Frequency Scaling of CPU by Utilization

KEERTHI.R
CHRIST UNIVERSITY,BANGALORE

Abstract- Frequency scaling enables the kernel to issue instructions that slows the CPU (actually to reduce the CPU's clock speed) in a notebook computer when not in use. It will raise and lower the frequency of your processor depending on a set level of demand being made on the processor at the time. One of the reasons we want to do this would be to save energy and thus can save battery life on a laptop. Our paper makes use of the Linux kernel-Level governors for this purpose. We have devised a scaling algorithm that decides upon the frequency that dynamically needs to be set on a particular CPU core based on the utilization of the CPU core, unlike the existing 'ondemand kernel governor' that dynamically sets frequency of all the available CPU cores but not the one required in particular. This paper also uses benchmark to identify worst case performance loss when doing dynamic frequency scaling using Linux kernel governors and identifies average reaction time of a governor to CPU load changes. The battery-statistics is gathered and plotted to know about battery consumption and a method to decrease the battery consumption period with frequency scaling is formulated

Keywords- Frequency scaling, cpu frequency, gap analysis ,governors

1.INTRODUCTION

Mobile computing has got better with lighter components, better chips and faster processors. But the Achilles heel of a laptop has remained its battery. Modern graphic intensive operating systems and resource hungry applications are cutting down the life of your laptop's battery every day. The average battery life per con-tinuous use still stands at a maximum of three to four hours. So, a fast depleting battery could very swiftly put the crutches on our 'mobile' road trip. These systems require low power consumption to keep the operational time between the recharging of batteries long. One of the ways to achieve the above goal is frequency scaling the CPU based on CPU utilization.

Frequency scaling reduces the number of instructions a processor can issue in a given amount of time, thus reducing performance. Hence, it is generally used when the workload is not CPU-bound. It is proposed that CPU adapts frequency based on workload through a simple user feedback mechanism and rely only on CPU utilization. As a result of frequency scaling, power consumption will be reduced. It allows to save energy because usually neither program performance nor power consumption scale linearly with the processor frequency.

CPUfreq [1] refers to the kernel infrastructure that implements CPU frequency scaling. CPUfreq is the subsystem of the Linux subsystem that allows clock speed to be explicitly set on mobile processors. It is included in all recent kernels and enabled by default by recent distributions. Frequency scaling is usually handled by a governor program, according to system or user specific preferences. Likewise, CPUfreqd [2] is a very powerful and customizable daemon which can alter CPU frequency (and other ACPI settings) based not only on processor load, but also on battery level, temperature, power source, and what programs are running.

Rest of this section introduces the problem statement, similar systems and gap analysis. Section 2 details the system design used for implementing the application.Section 3 details the implementation of various elements described in the design.This section includes the algorithm used for CPU scaling policy. Performance met-rics and results are presented in Section 4. Section 5 presents some conclusions and scope for further development.

1.1 PROBLEM STATEMENT

The paper is aimed at developing a CPU frequency scaling application for Linux.The paper

caters to the need of a dynamic CPU frequency scaling based on the CPU utilization. The aim of this dynamic scaling model is to scale the CPU speed according to the workload (or according to a certain threshold) and that too without locking down the speed of CPU after scaling down. Amidst all research work done, it will be a system with an improvement over the existing systems. The paper also aims at making efficient consumption of the battery.

1.2 SIMILAR SYSTEMS AND GAP ANALYSIS

CPU frequency Scaling application is available for some Linux distros to scale the CPU frequency according to the CPU utilization. Some of the available software in this context are

1. CPU Frequency Monitor : The CPU Frequency Scaling Monitor provides a convenient way to monitor the CPU Frequency Scaling for each CPU. When there is no CPU frequency scaling support in the system, the CPU Frequency Scaling Monitor only displays the current CPU frequency. The state of the progress bar represents the current CPU frequency with respect to the maximum frequency. By default the CPU Frequency Scaling Monitor displays the current CPU frequency as a value in Hertz (the standard measure of frequency).

2. PowerNow [3] : PowerNow is speed throttling and power saving technology of AMD's processors used in laptops. The CPU's clock speed and VCore are automatically decreased when the computer is under low load or idle, to save battery power, reduce heat and noise. The lifetime of the CPU is also extended because of reduced electromigration, which varies exponentially with temperature. The technology is a concept similar to Intel's SpeedStep technology.The adaptation of PowerNow for AMD's desktop CPUs is called Cool'n'Quiet.

Gap Analysis

The CPU Frequency Scaling Monitor mostly used in the linux for dynamic CPU frequency scaling. However these applications using throttling the CPU through ACPI 'T' states is generally useless for power consumption reduction nowadays. It is an artifact of the past, when there was no clock frequency scaling and were mostly not implemented or did not exist.

