British Journal of Mathematics & Computer Science
4(20): 2884-2900, 2014

SCIENCEDOMAIN *international*
www.sciencedomain.org

Computing Power Utilization of Distributed Systems Using Distributed Compilation: A Clustered HPC Approach

Sunil Kr Singh[1*], Aman Madaan[2], Ankur Aggarwal[1] and Ankur Dewan[1]

[1]Department of CSE, Bharati Vidyapeeth's College of Engineering New Delhi, India.
[2]Department of CSE, Indian Institute of technology (IITB) Mumbai, India.

Received: 15 April 2014
Accepted: 25 June 2014
Published: 29 July 2014

Original Research Article

Abstract

To implement a farm of computer system which compiles source programs using the computers connected in the farm. A build farm can be defined as a set of computers (distributed systems) that work together to compile software. Distributed Compiling is a special type of compiling activity in which we compile the source program using several machines with the same or cross compilers to do the compilation in parallel. In this paper, we describe a system that uses the concept for reducing build times by using free cycles of idle computer systems in distributed. The challenge of distributing compilation is tackled by a distributing compiler. We use distcc for the purpose but It would be nice if distcc could automatically detect the best distribution, but it doesn't do that yet. But in our system we have added a new php script which automatically does this work. Whenever it sees that a system is overloaded or its utilization increases above a threshold value (in our case it is 65%), it automatically removes that node. In this way we can combine the computing powers of individual machines to get a more powerful machine.

Keywords: Virtualization, cloud computing, resource monitoring, service-oriented, Infrastructure As A Service (IAAS).

1 Introduction

Build-Farm is currently a free of cost "Infrastructure As A Service" which provides a grid of heterogeneous systems that may be used to compile or build any software. The service is designed in such a way that the user just has to select the numbers of nodes for compilation of his/her program with reference to the current usage of the system provided in the form of a graph by logging into the website of the system by providing a username and password, which once verified allows the user the capability to build any program residing on user's system. The feature that makes this project an different from the current environment and useful one is that it uses

**Corresponding author: anujsunilsingh@yahoo.co.in;*

Distributed Compilation which divides the work to be run parallel and thus increases the performance by significantly reducing the compilation time. In order to know more about how the project works and what the user must do to avail the service, he can go through the interactive video tutorials and sign up to be a member [1].

Distributed Compiling is a special type of compiling activity in which we compile the source program using several machines with the same or cross compilers to do the compilation in parallel and hence speed up the entire process of compilation. Here the source code of the program need not to be parallel, but the compilation of the program is done over several machines in parallel, where the source code is divided into several parts and each machine is provided its part and they compile it together. Distributed Compilation is similar to the normal compilation with the difference that it compiles the normal program in parallel over various systems [2].

A server farm like the one we will implement can be an asset to the organizations which maintains a code base of the order of a million lines of C/C++ code. In most of the cases whenever some changes are introduced in the code, either as new functionality or as bug removals of original system the entire compilation of the system is compulsory, so that their remains consistency in the source code among its various distributions, it is mostly done in the Open Souce projects like Linux, Calligra etc. But now a day's even in the Commercial projects this thing is being employed. Reduction in build times can prove to be an effective productivity enhancer. The fact that we don't need dedicated machines running the distcc daemons means that such a system can be implemented with minimal resources. For instance, a Linux Kernel that could have taken somewhere around 40-45 minutes to an hour on a dual-core system actually took hardly 20-25 minutes on two systems that were each loaded with quad-core processors [3].

It is usually preferred that the fastest/closest/least loaded machines are put at the start of the DISTCC_HOSTS list. We have to write the list of the available nodes in our priority order in the "/etc/hosts" file in each of the nodes so that each node can connect with each other but this list is changeable based upon the fact that a machine may be down for some time or any software or hardware failure. This is mostly significant when executing "./configure" scripts because all compilation will be done on the first system listed in a compilation table nodes. Usually, it would be homehost, however if another machine is much quicker then possibly not, to some extent this is still an open bug that is to be addressed further. It would be great if distcc might automatically detect the best distribution process for load distrbution, but it doesn't perform that so far. But in our system we have added a new php script which automatically does this work. Whenever it sees that a system is overloaded or its utilisation increases above a threshold value (in our case it is 65%), it automatically removes that node from its available list and as soon as its utilisation decreases the particular system is added back into the available list of helper nodes.

2 Difference between Reducing the Complexity and Reducing the Compiling Time

It is worth noticing that the performance gain is obtained because distcc compiles the file using more than two processors by dividing the work to be compiled into chunks and compiling these chunks in individual processors available for the compilation to the server, more than one instruction in a single time unit. An algorithm that is O (n^2) will still run 10^4 slower if the input size if increased by a factor of 100.However, the program will be compiled much faster because of

the fact that it is using many processors for the compilation process. As expected, the factors that prevent us from getting the ideal speed up are:

1. Communication Delays.
2. Typical symptoms include linker errors, weird syntax errors or compiler crashes, caused by corrupt object or source files can slow down overall compilation process.

3 Infrastructure As a Service

Infrastructure can be simply defined as the resources of any system. In terms of computer the infrastructure is simply the CPU processing, memory space both RAM and Hard Disk. Every computer system has some infrastructure in it, but in larger projects and in special cases, we may require additional infrastructure for the processing of our work then we may require some other infrastructure. This can be provided to a user in the form of Infrastructure as a Service.

