Design and Implementation of A High Performance Computing System Using Distributed Compilation

Dr. Sunil Kr Singh
Computer Science
BVCOE, New Delhi
New Delhi, India

Aman Madaan
Computer Science
BVCOE, New Delhi
New Delhi, India

Ankur Aggarwal
Computer Science
BVCOE, New Delhi
New Delhi, India

Ankur Dewan
Computer Science
BVCOE, New Delhi
New Delhi, India

Abstract— **The idea of using the idle resources of a system for some other useful purpose is not new. seti@home pioneered this concept of "public resource computing" by bringing together millions of users worldwide who were ready to donate their idle CPU cycles to the cause of searching the extra terrestrial intelligence. In this paper, we describe a system that uses the same concept for reducing build times by using free cycles on idle computer systems in computer labs of our institute. The challenge of distributing compilation is tackled by a distributing compiler. We use distcc for the purpose. The challenge then is to design a system that keeps track of free helpers, dividing the work fairly among the helpers and most importantly, providing an intuitive interface to the clients who would use the system oblivious of the complexities of the back end.**

Keywords - virtualization, cloud computing, resource monitoring, service-oriented

I. INTRODUCTION

A typical lab session in colleges spans approximately 2 hours and has interleaved periods of activity and total idleness. In typical programming labs, the CPU utilization even during the periods of activities is not much because of the nature of work. Since the systems are up and running all the time, it will be valuable to employ the collective power of several such systems for a allotted task, rather than employing a devoted computing beasts, leading to considerable savings in terms of power and costs.

Fig1 shows the CPU usage of a machine over 2 hours of time while the user surfed internet, listened to music and used a word processor. The CPU usage was read every 1.5 seconds thus generating close to 5000 samples. As can be seen, the usage mostly stays below 10%. The mean usage is 8.008%. This motivates us to utilize them for alternate purposes.

The task for which we would be using our helpers is compilation. For large code bases, the compilation times can vary from several minutes to several hours. Software systems that employ a continuous integration engines like Jenkins, which compiles the codebase every time there is a change to prevent big bang integration (i.e. to ensure that changes in one component have not hampered working of another) can gain a lot from tools like distributed compilers, which help in reducing the build times.

Icecream and distcc are the two most popular distributed compilation tools; we use distcc as the distributed compiler for our build farm. Some of the benchmarking results from the distcc website are as compiled in table 1. The single node is an Intel Pentium 4 machine. "Distributed" refers to a cluster of 152 such machines. The table clearly brings about the impressive reduction in the compilation time that such a system can offer.

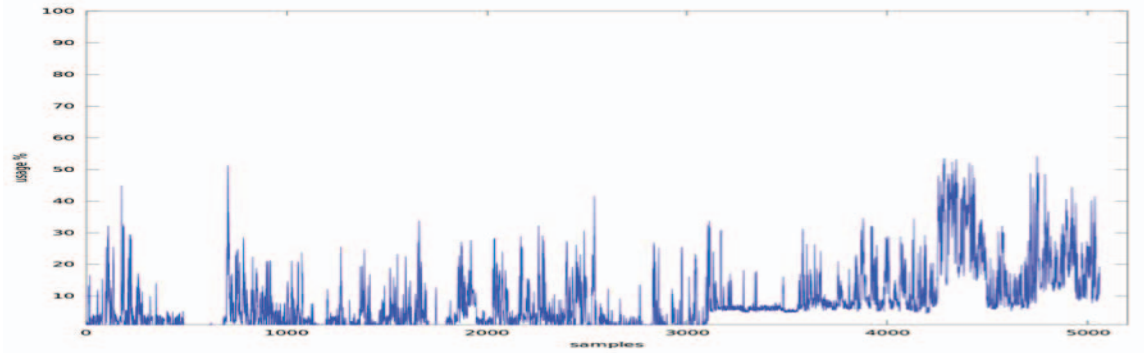

Fig.1. Percentage CPU Usage of a machine over 2 hours

TABLE 1: Reduction in build times of various software suites by using distcc

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%

The distcc, the distributed compiler, has to be informed about the helpers, the nodes over which the compilation can be distributed, before the process starts. An end user taking help of our build farm cannot be expected to know which nodes are idle and which are free. We also need a mechanism to ensure that a node is not overloaded with jobs.
Finally, we should make the process of requesting extra nodes as hassle free as possible. In the subsequent session, we discuss the design of our build farm manager and some of the challenges faced there.

II. SYSTEM DESIGN

A. Entities

We first define the various entities that form the system.

1) Helpers: The computer systems that donate their free cycles for use by distcc. Each helper runs a usage server that sends it's usage to whoever needs it.