Throttling does not decrease clock frequency at all, and it can even increase power consumption in a modern CPU capable of ACPI 'C' states [4], as it can interfere with the CPU reaching the higher C states. By frequency scaling the CPU downclocks to the lowest multiplier and remains locked in low speed. This happens irrespective of the scaling method in use (kernelspace or userspace) and of the frequency governor selected (ondemand, performance, etc.). It is also worth to mention here that the existing models used for determining frequency scaling algorithms are computationally expensive.

2. SYSTEM ARCHITECTURE

2.1 ARCHITECTURE

The system architecture is a 3-Level frequency management system where User-Level, Kernel-Level and architecture specific intricacies are handled at each level respectively. The levels interact with each other through interfaces to get control from user view to driver specific view of frequency management. The three levels are:

(a) User-Level

The user having the root permissions can make choices here from list of available frequencies/governors. The daemon program takes these inputs from user interface and passes it to the next level.

(b) Kernel-Level

The actual frequency scaling policies are defined here in the form of frequency scaling governors. These are the predefined policies

implemented in CPUfreq framework. This a gateway for frequency scaling.

(c) Driver-Level

In this level hardware drivers/acpi modules are present. Finally, CPU frequency is altered at this level.

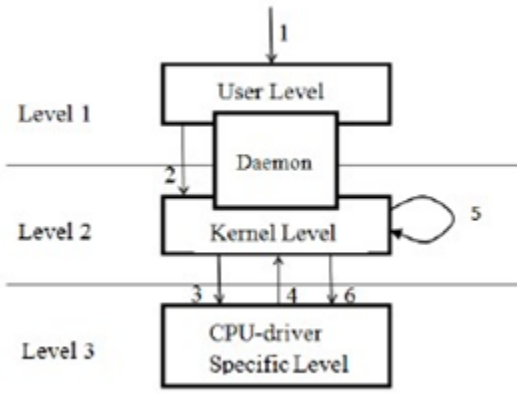


Figure 1: 3-Level Architecture For CPU Frequency Scaling

1. User requests for the desired frequency from the available CPU frequencies list.

2. User request is taken by the User-Level governors which will in turn request for a frequency scaling policy through daemon (viz. a program which continuously checks for the user action and tracks the CPU utilization. In the absence of any user action it will automatically scale the CPU frequency based on the utilization).

3. The required utilization parameters (like frequency and temperature) are fetched by the daemon by using the Kernel-Level governors which in turn uses the CPU specific drivers for the required data.

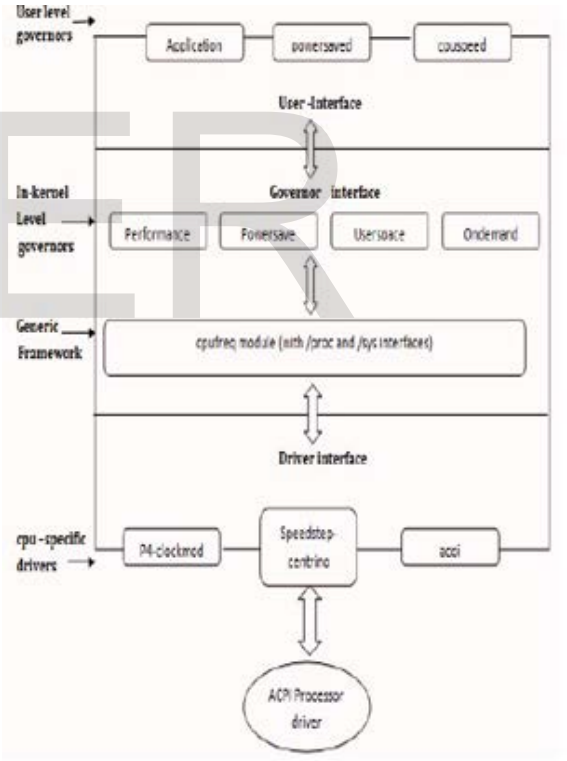
4. Once the required data is fetched the Kernel-Level governor will compare the fetched data to the pre-defined threshold values.

5. After the comparison the appropriate Kernel-Level policy is set.

6. The selected governor now sets the CPU frequency to nearest possible one amongst its frequencies range.

2.2 DETAILED ARCHITECTURE

The prime components of this 3-Level infrastructure are governors. The governor decides what frequency should be used using CPUfreq driver to actually switch the CPU's policy. The detailed description [5] of system architecture is as follows:



1. User-Level governors: The user-level governor give control to the user (superuser or root) to set the frequency on a particular platform. This userspace interface could then be used by the daemons running in userland to manage the CPU frequency over time, depending on the load. There are multiple userspace

programs, like cpuspeed and powersaved that can use userspace governor interface and change the frequency based on load. The userspace governors would typically sample the utilization every few seconds, and then take a decision on what frequency to go to for the next sample interval. This method of changing the frequency operates properly with almost any frequency/voltage-hanginghardware.