IAAS, Infrastructure As A Service, is a kind of service provided by cloud computing, can supply users with the convenience and flexibility of virtual resources to achieve the same power supply as the water supply [4]. Service-oriented IAAS will extend service lifecycle management to the field of fine-grained virtual resources, taking full advantage of the virtual reconstruction of the concept of service-oriented resource management methods and strategies for the service platform to provide more flexible physical resources to support and improve the quality of service.

An infrastructure has been designed which can help the clients to use the service though a webpage and an Internet connection. The front end gives users to make a secure connection to the servers that will provide cores for the distributed compilation [5].

The website allows users to make an account which after a successful login redirects him to a dashboard (home page). This dashboard lets user to check the current usage of the nodes, generate a script that can be copied to the client's terminal to connect to the helper nodes at the server side.

export DISTCC_HOSTS = 'IP Address of all the Helper Nodes';

The Usage gives the user a better insight of the load that is being held at the server side, which will allow users to make a decision about the number of nodes he should use to assign his work to. If the usage comes out to be very high, then he can delay the assignment of the compilation work to the nodes to disallow any system failure.

The red bars in the graph, as indicate in figure 1, represent the excessive use of the cores giving an indication that the system is busy doing some task [1].

For the user to ask for nodes to compile his work, he needs to click on Generate button that allows him to select the number of nodes he needs assign his work. More the number of nodes will result in faster compilation but hindering other people to us the nodes for the amount of time taken for the compilation. For this paper we have implemented a cluster of four helper nodes and hence a drop down list with choices of 1-4 was available to the clients. Each node has 4 cores and all the four cores are assigned to the user while the node is selected to do the job [6,7] as shown in the Fig. 2.