2) Allocation server: The central authority that is responsible for the following three activities:

a) Usage Monitoring: Keeps track of the CPU usage of the helpers.

b) Allocation: Decides which helpers are free and allocates them.

c) Providing interface to the service: Provides a web interface to the user who wants to use services of the build farm

3) The build farm: The network of helpers and the allocation server along with the controlling mechanisms constitute the build farm.

B. Collecting CPU usage

The first challenge was to propose a method that would allow the allocation server to keep a track of CPU usage for each of the helping systems.

The solution is modeled on the classical client server paradigm. Each machine runs a "usage server" that sends the CPU usage data of the machine to the "usage client" running on the allocation server. The CPU usage is obtained by using a simple mechanism detailed in the subsequent sections.

The Fig 2 clarifies the situation.

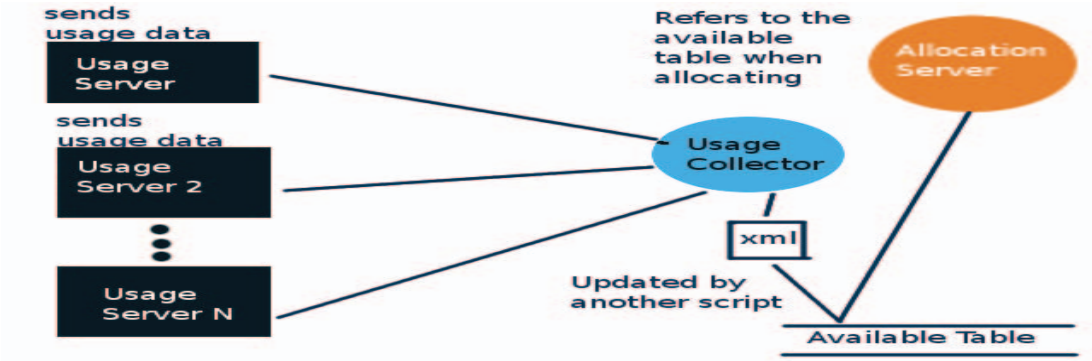

Fig.2. Design of the usage collector model

1) Obtaining the CPU usage

The cpu usage over a time span T is simply the fraction of T the cpu has spent on executing the instructions. Rest of the time is wasted waiting for other actions (like input/output) to complete.

In Linux, the CPU usage can be obtained by reading the file "/proc/stat". The /proc/stat file hosts a variety of data. The details can be obtained by using the man page of proc[13]. Of interest to us is the first line of the file, which has the numbers that can be used to calculate the combined CPU usage. A typical first line 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 "previous" usage values and divide by the sampling interval. The "previous" 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 %

The following pseudocode conveys the idea presented in the above paragraph

```
Prev_Values = 0
While(1) {
        Current_Values = read('/proc/stat')
        Usage = (Current_Values - Previous_Values) /
SAMPLE_INTERVAL
        wait for SAMPLE_INTERVAL
}
```

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

The allocation node uses the average of per core usage for making decisions. Sending detailed usage helps in plotting detailed usage plots

3) Compiling usage from each node

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

```
<xml version = '1.0' >
<usagedata>
<info>
<servernumber>127.127.0.1</servernumber>
<core>10.2041</core>
<core>4.10959</core>
<core>1.33333</core>
<core>0.6711dd</core>
</info>
</usagedata>
</xml>
```

III. SYSTEM AS A SERVICE

One of the most important aspects of this project is to provide an interface that makes requesting a node from the build farm simple. The complexities of the back-end should be hidden from the user. A good analogy is a customer visiting a shop with free soda vending machine. Setting up of the soda machine and the ingredients is done by the shop owner. The "user" can simply walk to the shop and order soda. He may have to wait for a glass if there is a queue. Since the soda is given for free, the

shopkeeper must ensure that each user is given a fixed amount of soda so that for a given amount of ingredients, maximum number of customers can be served.

We provide a web-based interface, just like the soda machine, to the users of our build farm. This ensures that the user is connected to the network and can access the system using nothing more than a browser. The intuitive interface makes requesting the additional resources a matter of clicks.

A. Account Creation

A user interested in using the build farm can start by creating an account on the website. Once the user registers, a request is sent to every system administrator to add the IP address of the user to the list of allowed IP addresses. This process can be automated. Once the process is complete, the login details are mailed to the user.

The user can login to the site using the details. A successful login redirects him/her to a dashboard (home page). The dashboard consists of several tabs as discussed below.