2. Kernel-Level governors: The CPUfreq infrastructure allows to frequency scaling policy governors, which can change the CPU frequency based on different criteria such as CPU usage. Which governor to use depends a lot on hardware and workload. Everyone has different needs and expectations from their system, and there is no one-size-fits-all governor. Each governor has its own individual advantages and disadvantages. Understanding how each governor behaves is the key to best leveraging the most power savings and performance from your hardware.Below is a brief description for four in-kernel governors used in CPUfreq:-

- Performance governor: This governor force the CPU to use the highest available clock frequency. This frequency is statically set and does not change. This governor is complementary to and most often utilized in conjunction with the powersave governor.

- Userspace governor: This governor exports the available frequency in formation to the user level (through the /sys file system) and permits user-space control of the CPU frequency. All user-space dynamic CPU-frequency governors, which run as daemons and control the CPU speed,use this governor as their basis. The CPUfreq governor 'userspace' allows the user, or any userspace program running with UID 'root', to set the CPU to a specific frequency available in the CPU-device directory.The userspace governor allows the user to define policy. It is the most customizable of any of the governors. It may save even more power, or yield even more performance, but this is largelydependent on how it is configured.

- Ondemand governor: The CPUfreq governor 'ondemand' sets the CPU frequency depending on the current usage. To do this the CPU must have the capability to switch the frequency very quickly. This gov ernor is a dynamic governor, it allows the system to achieve maximum performance if the system load is high and allows the system to achieve maximum power savings if the system is idle. The ondemand governor will step from one frequency to the next with respect to system load. The downside of this governor is latency. The system requires some time to ramp up or down in clock frequency.

- Conservative: The CPUfreq governor 'conservative', much like the 'on demand' governor, sets the CPU depending on the current usage. It differs in behaviour from later, as it gracefully increases and decreases the CPU speed rather than jumping to max speed the moment there is any load on the CPU. This behaviour is more suitable in a battery pow ered environment. The governor is tweaked in the same manner as the 'ondemand' governor.

3. CPUfreq module (generic framework): The CPUfreq module provides a common interface to the various low level, CPU-specific frequency-control technologies and high-level CPU frequency-controlling policies. CPUfreq decouples the CPU frequency-controlling mechanisms and policies and helps in independent development of the two. It also provides some standard interfaces to the user, with which the user can choose the policy governor and set parameters for that particular policy governor.

4. CPU-specific drivers: Various low-level, CPU-specific drivers implement various CPU frequency-changing technologies, such as Intel SpeedStep Technology, Enhanced Intel SpeedStep Technology, and Pentium 4 processor clock modulation. On a given platform, one or more frequency-modulation technologies can be supported, and a proper driver must be loaded for the platform to perform efficient frequency changes. The CPUfreq infrastructure allows the user to use one CPU-specific driver per platform.

Below are some of them used in CPUfreq:-

- SpeedStep: Intel's SpeedStep Technology reduces the latency associated with changing the voltage/frequency pair (referred to as P-state). This aids for the frequency transitions to be practically undertaken more often, which enables more-granular demand-based switching and the op timization of the power/performance balance based on demand.

- ACPI (Advanced Configuration and Power Interface): Usage of the acpi-CPUfreq reduces the voltage along with CPU clock frequency, allowing less power consumption and heat output for each unit reduction in performance.

- p4-Clockmod: p4-clockmod driver in this context is used to reduce the clock frequency of a CPU, but it does not reduce the voltage.

2.3 TECHNOLOGIES

Implementation of the frequency scaling software was done with the following set of standard technologies:

- Development Language:- C
- Compiler:- gcc
- Linux Kernel version:-2.6.xx-generic
- Platform:- Ubuntu 11.10 Linux Desktop OS
- Libraries:- Gtk library, CPUfreq library, pthread library

3.IMPLEMENTATION

We have implemented the proposed system architecture in three levels

1. User-Level
2. Kernel-Level
3. CPU-Specific-Level

3.1 USER-LEVEL

At the User-Level a daemon is implemented which is responsible for frequency scaling of the CPU.

A daemon is a process that runs in the background automatically and is independent of control from all terminals ,i.e., not being under the direct control of an
interactive user. Since a daemon does not have a controlling terminal, it cannot just fprintf to stderr. So, we are using syslog system call which in turn uses syslogd, is the daemon that implements the system logging facility.