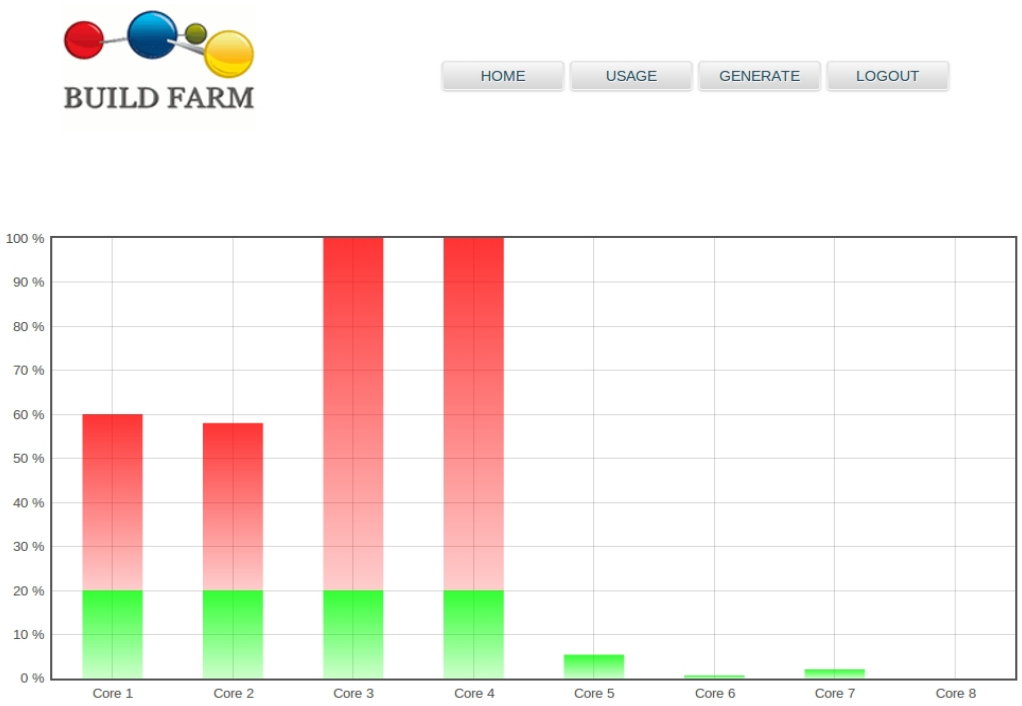

Fig. 1. Usage Graph for all cores of two Helper Nodes with one Node Busy

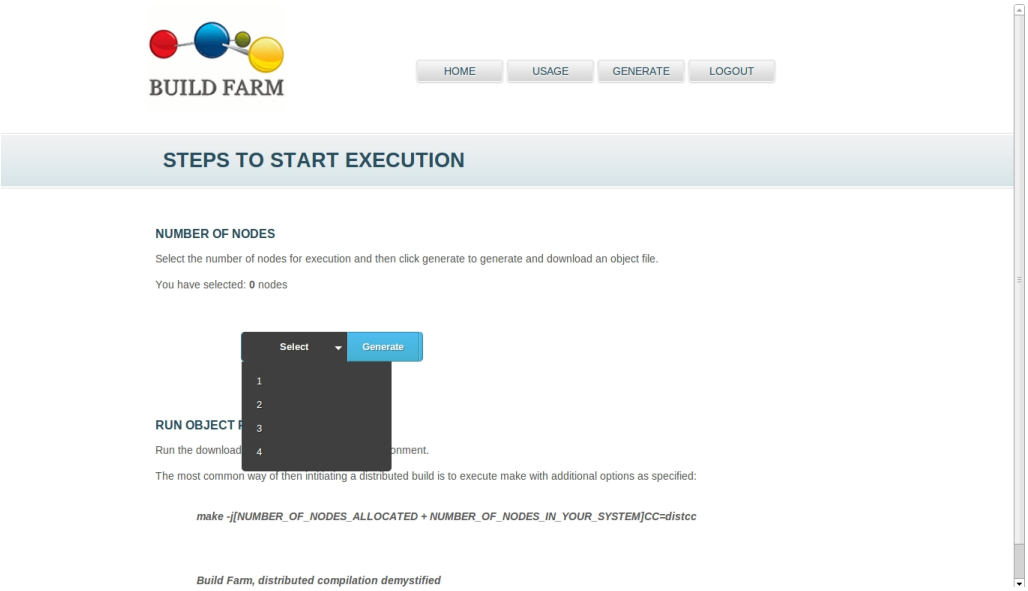

Fig. 2. Generate Page when the user hovers over drop-down menu

If the number of requested nodes is more than the nodes available for job compilation the server automatically gives an alert with available nodes and asks if user wants to continue. In the screenshot, as given in Fig. 3, the user requested for 2 nodes but only 1 was available so an alert for the same was generated.

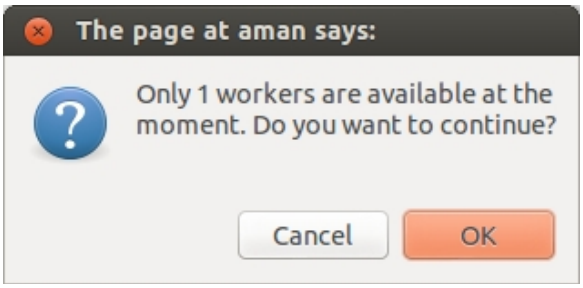

Fig. 3. Alert Message Generated when number of nodes requested by user is unavailable

4 Design and Implementation of Usage Collector

We begin with high level overview of the system into which this component fits in. We are making a system that will allow a user to compile his program on several idle systems. The user (client) requests an allocation server to allocate some idle machines for this task from the cluster. The task performed by the allocation server can be stated as follows:

When the client logs in, check if there are free helpers that can be allocated to the client. If so, allocate them. If not, inform the client about the maximum number of systems that can be allocated.

The usage collector component helps allocation server in deciding which helpers are free (and thus can be allocated) and which ones are busy. It maintains a Table called "Available" that has 2 columns, IP address and CPU usage. The helpers that have CPU usage below a threshold are added to the available table.

As the aforesaid table is maintained and *updated* by our usage collector, all that allocation server has to do is to read IP addresses that correspond to the minimum usage and display them to the client.

The details of the design and implementation follow as shown in Fig. 4. The description also answers questions like why continuous updating is required and how noise in the usage data is handled [8].

- Calculating CPU usage
- Design of the data collector
- Handling Noise

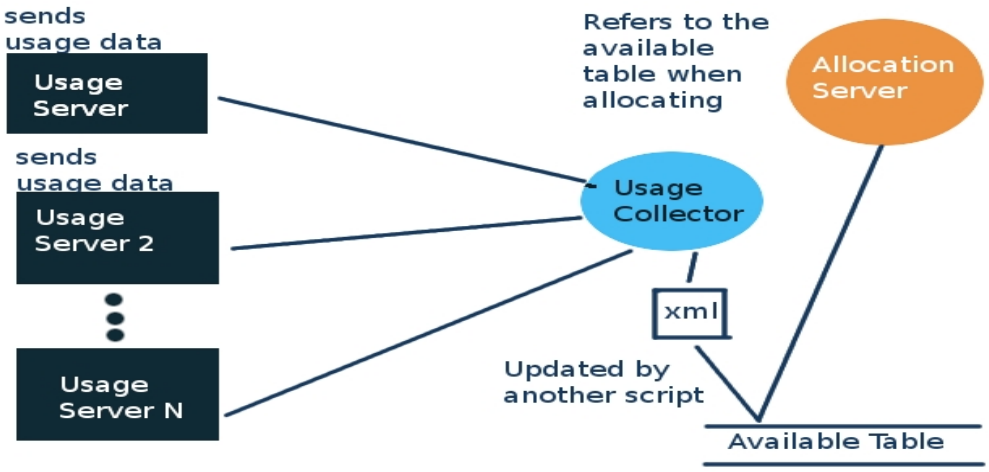

Fig. 4. Design of the usage collector model

Note that the work of collector stops after updating the usage.xml. The available table is updated by another script after averaging the usage to remove noise.

The design is quite simple and is based on the UNIX philosophy of doing one thing well. The component is tasked with generation of the usage.xml file only. The same concept is employed on whole of the system

4.1 Obtaining CPU Usage

In Linux, the CPU usage can be obtained by reading the file /proc/stat. A typical output of the stat file looks like:

cpu 476208 1727 99018 7362330 62460 3 3431 0 0 0.

The man page tells that the numbers represent the amount of time, measured in units of USER_HZ (1/100ths of a second on most architectures), that the system spent in user mode, user mode with low priority (nice), system mode, and the idle task, respectively.

