They exhibit many similarities because Icecream was created with the Distcc source code. Icecream creators affirm that although the tool was created with the source code of Distcc, they have improved the performance by introducing a dinamic scheduler of the compilation jobs that chooses the fastest free server. They affirm that this advantage is specially apreciated in environments where the nodes of the cluster do not have full dedication for cluster tasks[Icecream project 2012].

This work presents a comparative review based on a performance benchmarking with the objective of confirm if the changes done by Icecream creators improved the performance of the compilation process compared to Distcc and Distcc working in pump mode. The work also presents the main principles about clusters and distributed workload including the construction of binary executables with Gnu Compiler Collection, these concepts are described because are base concepts to understand the functionality of Distcc and Icecream. Later the tools performance is compared by measuring their computational resources utilization, according to computational complexity theory by [Blackburn et al. 2007]. To achieve this objective with significant results, a generic GNU/Linux system was constructed over the different kind of clusters.

The rest of the work is organized in the following way. On the second section are presented basic concepts about cluster classification. The third section presents the functionality of GNU Compiler Collection and how a binary executable is constructed. The fourth section presents a brief introduction to Distcc and Icecream tools. The fifth section presents the benchmark test between the two tools. The sixth section presents the discussion about the results of the tests and the seventh section presents the conclusions of the work.

2. Computer clusters classification

Computer clusters can be classified depending on their tasks in three types: High Availability Clusters, High Performance Clusters and High Throughput Clusters. The architecture and the selection of the right kind of cluster depends on the tasks to be executed, specially if these tasks could be executed in a parallel way and if exists independence between the planned tasks on every node of the cluster.

2.1. High Availability Cluster

In High Availability Clusters (HA), a cluster is constructed with a group of computers in order to provide a vision of a big computer that supports more requests than a single computer, HA clusters are built typically with two or more robust machines that mirror one or more tasks to increase quality of service and fault tolerance [Ferreira L. et al. 2001].

2.2. High Performance Cluster

High Performance Clusters (HPC) are used to process parallel algorithms and related tasks, specially in sectors where distributed parallel computing is needed to solve large scientific problems and/or store and process large amounts of data in a parallel way

[Ferreira L. et al. 2001], the only requirement for this kind of cluster is that the tasks should be parallelizable.

2.3. High Throughput Clusters

The idea behind a High Throughput Cluster (HTC) is different respecting a HPC cluster in the fact than a HPC cluster is designed to increase computational power to resolve large tasks in a parallel way. In contrast an HTC cluster is designed to process a lot of information by subdividing the tasks in a non dependant way between the nodes of the cluster and later produce a result as a product of the individual results of the nodes, these kind of tasks are also named embarrassingly parallel [Ferreira L. et al. 2001].

3. Binary executables construction with GCC

Although Icecream and Distcc have experimental support for many C++ compilers, these tools are mainly developed to be used with GNU Compiler Collection (GCC). GCC was originally designed to be a free and open source compiler for the GNU Operating System. Nowadays it supports compilation for almost all architectures of computer processors and is one of the most used compilers on UNIX systems.[Gough and Stallman 2004]

Creating a binary file with GCC is a four steps task. In the first step, GCC process the source code files in order to verify the syntax of the program and verify the existence of the libraries declared in the header files and the headers of the source files of the program, if all is right the source code files are expanded to include the macros for the libraries declared at source files producing new intermediate source code files.

In the second step the source code files are compiled to assembly language and at the third step the source code files in assembly are compiled again to a binary file compatible with the destination architecture, at this step the result is a collection of .o -object- files. Later in the final step the .o files are dynamically linked in a executable binary file.

In the second step the source code files are compiled to assembly language, and at the third step, the source code files in assembly are compiled to a binary file compatible with the destination architecture, at this step the result is a collection of .o -object- files. Later in the final step the .o files are linked in a executable binary file [Gough and Stallman 2004].

4. Distcc and Icecream

The functionality of the tools is achieved by separating the compilation phases of GCC between a main coordinator node and compilation workers nodes as is described in the Figure 1.