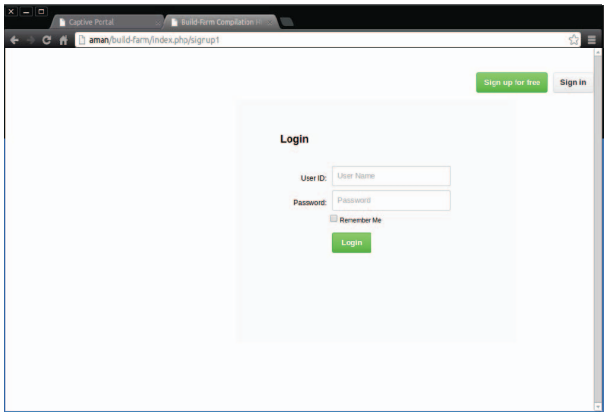

Fig.3. Login Page for using the Build Farm

B. Usage Tab

This tab lets user to check the current usage of the nodes in the farm. The usage gives the user a better insight of the load at the server side, and thus helps him in deciding whether to request resources from the farm and if so, how many nodes should be requested. If the usage is very high, then he can delay his request to the system. The usage is depicted by a bar for each core of each node. Cognition techniques are used like coloring the bar red when the system is highly loaded and updating the usage regularly helps the user in perceiving the situation easily.

C. Generate Tab

The generate tab provides the user options to select the number of helpers to use and then generate the IP addresses of the nodes. 2 cores from every node are allocated. (Figure 4)

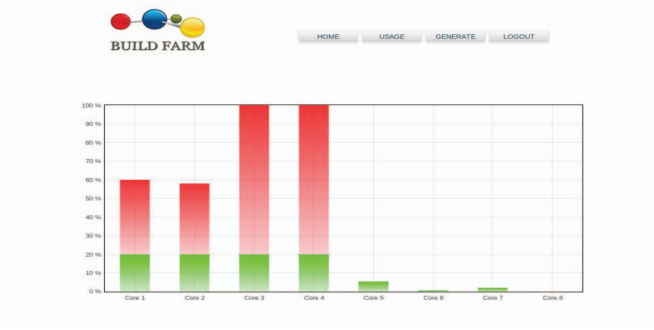

Fig.4. Usage Graph for all cores of two Helper Nodes with one Node Busy

Once the number of helpers is decided, the user needs to click on Generate button that allows him to select the number of nodes required. The number of nodes and the compilation time are in inverse proportion, but there exists a limit after which adding more nodes are not of any use. 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.

Fig.5. Generate Page when the user hovers over drop-down menu

If the number of requested nodes is more than the nodes available at the moment, the allocation server generates an alert with available nodes and asks if user wants to continue. In the screenshot the user requested for 2 nodes but only 1 was available so an alert for the same was generated.

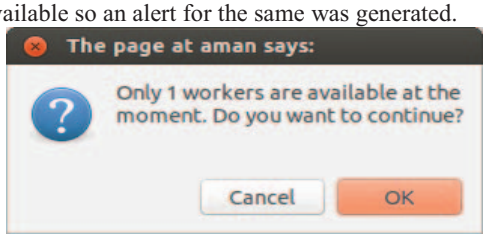

Fig.6. Alert Message Generated when number of nodes requested by user is unavailable

Finally, after the nodes have been allocated to the process, the allocation server dumps the command that has to be executed by the user in order to distribute the compilation.

The command looks like:
export DISTCC_HOSTS = 'IP Address of all the Helper Nodes';

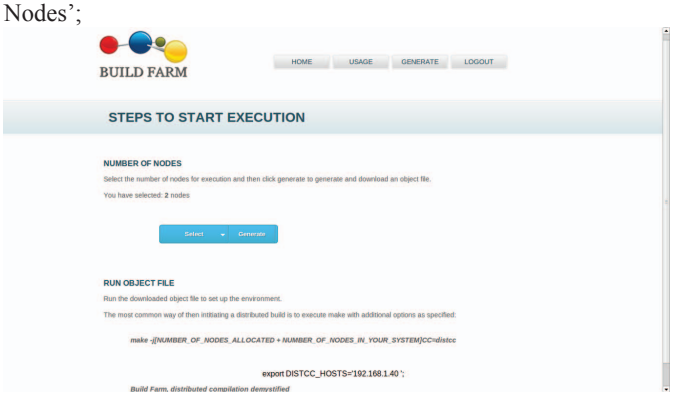

Fig.7. Generate Page that gives the user export command with IP of helper nodes

IV. RESULTS

The interface provided for the system is quite intuitive. Several users tried using the application and they were able to use the extra core power by simply plugging in their systems in the network, installing the distcc daemon and logging on the web page. As discussed earlier, the systems in consideration are mostly idle hence there was never a problem in getting extra workers to help. The results are tabulated below.

A. Compilation Results for Linux Kernel 3.8.5

The system was used to compile Linux – 3.8.5 kernel for different numbers of nodes and for different system loads. 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 results for various situations are as tabulated in the following subsections. The make command was used with j parameter set to 12.