To get from these numbers to the actual usage, we need to go through one more step. The usage numbers here are cumulative, that is, they represent the total time spent in various modes since the system started. To get the actual usage in the past T time units, we use the following approach:

Let t0 be the time at which the system was started.
Let U1 be the usage at time t0 + T1
Let U2 be the usage at time t0 + T1 + T

Average CPU usage in the past T time units = (U2 – U1) / T

To accomplish this, we sample the /proc/stat file at regular intervals, subtract current value from "backup" values and divide by the sampling interval. The backup values are then updated.

As an example, here is an instance of /proc/stat (first line only):
CPU 180865 662 46290 4534456 24346 3 1370 0 0 0
Here are the values 10 seconds later :
CPU 181198 662 46440 4546287 24383 3 1379 0 0 0
CPU Usage = (228300 – 227817) / 10 = 483 / 10 = 48.3 %

4.2 Sending the CPU Usage

The usage server calculates the CPU usage after regular intervals using the method specified above. The usage is then sent to the usage client running on the allocate server in the following format:

Core1usage; Core2usage;...;CoreNUsage

4.3 Compiling Usage from Each Node

It is the responsibility of the usage collector node to collect usage data from the all the usage servers running on each helper and finally compiling it in one document. The usage is compiled into an xml that is used by other components of the system as described later.

4.4 Dealing with Noise: Usage Spikes

Sometimes the CPU may get loaded for a split second due to a user action. This may cause spikes in the usage data. The spikes are troublesome because they tend to indicate that the helper is busy, when it really is not.

We handle this problem by not relying just on the immediate usage but by referring to usage 2 seconds and 4 seconds back. A script runs continuously that regularly creates back up of the usage.xml file by copying it in usage1.xml (usage 2 seconds back) and then in usage2.xml (usage 4 seconds back). While deciding whether a node is busy or not, usage from all the 3 files is averaged.

5 System Design

The design of the system is a bit complex as there are various different processes that are concurrently running and are all inter-dependent. The chapter thus helps in individually explaining each of these processes and give a brief understanding of the system design and the background jobs that make the system run efficiently.

5.1 Usage Collection of Helper Nodes

The system is required to keep a track of the current usage of all the nodes as well as the server to know that which nodes are occupied and the level of usage or processes that are running on the cores of every node.

A central system works on this problem by requesting all the connected nodes every 2 seconds to send their current CPU usage as given in Fig. 5. The helper nodes then reply with their respective CPU Usage and central system then calls a script called "Collector Script" that compiles all the results and generates an XML file known as "usage.xml".

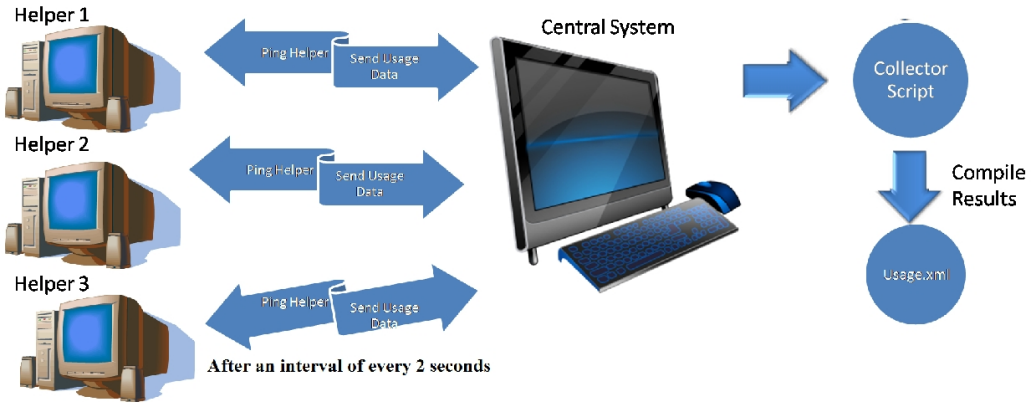

Fig. 5. Usage collection of all the nodes

5.2 Back – Up Creation of Usage.Xml File

The system is required to keep a back-up of the old usage of all the nodes so as to calculate the average of the usage and remove any usage spikes that might cause errors in judging the actually free nodes [8].

Thus, "UsageDataArchiver.sh" is the script that copies at every instant:

- All the values from the usage1.xml to usage2.xml (4 seconds old values)
- All the values from the usage.xml to usage1.xml (2 seconds old values)

Updates usage.xml with current values.

5.3 Update Available Table

The system maintains an available table that stores the IP addresses of all the nodes that are available and can be used for assigning a compilation job. The table is updated by a "UpdateAvailableTable.php" script that continuously runs and take the usage values of all the nodes from the three files: usage.xml, usage1.xml and usage2.xml and then averages the value as mentioned in Figs. 6 and 7.

The average value is then checked and the following three steps are taken:

- If node is already in table and its usage is more than 65%, it is removed otherwise it is left as an entry.
- If a node that was earlier not available due to high usage but now has usage <65%, is added in a new row in the table.