Daemon works in a way that it reads input parameters such as frequency, core number and governor given by the interactive user using graphical or command line interface. It also calculates each core utilization using get util script. So, daemon keeps on working in the background and keeps on checking whether parameters are input or not. The daemon then works accordingly if parameters are input or not and if not then work based on utilization of different cores in the system.

Script File: The utilization of each individual CPU is obtained by running shell script get util.sh. The script reads /proc/stat file. It contains the number of CPU ticks performed since the system has been booted. A typical snapshot of /proc/stat [6] [7] file is given in Table 1

cpu	68854	0	19526	2763461	11898	0	400	0	0	0
cpu0	24904	0	6835	675495	5423	0	33	0	0	0

Table 1: Snapshot of /proc/stat

The numbers in each line identify the amount of time the CPU has spent performing different kinds of work. Time units are in USER HZ or Jiffies (typically hundredths of a second on most architectures, use sysconf (SC CLK TCK) to obtain the right value).

The meanings of the columns are as follows, from left to right:

1. user: normal processes executing in user mode
2. nice: niced processes executing in user mode
3. system: processes executing in kernel mode
4. idle: twiddling thumbs
5. iowait: waiting for I/O to complete
6. irq: servicing interrupts
7. softirq: servicing softirqs

Since Linux 2.6.11, there is an eighth column, steal - stolen time, which is the time spent in other operating systems when running in a virtualized environment.

Since Linux 2.6.24, there is a ninth column, guest, which is the time spent running a virtual CPU for guest operating systems under the control of the Linux kernel.

So, to get utilization of CPU core first we extracted the amount of time core remains idle from 5th column and total time, i.e., addition of all time units that a CPU has spent at that instant. Then we took two snapshots of this file at two different instants and took the difference to find how much percentage of time core was idle during that period using following formula:

$$\text{idle period}(\%) = \frac{(\text{idle}(2) - \text{idle}(1))}{(\text{total time}(2) - \text{total time}(1))}$$

∴ utilization of core(%) = 100 - idle period

1. GUI :

GUI has been implemented using the GTK 2.0 [8] environment with C interface. GTK (GIMP Toolkit) is a library for creating graphical user interfaces.It is licensed using the LGPL license.

- GTK Widgets: The GUI makes use of the GTK Widgets viz Radio but tons, labels used to select the system frequencies under one column, CPU core under the other column and display the CPU core and their current frequencies in the last column.
- C Interface: The C interface is used to get system details from the /sys/devices/system/cpu interface to obtain the system defined frequency values and also the current CPU core frequency and the scaling governor. Its also writes the user selected values from the GUI into a file which is used by the daemon to scale the CPU frequencies.

A typical GUI implementation of client interface with gtk2.0.

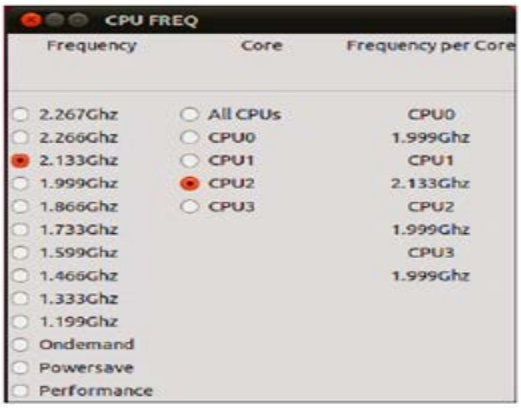


Figure 3: User Interface For CPU Frequency Scaling

2. Command Line Interface :

The user can change the frequency of the CPU as desired through command line interface. The command line interface implemented accepts user input viz, desired frequency, cpu core number and governor parameters. This interface will check for valid frequencies that can be set on a CPU core. The valid scalable CPU frequencies are obtained from the below file

$/sys/devices/system/cpu/cpu(k)/cpufreq/scaling available frequencies where k is the number of the CPU core.

This file gives those frequencies on which the CPU core frequency can be set.Since the number of cores are different for different systems we first obtain the number of available cores in a particular system using the below command

$ sysconf(SC NPROCESSORS ONLN)

This interface checks for the available cores in a system for which frequency can be scaled. All necessary conditions are checked in which user can input only those frequencies to which a core can be scaled. If the user inputs frequency greater than the largest available frequency, the core will be scaled to a the largest scalable available frequency for that particular core and if the user tries to input a frequency less than the lowest available frequency than the core will
be scaled to the lowest available frequency. In addition, it checks another condition in which if the user inputs both frequency and governor. If both are given then by default userspace governor will be set.