1) Compilation using a Single System

Table 2 lists the time for compiling the kernel on a single system, which was the Localhost.

TABLE 2: Time of Compilation for Single System

Real Time	User CPU Time	System CPU Time
5m 58.358s	20m 20.223s	4m 6.136s

2) Compilation using three systems with all Busy

Table 3 lists the time for compiling the kernel on all the three systems but they were all busy in doing some other jobs already. A reduction by almost one third

TABLE 3: Time of Compilation for Three Systems with all busy

Real Time	User CPU Time	System CPU Time
3m 3.638s	8m 31.508s	1m 6.576s

3) Compilation using three systems with one system down

Table 4 lists the time for compiling the kernel on all the three systems but before the compilation started the system with IP 192.168.1.30 went down and effectively the compilation occurred on two systems only. Since the build time has roughly stayed the same, we can conclude that using 3 was bit of overkill. As sending compilation jobs to remote systems includes a time delay introduced by the network, distcc only sends jobs for remote compilation if the local node has been given sufficient load.

TABLE 4: Time of Compilation for Three Systems with One System Down

Real Time	User CPU Time	System CPU Time
3m 1.405s	8m 18.175s	1m 6.332s

4) Compilation using all three systems

Table 5 lists the time for compiling the kernel on all the three systems but while executing command the order of the systems used for compilation was not given any consideration. As expected, the build time is reduced as the systems were not loaded.

TABLE 5: Time of Compilation for Three Systems

Real Time	User CPU Time	System CPU Time
2m 58.358s	8m 20.223s	1m 6.136s

5) Compilation using all three systems with fastest system listed first

Table 6 lists the time for compiling the kernel on all the three systems but while executing the command first system was the fastest system i.e. 192.168.1.30 and as per the design of Distcc, maximum load is sent to the system first in the order of command. Again, the results were as expected and the build time was reduced further.

TABLE 6: Time of Compilation for Three Systems with Fastest System Listed First

Real Time	User CPU Time	System CPU Time
2m 58.388s	7m 33.324s	1m 5.260s

V. CONCLUSIONS AND DISCUSSION

It is hypothesized that in the next decade, the advancements in the disk and the processor speeds will saturate, and we will see a growth in the only bottleneck that restricts the systems today, the bandwidth. With the systems connected via a fast network, the horizon of possibilities will be pushed a notch back, and we can expect to see revolutionary things being achieved by set of systems working together. The giant computing beasts would slowly give way to the colonies of smaller systems, who working alone are the small machines, but working together provide a computing power that was hitherto beyond the human reach. The fairy tale like description has started to shape up as a technology that marketing teams have labeled as the "cloud". As the companies start understanding the benefits of pinpointed costing, as they get used to the luxury of ordering more systems for their website that is suddenly seeing exponential growth in the footfall, as they understand that the petabytes of data they have cannot fit in the memory, as they start appreciating that the framework of computing they are used to lacks flexibility, the distributed systems will establish themselves as the de facto solution to cater the needs.

The ease of usage is one of the most critical and deciding factor in success of any technology. Ideally, any new system should blend with its predecessors. If the transition curve is too steep, the users hesitate in shifting, even if it offers some benefits, a running system providing delivering average result is any day better than a system that is difficult to configure but can deliver better results. Thus, effective and intuitive front end for the users will provide to be a catalyst in the growth of the distributed systems.

Distributed compilation is one of the techniques that help 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.

The experiments we did demonstrated that the gain that we expect of such a system is actually attainable and is not only intuitive. The interface provided by us to the system was also effective and most of the users were able to operate it without any difficulty at all.

The system can also be used as a reference for those who want a way to control a system of computers working together. 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 system that can be employed as it is on similar problems due to a modular design.

REFERENCES

[1] Aman Madaan, Dr. Sunil Kumar Singh and Ankur Aggarwal, titled "Junk Computing: Performance Evaluation and Comparison of an MPI based Heterogeneous Cluster", April 2013.

[2] 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.

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

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

[5] Gruman, Galen (2008-04-07). "What cloud computing really means".InfoWorld.http://www.infoworld.com/d/cloud- computing/what-cloud-computing-really-means- 031.Retrieved 2009-06-02.

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

[7] Java Script Library for Graph making "http://www.flotcharts.org/" .

[8] Jes Hall titled "Distributed Compiling with distcc", October 05, 2007.

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

[10] P. Mell and T. Grance, "Cloud Computing Definition", National Institute of Standards and Technology, Version 15, 10-7-09.

[11] Process information pseudo-filesystem,"http://linux.die.net/man/5/proc".

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