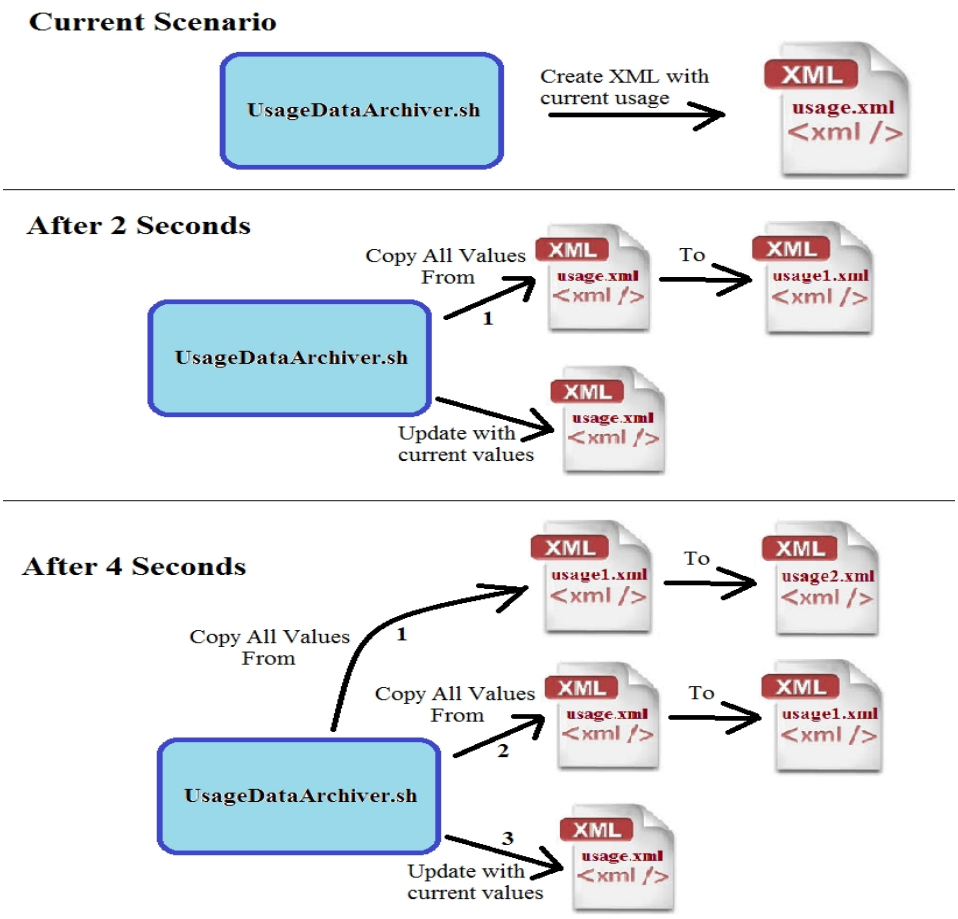

Fig. 6. Back-up creation of usage.xml files

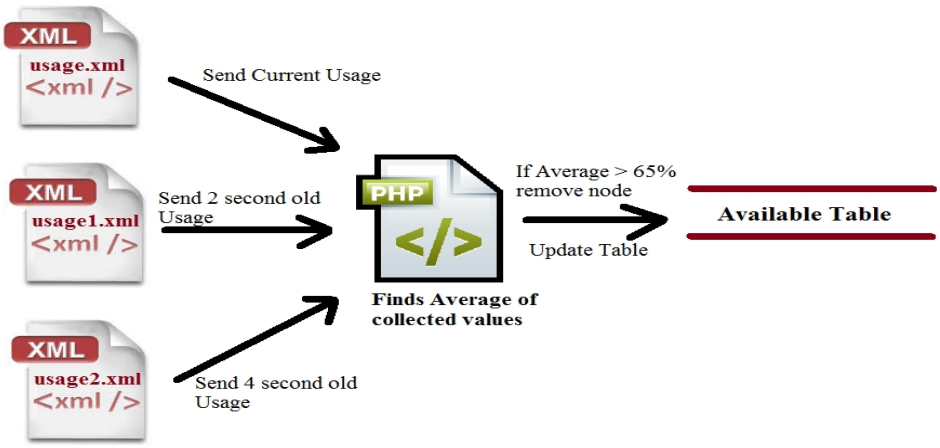

Fig. 7. Updating available table

5.4 Client – Server Interaction for Getting Helper Nodes

The process of obtaining the nodes for compilation job also has several steps involved in it. The client requests the allocation server for N number of nodes that needs to be less than or a maximum of 4; the server then checks with the available table for the required number of helpers [9,10,11,12].

The server then returns with either of the two:

- The IP addresses of N nodes that may be used for compiling a job by the user.
- An alert message with the available nodes that is less than the demanded nodes and if the user accepts to continue the IP address of those are provided. If the user declines then he/she may wait for the required nodes to be freed.

The complete process of client- server interaction is highlighted in Fig. 8.