3.2 KERNEL-LEVEL

In the kernel-level implementation firstly all the governors are loaded by using the acpi interface, then the set policy() is called to scale the CPU frequency in two conditions viz, with user-input and without user-input.

Implementation of the CPUfreq governors is done using the ACPI interface.

1. ACPI Interface :

ACPI (Advanced Configuration and Power Interface) is an open industry speci fication establishing industry-standard interfaces for OS directed configuration and power management on laptops, desktops, and servers.
ACPI [9] enables new power management technology to evolve independently in operating systems and hardware while ensuring that they continue to work together. Before selecting and configuring a CPUfreq governor, first appropriate CPUfreq driver is added. Below are the steps used to set up the CPUfreq:

1.CPUfreq Setup

- How to Add a CPUfreq Driver Use the following command to view which CPUfreq drivers are available for your system:

$ls/lib/modules/[kernelversion]/kernel/arch/[architecture]/kernel/cpu/cpufreq/Sample Output: e powersaver.ko p4-clockmod.ko pcc-cpufreq.ko

Use modprobe to add the appropriate CPUfreq driver.

$ modprobe [CPUfreq driver]

When using the above command, be sure to remove the .ko filename suffix

- Enabling a CPUfreq Governor If a specific governor is not listed as available for your CPU, use modprobe to enable the governor you wish to use. For example, if the ondemand governor is not available for your CPU, use the following command:

$ modprobe cpufreq ondemand

Once a governor is listed as available for your CPU, you can enable it
using:

$echo[governor]>/sys/devices/system/cpu/[cpu ID]/cpufreq/scaling governor

2. Set Policy()

For scaling the CPU frequency, we make use of the Kernel-Level governors (mentioned in the detailed architecture in section) and define a policy for the purpose which is executed by the daemon. Before applying the policy the the daemon first checks is the user is trying to set the frequency manually through the GUI or through the command line interface. This condition is checked by setting a User flag which is set to 1 in case of user input given, if not it remains set to 0.

Algorithm used for frequency scaling is mentioned below in Algorithm 1.

Algorithm 1: (Set Policy)

```
1  Set Policy()
2  # DEFINE lower threshold=20;
3  # DEFINE higher threshold=80;
4  # DEFINE UserFlag=0;
5  utilization = read utilization file;
6frequency,governor,cpuno=read(freq,gov,cpuno);
7  if (UserFlag==0) then
8  if (0 < utilization < lower threshold)
   then
9  then governor="powersave";
10  end
11  if (lower threshold < utilization <
    higher threshold ) then
12 then governor = "ondemand";
13 end
14 if (utilization > higher threshold ) then
15 then governor="performance";
16 end
17 end
18 else
19 set governor= "userspace";
20 frequency = find nearest possible frequency
(from system);
21 if (cpuno==null) then
22 do for all cpu(frequency,governor);
23 end
24 else
25 do for selected cpuno(frequency,governor);
26 display current settings now;
27 end
28 end
```

Line 1-4 defines the threshold values for changing the governor based on utilization.

Line 5-6 reads the utilization of cpu core and input parameters like frequency, governor if user has provided.

Line 7-17 specifies the case when input is not specified by the user then upper and lower threshold for CPU utilization are used to select the frequency scaling policy. If utilization is less than lower threshold, then 'powersave' governor is used and if utilization is between lower threshold and upper threshold then governor is set to 'ondemand' and if the utilization is greater than upper threshold then governor is set to 'performance'.

Line 18-28 specifies case when user has given input then frequency is scaled using 'userspace' governor for particular core or for all cores depends upon user input.

3.3 CPU-SPECIFIC LEVEL

At the CPU specific level we make use of the battery status to improve the frequency scaling of the CPU.

1. Battery Status Check- The status of the battery can be obtained from the interface /proc/acpi/battery.Mainly there are three possible states of battery for device which can work on battery. These are-

(a) Charging- This means that the battery is currently charging, which intern means that the device is plugged in.
(b) Charged- This means that the battery is charged, but the device is still plugged in for charging.

(c) Discharging- This means that the device is not plugged in for charging, and the battery is discharging.

There is a file "/proc/acpi/battery/BAT0/state" maintained by the OS which logs the current battery status.

We are reading the status from this file, and according to that status we are implementing our policies. The policy is that if the status is either 'charging' or 'charged', then we are just setting the governor to performance mode, else, i.e., the status is 'discharging', we will set the governor to ondemand.