In Distcc and Icecream clusters the coordinator node executes the instructions described in the makefiles and also does the first step of GCC compilation process -expand the source code files with the library macros- and it sends the preprocessed source files to the compilation nodes of the cluster, later these nodes send back to the coordinator node the .o compiled files in order to link them in a binary file.

Distcc also presents a special functionality mode called “pump mode”, in that mode the source files are sent in conjunction with their header files to execute the preprocessing step of GCC compilation at the compilation nodes of the cluster.

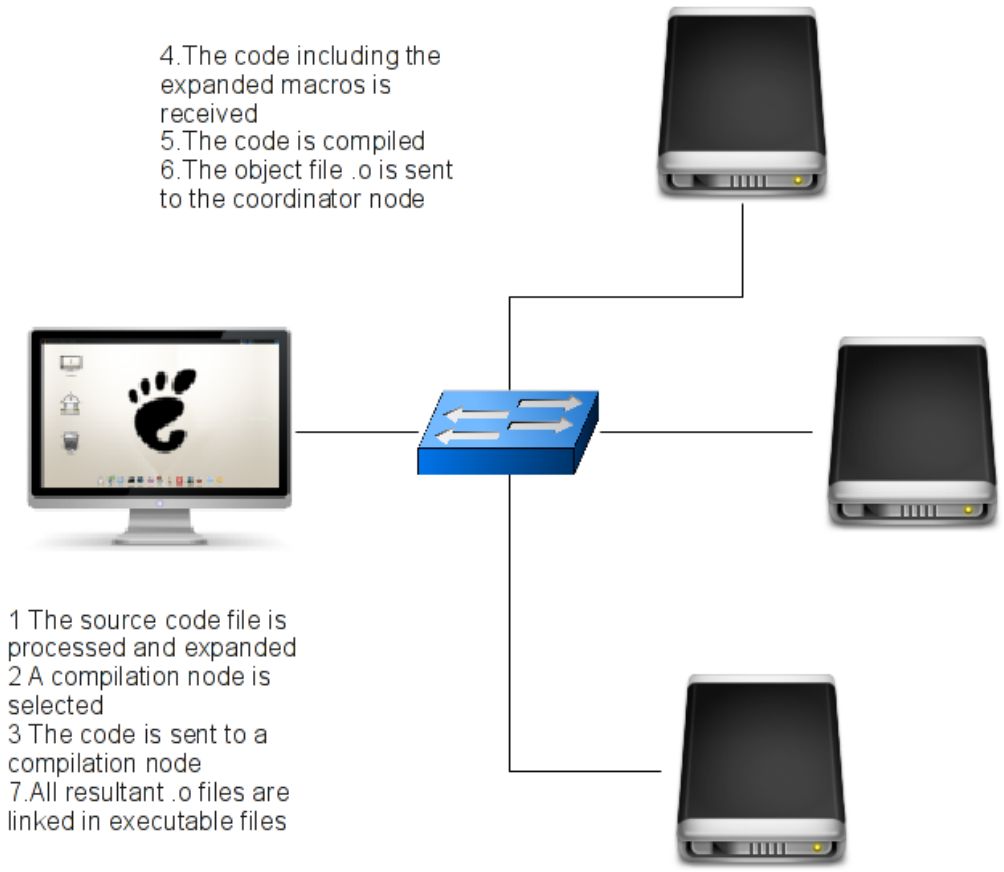

Figure 1. Standard functionality for Icecream and Distcc.

5. Distcc and Icecream benchmark

[Blackburn et al. 2007] present in their work an introduction of computational complexity, they explains that complexity theory is a part of the theory of computation that deals with the required resources during computation to solve a given problem. The most common resources are time (how many steps are required to solve a problem) and space (how much memory it takes), so according with them, the metrics chosen for the experiment were: time required to produce a package (as a replace of the time notion of the number of steps) and the average memory usage for a package compilation (as a space notion). As an extra the total source code size for a package, and average processor usage for a package compilation were also measured.