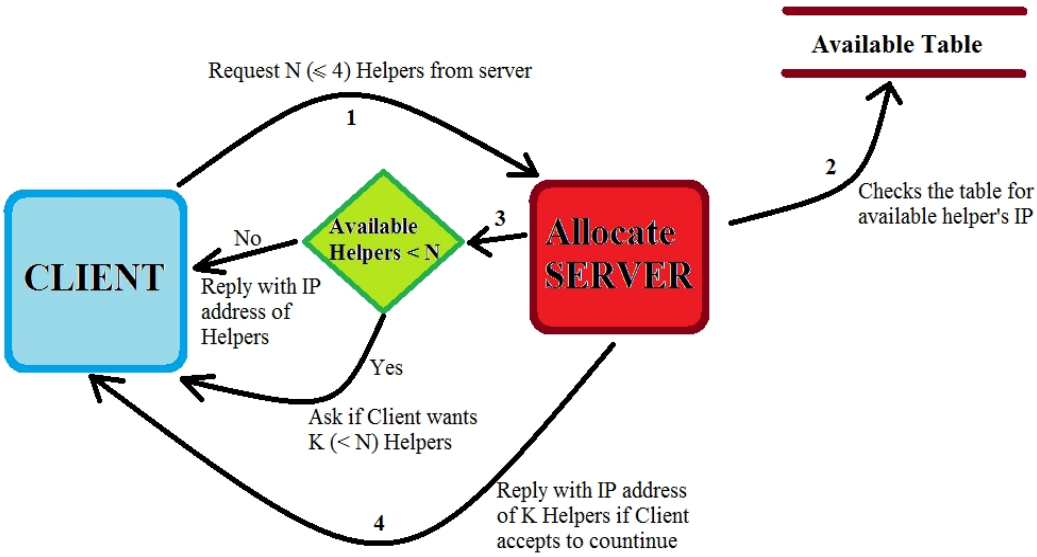

Fig. 8. Client – server interaction for getting helper node

5.5 Compilation Results for a Linux Kernel

The system was used to compile Linux – 3.8.5 kernel for various numbers of nodes and in different situations. The differences in the results show how much gain can be achieved for different setups. The three systems used for the compilation jobs were: Localhost: Quad Core i3 Processor with 2.0 GHz Clock Speed

192.168.1.30: Quad Core i5 Processor with 2.4 GHz Clock Speed
192.168.1.40: Quad Core i5 Processor with 2.3 GHz Clock Speed

The time was recorded using the 'time' command in linux that gives duration of execution of a particular command. The command provides user CPU time, system CPU time and real time. The result for various situations is tabulated in the following Table 1.

Table 1. Various readings of time for compilation of various programs

Package name	Wall time (single node)	Wall time (distributed)	CPU time	CPU time (distributed)	Decrease in CPU Time
Binutils-2.18	125.8s	35.4s	102.9s	25.6s	75.16%
Glibc-2.6	946.4s	534.0s	568.7s	390.5s	31.33%
Httpd-2.0.43	139.8s	85.1s	117.2s	51.3s	56.22%
Linux-2.6.25	818.3s	203.2s	543.9s	185.3s	66.04%

6. Seti @ Home

SETI@home is a project where our project finds its relevance and the following chapter gives a brief explaination about the same. SETI@home works on the idea that millions of computer owners worldwide can contribute their computer's processing power to the search of extraterrestrial intelligence by performing the largest computation ever [13,14,15]. This can be achieved by using computers in homes and offices around the world to analyze radio signals from space. The approach might be complicated but delivers unprecedented computing power.

6.1 About Seti

SETI (Search for Extra-terrestrial Intelligence) is focused to detect intelligent life outside Earth. One approach uses radio telescopes to listen for narrow-bandwidth radio signals from space which are not known to occur naturally. So detection of these signals would provide evidence of extra-terrestrial technology. Radio SETI projects digitally analyse the data by computing its time-varying power spectrum, then finding candidate signals through pattern recognition on the power spectra and finally, eliminating those signals that are probably natural or man-made.

Before SETI@home, radio SETI projects used special-purpose supercomputers located at the telescope to do the bulk of its data analysis. In 1995, an idea to use a virtual supercomputer consisting of large numbers of Internet-connected computers was proposed which established the viability of public-resource computing in which computing resources are provided by the general public. But for many tasks, huge computing power implies huge network bandwidth, which is typically expensive or limited [16,17].

6.2 Interesting Facts about Seti

The whole client program of SETI@home is written in C++ which comprised of a platform-independent structure for distributed computing. This size of program is 6,423 lines of code and their components with platform-specific implementations (such as the graphics library with 2,058 lines of code). In the same way, specific data analysis code of SETI is 6,572 lines as well SETI-specific graphics code is 2,247 lines [18,19].

The client has been ported to more than 170 various platforms. These tasks are make possible with the help of GNU tool together with gcc and autoconf. All client can be executed as a background process as either a GUI application or as a screensaver. The whole system occupies with an structure in which one thread handles data processing and communication, the second thread handles GUI interactions, and the third thread (possibly in a split address space) renders graphics based shared memory data structure to maintain different modes of various platforms [20,21].

As per the facts provided SETI@Home was open to public in 1998 and by July 2001 the participants had processed over 221 million work units with an average throughput of 27.36 Tflops and performed 1.87 X 10^{21} floating point operations.

6.3 Conclusions Drawn from the Study

Public-resource computing relies on personal computers with excess capacity, including idle CPU time. The following conclusions were made about Public – resource computing that:

The role of distributed compilation is more prominent and efficient when Computation to Data Size Ratio is high i.e. the size of data on which the computations are to be performed is comparatively small. If we take SETI@home as example then each data unit of SETI@home takes 3.9 trillion floating- point operations, or about 10 hours on a 500MHz Pentium II, yet involves only a 350KB download and 1KB upload. This ratio keeps server network traffic at a convenient level while imposing minimal load on client networks [28]. Some applications, such as computer graphics rendering, use large amounts of data per unit computation, possibly making them inappropriate for public-resource calculation. though, reductions in bandwidth expenses dispel these troubles, and multicast techniques decrease costs when a large part of data is constant across work units.