2. The above set policy() algorithm to scale the CPU frequency is modified according to current the battery status. Refer Algorithm 2

Algorithm 2: (Set Policy with battery status check)

1 Set Policy()
2 # DEFINE lower threshold=20;
3 # DEFINE higher threshold=80;
4 # DEFINE UserFlag=0;
5 utilization = read utilization file;
6 frequency,governor,cpuno=read(freq, gov, cpuno);
7 if *(UserFlag==0)* **then**
8 if *(battery status=="charging"||battery status=="charged")* **then**
9 set governor="performance";
10 end
11 else
12 if *(0 < utilization < lower threshold)* **then**
13 governor="powersave";
14 end
15 if *(lower threshold < utilization < higher threshold)* **then**
16 governor = "ondemand";
17 end
18 if *(utilization > higher threshold)* **then**
19 governor="performance";
20 end
21 end
22 end
23 else
24 set governor="userspace";
25 frequency = find nearest possible frequency (from system);
26 if *(cpuno==null)* **then**
27 do for all cpu(frequency,governor);
28 end
29 else
30 do for selected cpuno(frequency,governor);
31 display current settings now;
32 end
33 end

This algorithm is an add on over previous algorithm. Line 8-10 takes into account the battery state information and in case if it is charging or charged then governing policy is tuned to be 'performance' and while discharging utilization based algorithm is applied same as before.

3.4 WORKING OF 3-LEVEL IMPLEMENTATION OF CPU FREQUENCY SCALING

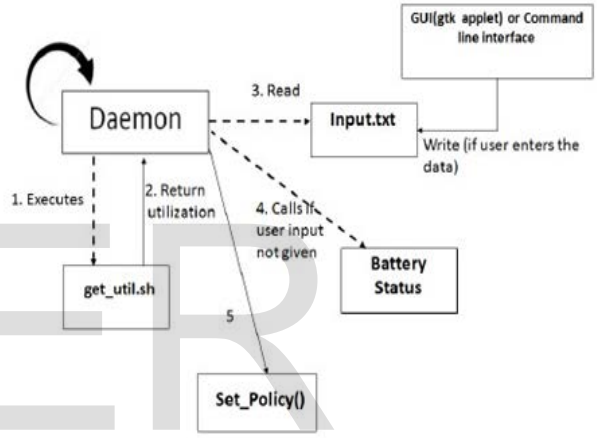


Figure 4: Working Of 3-Level Implementation

1. In the first step the daemon calls for shell script 'get util.sh' to fetch the utilization of each CPU.

2. The utilization is then read from file 'loadstat.txt' and utilization parameters are returned back to the daemon in this stage.

3. Next the daemon monitors if any user input is being given via GUI or command line interface written into the file 'Input.txt'. sets the CPU frequency accordingly.

4. In the absence of user input it checks for the battery states viz 'charged' or 'charging' and sets the governor as 'Performance' governor. For any other

battery status and absence of user input it gets the CPU utilization from 'get util.sh' file.

5. The 'set policy()' method is called to set the CPU frequency accordingly. The above steps are repeated by daemon running in background continuously. Any changes in current frequencies are reflected back to user in GTK applet.

4.OBSERVATIONS AND RESULTS

4.1FREQUENCY SCALING VS UTILIZATION

When the dynamic frequency scaling algorithm based on utilization of cores is implemented on a 2.4 GHz processor with 4 cores, the frequency response for that is shown in Figure 5.

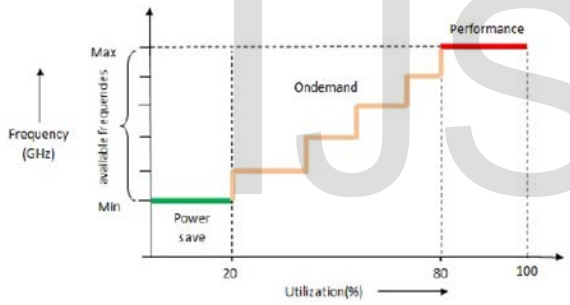


Figure 5: Frequency Scaling With Utilization of intermediate frequencies

For workloads below the lower-threshold (say 20%), the frequency is set to minimum and for workloads greater than upperthreshold (say 80%) the frequency is set to maximum. The average workloads are handled by 'ondemand' governor which dynamically switches between minimum and maximum frequency in gradual steps Performance With Time After creating our frequency scaling policy and integrating it with daemon, we tried to analyse the performance of our governor under different CPU workloads. For this we used, "CPUfreq-bench" [10] as a benchmark program which is standard for CPUfreq framework testing and identified worst case performance loss when doing dynamic frequency scaling using Linux kernel governors.