5.1. Experiment structure

Cluster computers were running a Gentoo Linux 10.0 stage 3 installation with the necessary packages to construct and monitor the Icecream and Distcc tests. Gentoo was chosen because one of its objectives is to provide a metadistribution designed to build systems with emphasis in optimization and customization [Gentoo project 2012] and also because its package manager -called portage- has support for Icecream and Distcc. All tests (Icecream, Distcc and Distcc in pump mode) were executed in the same set of computers and they were restored to a clean configuration for every test execution in order to avoid the variation of the results as a product of the software aging. Sar, Genloop, Distcc-mon and

Icecreamon and also the standard UNIX commands ls and sleep were the selected tools to retrieve the data. The tests were executed in a small cluster with three computers, two computers working as compilation nodes and one computer working as coordinator node, their technical specifications are described at the Table 1.

The executed workload over the clusters was the construction GNU/Linux distribution with a standard KDE desktop environment to generate significant data for a distribution construction. KDE was selected because most of packages are developed with C or C++. At the moment of the test the KDE standard desktop installation required the compilation of 372 packages in a Gentoo stage 3 clean system.

Table 1. Cluster technical specifications.

	Processor	RAM Memory	Operating System	Kernel version
Coordinator Node	Intel Celeron M @ 1.6 ghz	1536 MB	Gentoo Linux 10.0	2.6.32 with Gentoo Patchset
Node 1	Intel Pentium 4 @ 2.8 ghz	768 MB	Gentoo Linux 10.0	2.6.32 with Gentoo Patchset
Node 2	Intel Pentium 4 @ 2.8 ghz	768 MB	Gentoo Linux 10.0	2.6.32 with Gentoo Patchset

5.2. Special tools

5.2.1. Sar

Sar tool is a performance monitor developed to collect, store and generate reports, about cumulative activity counters in the operating system, its count an interval parameters can be personalized to retrieve information about some particular area of a Linux system between short laps of spaces. It was selected because offers a small memory footprint, condition that is ideal to not alter the results of the tests[Sar manual page 2012].

5.2.2. Genlop

Genlop is a Gentoo Linux logs parser utility. It is designed to extract useful information from the portage logs. Between its characteristics it has full portage logs support, and also has support to display date, build time, and build specifications of every constructed package[Genlop manual page 2012].

5.2.3. Distcc-monitor

Distcc-monitor is a tool designed to monitor the actual status of the compilation processes in the member nodes of a Distcc Cluster[Distcc project 2012].

5.2.4. icecream-monitor

Likewise Distcc-monitor, Icecream-monitor is a tool designed to monitor the actual status of the compilation processes in the member nodes of an Icecream Cluster[Icecream project 2012].

5.3. Data capture process

5.3.1. Build time

The build time for every package is stored on the coordinator node at the Gentoo package manager log files in a special format based on epoch timestamps. genlop tool was used to get the data in an easy readable way and later it was dumped to a plain file.

5.3.2. Packages size

The package size was retrieved using the standard UNIX command ls over the directory where the source code files were downloaded by the Gentoo package manager -/usr/portage/distfiles-. Also the h -human format- option was added to ls in order to get the data about the size of the source code of the software packages in a easy readable way.

5.3.3. CPU and memory usage

To get a better impression, the CPU and memory usage was captured in all nodes of the cluster at every thirty seconds using the sar tool in conjunction with sleep standard UNIX command. Later the data was classified based on the timestamps available in sar results contrasting them with the previously obtained timestamps with genlop. Finally the information about average CPU and Memory was calculated using the classified information..

6. Results

6.1. Data classification

The total size of the collected data was 1.1 GB and the difference of the data of one package to another is expressed in milliseconds for the compilation time, and in kilobytes for the memory and package size. That fact introduces considerable variability in the analyzed data and makes it difficult to analyze in an statistical software package

To avoid that difficulty [Pallant 2010] proposes the usage of a categorical data analysis in conjunction with a codebook that converts the captured data to a format that a statistical software package can understand, the codebook is presented at the Table 2.