- Applications those are comprised with independent parallelism are easier to handle as compare to those with many data dependencies. The computations work-unit of SETI@home is independent; hence participating computers never wait or communicate with one another. If a computer be unsuccessful while processing a work unit then the work unit is ultimately sent to another computer.
- Applications which required frequent synchronization and communication among others node have been parallelized using such hardware-based approaches like shared-memory multiprocessors and more newly software-based cluster computing, like as Parallel Virtual Machine software.

Based on these concerned points, public-resource computing with its frequent computer outages and network disconnections seems badl-suited to such applications. Though, scheduling mechanisms which find and develop groups of LAN-connected machines may remove these problems [22]. This paper work may be used in research works like SETI by enhancing the computation requirement at various phases and at the same time, using a LAN is for high speed data transfers.

Tasks that tolerate errors are more agreeable to public-resource computing; like if a SETI@home work unit is analyzed incorrectly or not completely, the overall aim is affected only slightly [23]. The success of public-resource computing concepts has the auxiliary benefit of increasing public awareness of science and democratizing, to an extent, the allocation of research resources.

7. Distributed C Compiler

Distributed C Compiler is a program that distributes compilation of C, C++, Objective C and Objective C++ codes among several machines on a network, and generates the same results as that of a local build. It is avaiable under the GNU General Public License. Distcc has been under development since early 2002 [24,25]. Distcc is developed on GNU/Linux, but has been reported to work efficiently on other systems as well. Distcc is not a compiler itself; rather it is a front-end to the GNU C/C++ compiler (gcc) [26]. There is preliminary support for some other compilers though. Almost all gcc options and features work as normal. Distcc sends the complete pre-processed source code across the network for each job and all it requires of the volunteer machines is that they run the distccd daemon, and have an appropriate compiler installed.

Distcc is designed to be used with the -j parallel-build feature in GNU Make or other build tools. Sending 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. Programs known to have been built correctly with distcc include the Linux Kernel, KDE, GNOME, Samba and Ethereal [27].

7.1 History and Shortcomings of Previous Distributed Build Systems

Several distributed build systems have been developed before distcc came into existence, like pvmmake, doozer and dmake [28], which have following short comes.

- All these systems required the build directory to be a networked file system and shared by all the machines that are being used to build particular software.
- These systems also required the machines to have the same header files and libraries installed, which if inconsistent may lead to error messages and crashes in worst cases.
- The systems developed earlier even required the clocks to be tightly synchronized so as to create an accurate timestamp.
- These systems were particularly designed for dedicated build clusters. This became a handicap for users in an ad-hoc network who may want the liberty to upgrade libraries at their own will and use a different network file system (NFS).
- A different approach that was also used previously was to use kernel – level clustering and have very tight machine coupling which was achieved by distributing parallel tasks. The foresaid approach was quite ineffective with respect to performance as there may be many short – lived processes that might be generated during compilation process.

7.2 Design of Distcc

Distcc is designed to follow the principle of “Worse is Better” by Richard Gabriel, that can be summarized and explained in the following points:

- Simplicity–*The design must be simple both in implementation and interface. It is important for implementation to be simpler than interface as it affects the design.*
- Correctness–*The design must be correct but at the same time it should be more simple than correct.*

- Consistency–*The design must be as consistent as possible and simple but it is preferred to drop those portions that are less common and introduce complexity or inconsistency.*
- Completeness–*The design must cover all the possible and reasonably expected situations. Completeness can sometimes be sacrificed for quality and should be sacrificed if simplicity is at stake.*

Distcc is not a general-purpose clustering system as they are too hard to write, and none that can be installed in a matter of minutes. Distcc is incomplete in coverage of all the tasks a user might want to distribute across machines, but it does handle a large task that is important to a significant number of users. Most tasks can be distributed that fall under the category of C compilation. There is always an overall benefit even if a few jobs during a software build cannot be distributed, as others that can be run in parallel are distributed. Distcc's documentation and diagnostic output make it easy to identify problem commands. Some gcc options that might be distributed in practice are run locally for the ease of implementation complexity. Commands that produce assembly listings are no reason why the assembly output could not be relocated back to the client except this option is hardly ever used because complexity is not defended [29].

7.3 Working of Distcc

Distcc distributes work from a client machine to any number of volunteer machines. (The term volunteer is used to refer the slaves that are used by other systems.) Distcc works with a client program and a server.

The responsibility of the client is as follows:

- Analyses the command run,
- For the jobs that can be distributed choose a host,
- Run the pre-processor,
- Send the request across the network, and
- Finally, report the results.

The server on the other hand accepts and handles requests containing command lines and source code and responds with object code or error messages

7.4 Advantages of Distcc

The advantages of Distcc are as follows;

- Easy to set up
- Builds software projects several times faster than local compilation
- Does not require all machines to share a file system, have synchronized clocks, or have the same libraries or header files installed. They can even have different processors or operating systems, if cross-compilers are installed [30].

8. Conclusion

The experiments we did demonstrated that the gain that we expect of such a sytem is actually attainable and is not only intuitive. The system can also be used as a reference for those who want

a way to control a system of computers working together as well as in reconfigurable computing system [31]. The components that help the users share their usage with a central server, the collector that compiles the usages into an xml, and the php script that reads the xml and updates the available table; together make a component that can be employed as it is on similar problems owing to the modular design.