You can specify load (100 % CPU load) and sleep (0% CPU load) times in micro
seconds in a configuration file which will be run X time in a row (cycles):

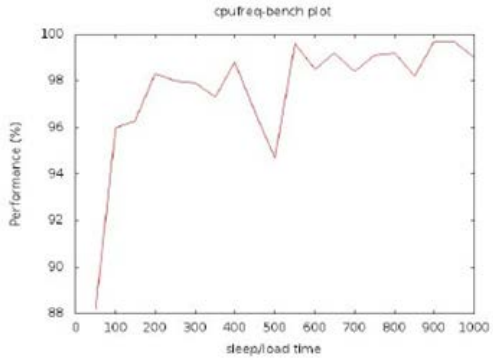


Figure 6: Performance Benchmark Run With Conventional Ondemand Governor

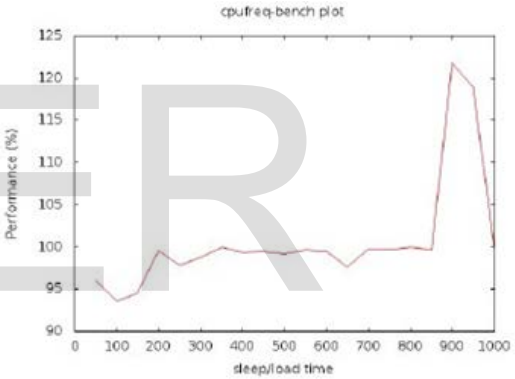


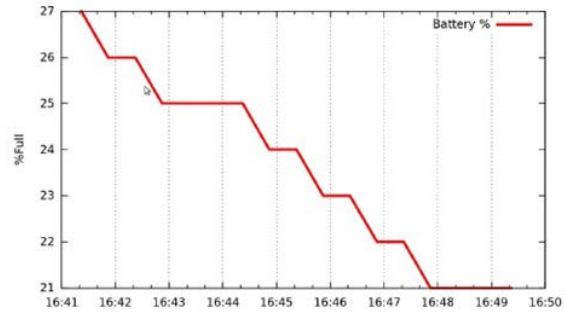
Figure 7: Performance Benchmark Run With Utilization Based Governor: From these two observations we concluded that there is a performance improvement while using utilization based governor in comparison conventional governor.

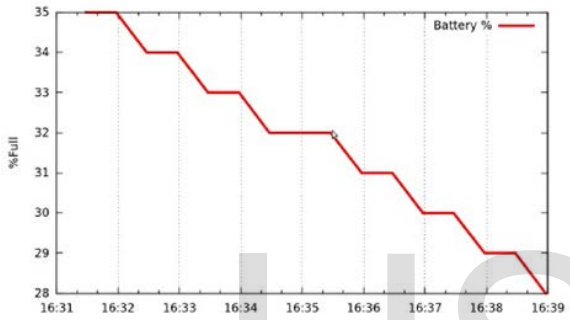
From the figures we can notice that at higher loads performance in utilization based governor varies from 95% to 120% where as conventional ondemand governor it varies from 88% to 100%.

4.3 BATTERY STATISTICS

We collected the battery stats from both conventional and utilization based governor with the help of battery-status-collector [11] for time period of 8 minutes as shown in Figure[8, 9]. It was observed that in case of ondemand governor it varies from

27% full to 21% full while utilization based governor gives a battery drop from 35% to 28% under same loads and same time period.





variable CPU loads for almost 30 minutes over a 78 % charged battery.

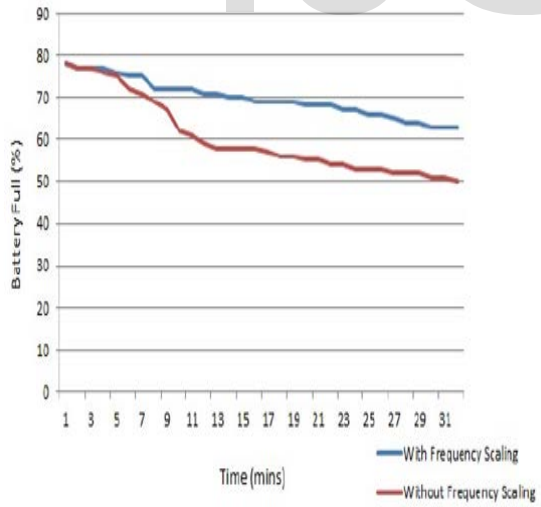


Figure 10: Battery Discharging Patterns With/Without Frequency Scaling

Here it is evident from the graph that the slope is more steeper for conventional than utilization based policy where battery discharging pattern is gradually coming down.

4.4 POWER CONSUMPTION

The power consumption statistics was gathered for last 10 minutes of System's CPU usage, and plotted as the power consumed in Watts versus time elapsed for both the conventional and utilization based frequency scaling.

Figure 11 shows the power consumption without frequency scaling. It is seen that between minutes 3 and 4 when load was high, the power usage shoots greater than 2 Watts.