By example if the source code of a package has a size of 250 KB, and the compilation process results where 70% for the average memory usage, 90% for the average processor usage and 7 minutes with 25 seconds for the compilation time, according to the codebook at Table 2 the corresponding categoric values are *one* for the package size, *four* for the average memory usage, *five* for the processor usage and *two* for the compilation time.

Table 2. Categorical data classification codebook.

Metric	Description	Metric Name	Category
RAM Memory	Memory used while a package is compiling (average time between nodes)	Mem	1= [0-20%) 2= [20-40%) 3= [40-60%) 4= [60-80%) 5= [80%-100%]
Build time	Time to compile the package	Time	1=[0-5) minutes 2=[5-10) minutes 3=[10-15) minutes 4=[15-20) minutes 5=20 and more
Packages Size	Size of the source code files of the package	Size	1= [0-2) MB 2= [2-4) MB 3= [4-8) MB 4= [8-10) MB 5= 10 MB and more
Processor	Processor usage while a package is compiling (average usage between nodes)	Proc	1= [0-20%) 2= [20-40%) 3= [40-60%) 4= [60-80%) 5= [80%-100%]

6.2. Data analysis

After the data classification all the statistical data was analyzed with the help of SPSS to get an overview of the differences between the cluster tools. The results are described following.

6.2.1. Package size

The package size is a constant in the three tests, the summary is presented at the Table 3. Most of the packages are in the first category, below 2 MB. This fact gives an idea of the modularity of the development in KDE and Linux in general.

Table 3. Package size summary

Category	Frequency	Percentage
1) [0-2) MB	322	86.5
2) [2-4) MB	23	6.2
3) [4-8) MB	14	3.8
4) [8-10) MB	0	0
5) 10MB and more	13	3.5
Total	372	100.0

6.2.2. Compilation time

The summary of the compilation time for the three clusters is presented at Table 4 where a smaller value means faster compilation. The *value* column presented by SPSS summary is the multiplication of the category per the frequency (number of occurrences). The information indicates that the Icecream cluster was the faster cluster with a sum of 420, compared to 422 for Distcc and 431 for Distcc with pump mode enabled.

Table 4. Compilation time summary

Category	Test 1 - Distcc			Test 2 – Distcc pump mode			Test 3 - Icecream		
	Frequency	Percentage	Value	Frequency	Percentage	Value	Frequency	Percentage	Value
1) [0-5) minutes	344	92.5	344	342	91.9	342	346	93	346
2) [5-10) minutes	15	4	30	14	3.8	28	12	3.2	24
3) [10-15) minutes	7	1.9	21	8	2.2	24	9	2.4	27
4) [15-20) minutes	3	0.8	12	3	0.8	12	2	0.5	8
5) 20 minutes and more	3	0.8	15	5	1.3	25	3	0.8	15
Total	372	100.0	**422**	372	100.0	**431**	372	100.0	**420**

6.2.3. CPU utilization

The results are presented at Table 5 where a larger value means more CPU utilization. The information is in the same path that the previous results indicating that Icecream has a higher CPU utilization with a score of 1072 and is taking advantage of the computational resources more efficiently compared to Distcc with 1030 and Distcc in pump mode with 964.

Table 5. CPU utilization summary

Category	Test 1 - Distcc			Test 2 – Distcc pump mode			Test 3 - Icecream		
	Frequency	Percentage	Value	Frequency	Percentage	Value	Frequency	Percentage	Value
1) [0-20) %	0	0	0	22	5.9	22	0	0	0
2) [20-40) %	164	44.1	328	165	44.4	330	134	36	268
3) [40-60) %	138	37.1	414	136	36.6	408	157	42.2	471
4) [60-80) %	62	16.7	248	41	11	164	72	19.4	288
5) [80-100] %	8	2.2	40	8	2.2	40	9	2.4	45
Total	372	100.0	**1030**	372	100.0	**964**	372	100.0	**1072**

6.2.4. Memory utilization

The memory usage results also are presented at Table 6 where a larger value means more memory utilization. Again Icecream presented a better score with 880, compared to Distcc with 750 and Distcc in pump mode with 793.