Distributed compilation is one of the techniques that helps the organizations in utilizing this concept of distributing the workload among several helpers to reduce the build time. The system we designed would help organizations in putting utilizing the existing computing power, most of which is anyways lying idle, for use by those in the organization who need it.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Klein A, Mannweiler C, Schneider J, Schotten HD. Access schemes for mobile cloud computing, in Proc. of IEEE Mobile Data Management (MDM). 2010;387–392.

[2] Aman Madaan, Sunil Kr. Singh, Ankur Aggarwal. Junk Computing: Performance Evaluation and Comparison of an MPI based Heterogeneous Cluster. IJARCSSE. 2013;3(4):124-131.

[3] Petcu D. Portability and interoperability between Clouds: challenges and case study. In: Proc of LNCS Springer Service Wave 2011;6994:62–74.

[4] Sushil Bhardwaj, Leena Jain, Sandeep Jain. Cloud Computing: A study of infrastructure as a service (IAAS), IJEIT 2010.

[5] Tien Van Doa, Csaba Rotter. Comparison of scheduling schemes for on-demand IAAS requests. February 16, 2012.

[6] LUO Xiao-xiang, SONG Mei-na, SONG Jun-de, "Research on service-oriented policy-driven IAAS management",September 2011, 18(Suppl. 1).

[7] Rahul Ghosh, Francesco Longo, Vijay K. Naik, Kishor S. Trivedi. Modeling and performance analysis of large scale IAAS Clouds, 27 June 2012.

[8] Report on Java Script Library for Graph making. Available: http://www.flotcharts.org/

[9] Armbrust M, et al. Above the clouds: a Berkeley view of cloud computing, Tech. Rep. UCb/eeCs-2009-28, EECs department, U.C. berkeley, Feb 2009.

[10] Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D. The eucalyptus open-source cloud-computing system, in: Cloud Computing and Applications (CCA08), 2008.

[11] Petcu D, Craciun C, Neagul M, Lazcanotegui I, Rak M. Building an interoperability API for sky computing, in: Proc of IEEE Inter Cloud. 2011;405–412.

[12] Foster I, Kesselman C, Lee C, et al. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation", in Proc. of Quality of Service, IEEE Press, London, UK. 1999;27-36.

[13] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer, titled "SETI@home An Experiment in Public-Resource Computing", November 2002.

[14] Carolan J, Gaede S, Baty J, Brunette G, Licht A, Remmell J, Tucker L, Weise J. Introduction to cloud computing architecture—white paper, 2009.

[15] Foster I, Kesselman C, Lee C, Lindell R, Nahrstedt K, Roy A. A distributed resource management architecture that supports advance reservations and co-allocation", in: Proceedings of the International Workshop on Quality of Service. 1999;1–9.

[16] Hill Z, Humphrey M. Csal: a Cloud storage abstraction layer to enable portable Cloud applications, in Proc. of IEEE Cloud Com-2010. 2010;504–511.

[17] Dean J, Ghemawat S. Map Reduce: simplified data processing on large clusters", Communications of the ACM. 2008;51:107–113.

[18] Wang L, Laszewski G, Kunze M, Tao J. Cloud computing: a perspective study. J New Generation Computing. 2010;1-11.

[19] Mell P, Grance T. Cloud Computing Definition", National Institute of Standards and Technology, Version 15.

[20] Shoch J, Hupp J. The Worm programs: Early experience with a distributed computation". Commun. ACM. 1982;25(3):172–180.

[21] Chang F, Ren J, Viswanathan R. Optimal resource allocation in clouds", IEEE 3rd International Conference on Cloud Computing. 2010;1–8.

[22] Jes Hall. Distributed Compiling with distcc. October 05, 2007.

[23] McCalpin J. Memory bandwidth and machine balance in current high performance computers. IEEE Technical Committee on Computer Architecture Newsletter. 1995;19–25.

[24] Report of distcc: a fast, free distributed C/C++ compiler: Accessed 29 January 2013. Available: https://code.google.com/p/distcc/

[25] Martin Pool. Distcc - a fast free distributed compiler, December 2003.

[26] Sotomayor B, Keahey K, Foster I. Combining Batch Execution and Leasing using Virtual Machines. Proc. 17th Int'l Symp. High Performance Distributed Computing (HPDC-08), ACM Press. 2008;87-96.

[27] Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kauffman, San Francisco; 1999.

[28] Sunderam V. PVM: A framework for parallel distributed programming. Concurrency: Pract. Exper. 1990;2(4):15–339.

[29] van Ommeren E, Duivestein S, de Vadoss J, Reijnen C, Gunvaldson E. Col- laboration in the Cloud-How Cross-Boundary Collaboration is Transforming Business. Uitgeverij kleine Uil, 2009.

[30] Vıctor L. Orozco, Jose F. Lobos. Distributed compilation with Distcc and Icecream clusters using Gentoo, a benchmarking study case.

[31] Sunil Kr Singh, Singh RK, Bhatia MPS. System Level Architectural Synthesis and Compilation Technique in Reconfigurable Computing System, in the proc of Embedded System and Application (ESA-2010). 2010;109-115.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
www.sciencedomain.org/review-history.php?iid=615&id=6&aid=5542