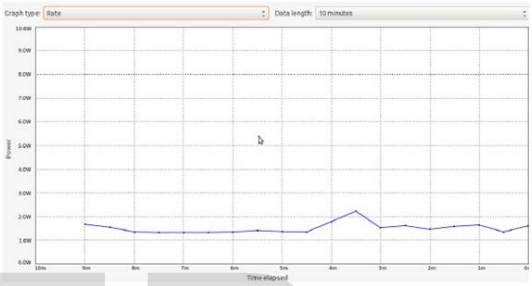


Figure 11: Power Usage Without Frequency Scaling

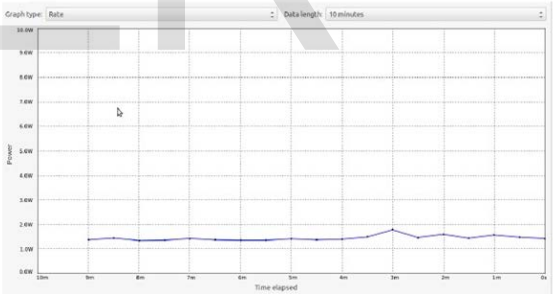


Figure 12: Power Usage With Frequency Scaling

But as you can see in Figure 12 with frequency scaling the power usage remained
below 2 Watts throughout. This shows that utilization based governor can be tuned to save power.

5.CONCLUSION

Traditional frequency scaling involved CPU throttling, which was not efficient. Now the CPUfreq framework along side with acpi drivers allows us to dynamically scale the frequency, so

that we can tune our system to performance/powersave policies as needed.

In the current implementation we are able to scale frequency of each CPU core based on core utilization and also user has privileges to change CPU parameters like frequency, governor etc. We have implemented both graphical interface using gtk and command line interface for changing CPU parameters. Also, we incorporated
battery status like charging, charged or discharging for improving battery consumption. When system becomes idle, frequency goes down and considerable amount of power is saved with slight loss of performance occasionally. When the utilization goes up frequency increases to match the performance. Our results suggest that this mechanism provides a practical and effective way to save significant amounts of
energy in many applications, with acceptable performance degradation.

REFERENCES

[1] D. Brodowski and M. Dongili, "cpufreq-info(1)," May 1991. [Online]. Available:
http://linux.die.net/man/1/cpufreq-info
[2] G. Staikos and M. Dongili, "Installation guide for Cpufreqd," August 2011.
[Online]. Available: http://www.linux.it/~malattia/wiki/index.php/Cpufreqd
[3] PowerNow! [Online]. Available: http://en.wikipedia.org/wiki/PowerNow!
[4] V. Pallipadi, A. Belay, and S. Li, "cpuidle. Do nothing efficiently.." July
2007. [Online]. Available: http://www.linuxsymposium.org/archives/OLS/Reprints-2007/pallipadi-Reprint.pdf
[5]Pallipadi,"EnhancedIntelSpeedStepTechnologyandDemandBasedSwitchingonLinux,"February2009.[Online].Available:http://software.intel.com/en-us/articles/ enhanced-intel-speedstepr-technology-and-demand-based-switching-on-linux/
[6] proc Linux Programmer's Manual (5). [Online]. Available http://www.
linuxhowtos.org/manpages/5/proc.htm
[7] /proc/stat explained. [Online]. Available: http://www.linuxhowtos.org/System/procstat.htm
[8] T. Gale and I. Main. GTK+ 2.0 Tutorial. [Online].Available:http://developer.gnome.org/gtk-tutorial/2.90/
[9] L. Brown, A. Keshavmurthy, R. Moore, D. S. Li, V. Pallipadi, and L. Yu, "ACPI in Linux Architecture, Advances, and Challenges,"(Intel Open Source Technology Center), 2005. [Online]. Available: http: //www.linuxsymposium.org/2005/linuxsymposium procv1.pdf
[10] cpufreq-bench. [Online]. Available: http://lwn.net/Articles/339862/
[11] BATTERY-STATS-COLLECTOR(8). [Online]. Available: http://manpages. ubuntu.com/manpages/natty/man8/battery-stats-collector.8.html
[12] A. Mallik, B. Lin, G. Memik, P. Dinda, and R. P. Dick, "User driven frequency scaling," vol. 5, Department of Electrical Engineering and Computer Science, Northwestern University, July-December 2006, pp. 1–4.
[13] G. Kroah-Hartman, Linux in a nutshell. O'Reilly, 1998, vol. 3.
[14] V. Pallipadi and A. Starikovskiy, "The ondemand governor. past, present, and future," vol. 2, Proceedings of the Linux Symposium, July 2006, pp. 1–8.