Table 6. Memory utilization summary

Category	Test 1 - Distcc			Test 2 – Distcc pump mode			Test 3 - Icecream		
	Frequency	Percentage	Value	Frequency	Percentage	Value	Frequency	Percentage	Value
1) [0-20) %	24	6.5	24	22	5.9	22	4	1.1	4
2) [20-40) %	318	85.5	636	279	75	558	228	61.3	456
3) [40-60) %	30	8.1	90	71	19.1	213	140	37.6	420
4) [60-80) %	0	0	0	0	0	0	0	0	0
5) [80-100] %	0	0	0	0	0	0	0	0	0
Total	372	100.0	**750**	372	100.0	**793**	372	100.0	**880**

6.2.5. Final summary

The summary of the data is presented at Table 7, Table 8 and Table 9.

Table 7. Categorical data classification - Distcc test

	Min	Max	Sum	Average
Packages Size	1	5	475	1,28
Build Time	1	5	422	1,13
Processor Usage	2	5	1030	2,77
RAM Memory Used	1	3	750	2,02

Table 8. Categorical data classification - Distcc with pump mode test

	Min	Max	Sum	Average
Packages Size	1	5	475	1,28
Build Time	1	5	431	1,16
Processor Usage	1	5	964	2,59
RAM Memory Used	1	3	793	2,13

Table 9. Categorical data classification - Icecream test

	Min	Max	Sum	Average
Packages Size	1	5	475	1,28
Build Time	1	5	420	1,13
Processor Usage	2	5	1072	2,88
RAM Memory Used	1	3	880	2,37

7. Conclusions

According to the tests results, the fastest cluster was the cluster constructed with Icecream with a score of 420, compared to 422 for Distcc and 431 for Distcc with pump mode in the last place. This indicates that the distribution planning done by Icecream effectively increased the original performance of Distcc.

Although Icecream overpassed Distcc performance the average results were too similar. The scores were Distcc 2.77, Distcc with pump mode 2.59, Icecream 2.88 for processor and Distcc 2.02, Distcc with pump mode 2.13 and Icecream 2.37 for memory usage. The difference between one cluster to another is less than 0.3, so for individual compilation of packages the difference of performance will not be so perceptible.

Distcc in pump mode presented less performance compared to normal Distcc. Build time for Distcc with 422 was smaller than Distcc working in pump mode with 431. That conclusion is contrary to the performance improvement affirmed by Distcc creators and is a good candidate to be double checked.

The information relative to resources utilization evidenced that Icecream was the better solution to offer better CPU and memory utilization to reduce the compilation times

References

Blackburn, P., van Benthem, J. F. A. K., and Wolter, F. (2007). *Handbook of Modal Logic,Volume 3* (Studies in Logic and Practical Reasoning). Elsevier.

Distcc project. Available at: <http://code.google.com/p/distcc/>. Last access: May 04 2012.

Ferreira L., Kettmann G., Thomasch A., Silcocks E., Chen J., Daunois J., Ihamo J., M. H., Steve ., W. B., and Ford E. (2001). *Linux Hpc Cluster Installation*. Ibm.

Genlop Manual Page. Available at <http://linuxreviews.org/man/genlop/index.html.en>. Last access: May 04 2012.

Gentoo Linux Project, 2007. Available at: <http://www.gentoo.org/main/en/about.xml>. Last access: May 04 2012.

Gough, B. J. and Stallman, R. M. (2004). *An Introduction to GCC: For the GNU Compilers GCC and G++*. Network Theory Ltd.

Icecream project, 2011. Available at: <http://en.opensuse.org/Icecream>. Last access: May 04 2012.

Ifrah, G. (2002). *The Universal History of Computing: From the Abacus to the Quantum Computer*. Wiley.

Michael, M., Moreira, J., Shiloach, D., Wisniewski, R. W., and Heights, Y. (2007). *Scale-up x Scale-out: A Case Study using Nutch/Lucene*.

Pallant, J. (2010). SPSS Survival Manual: *A step by step guide to data analysis using SPSS.* Open University Press.

Sar Manual Page. Available at: <http://linux.die.net/man/1/sar>. Last access: May 04 2012.