Analysis of Frequency Change for Compute-Intensive Task using Scheduler –Driven Frequency Scaling scheme in Linux Kernel

[1]Prof. Sunita Dhotre, [2]Pooja Tanaji Patil, [3]Dr. Suhas .H Patil

[1] Associate Professor, [2]PG Scholar, [3]Professor
Department of Computer Engineering,
Bharati Vidyapeeth Deemed University College of Engineering
Pune, India

***Abstract*— Linux based systems are getting more numerous and complex day by day. To meet the multiplicity of user need, the CPU frequency and software complexity is also increasing which demands the high power in such systems. But, the performance of system did not increase significantly. To maintain the performance of such system the analysis of frequency change is an essential task. The analysis of change in frequency leads to consume less power in the system. To reduce the energy consumed by these embedded systems the frequency of CPU has to minimize. This paper enhances the performance of Linux Operating systems by minimizing the response time with changing the frequency of scheduler-enabled DVFS scheme by using Compute- Intensive Task.**

***Keywords* — Linux Operating system, power management, performance, Response Time, CFS, CPU Frequency.**

I. Introduction

Modern Linux operating system is experiencing considerable growth in performance and functionality to meet the multiplicity of user need. For such systems, the CPU frequency and software complexity [1] is increasing day by day which demands the high power. But, the performance of system did not increase significantly. Hence, the user experience is greatly affected due to higher Response time. So, performance maintenance is the biggest challenge faced by today's Linux based systems. Performance of such systems can be maintained by scaling the frequency [4] and minimizing the response time. The frequency scaling scheme in Linux operating system is dynamic voltage and Frequency Scaling (DVFS). DVFS framework is supposed to save the power [1] at highest level by executing the processes at minimum performance level. The proposed research work emphasis on designing a Scheduler-enabled DVFS Scheme by estimating frequency change analysis for Compute-Intensive Task to minimize the Response time and to enhance the system performance.

II. Literature Survey

In the paper [3] author C. S. Wong, R. D. Kumari, and J. W. Lam has compared the two Linux kernel scheduler such as O (1) and completely fair scheduler (CFS) with regard to fair sharing policy and interactive performance. In Linux kernel 2.6 O(1) scheduler is used while in 2.6.23 uses the CFS. O(1) replaced by CFS . The results from the test conclude that the CFS is fairer than O(1) in the case of CPU bandwidth distribution and interactive performance.

In paper [7] One of the critical issues of OS is the distribution of several processes among the the multiple cores in order to achieve the fairness, and better performance. Fairness is very important criteria of any operating system scheduler. This allocation of tasks is done by Load balancer by transferring the task from heavily loaded processor to lightly loaded processor. The algorithm proposed by author provides in paper [7] is the balancing of the threads by its time which are running on both heavy and light processores. To evaluate the effectiveness of proposed scheme, author implemented one algorithm in Linux Kernel and demonstrate the efficiency of by performing different experiment. Finally, the results evalution of proposed algorithm concluded that author's proposed scheduler attains high performance and better fairness as compaired to earlier schedulers which is advantageous for multicore processors

In paper[2] author J. Wei, R.Ren, Juarez, F. Pescador provides the scheduling algorithm such as Energy based Fair Queuing (EFQ) which consume the energy on many devices. So, the CFS is extended by adding the new scheduling policy SCHED_EFQ.

In the paper [4] author has estimated the response time performance for smartphones. This response time estimation scheme is proposed by applying DVFS at CPU and CFS at Linux kernel. DVFS which controls scheduling and reduces the power consumptions in smartphones through adaption of CPU core frequency level and system voltage. The change in CPU frequency ultimately changes the Response Time. The effectiveness of proposed scheme is demonstrated by capturing various changes in frequency levels based on executing various background applications for Smartphone.

The research paper [1] focuses on maximizing user experience in battery limited embedded system by using Energy-fair queuing which is class of energy-aware scheduling algorithm. Author merges the traditional energy-efficient algorithm with EFQ to maximize user experience. The other energy-saving scheme such as combination of DVFS with the application self-adaption can be used to consume power and maximizing user experience.

III. Related Work

The above observations and literature studies [3-7] indicate that CFS is not connected with the frequency scaling scheme of the CPU. As there is a possibility to enhance the system perforamnce by minimizing the response time and by changing the frequency,[29][30] CFS can be linked to the Dynamic Voltage and Frequency Scaling (DVFS) Algorithm. This leads to the need of design of a DVFS Scheme with an added scheduler governor also the response time of Compute-Intensive Task can be estimated to enhance the system performance.

This leads to the need of design of scheduler driven frequency scaling scheme wherein the responsive time of Compute-Intensive Task is estimated by analyzing the change in frequency. Scheduler-driven Frequency scaling scheme desires to exploits both the global information and per-task in the scheduler to improve the frequency selection scheme and achieves better performance and lesser energy consumption[5] so, to obtain the efficient performance of the system, the proposed work utilizing the multi-threaded program executing in a multi-processor environment. The multithreaded program is implementing by compute-intensive task where the only CPU is utilized.

Completely Fair scheduler (CFS)

The latest Linux kernel scheduler is Completely Fair scheduler (CFS) [7][12][31]which was introduced in Linux Kernel 2.6.23 and is continued in 2.6.24. CFS is “Desktop” process scheduler which was implemented by Ingo Molnar. Its core design can be summed up in single sentence: “CFS basically models an 'ideal, precise multitasking CPU' on real hardware [9][14].” The primary function of CFS is to divide the processor time of resources between the runnable processes.

The CFS Scheduler supports the following scheduling policies [11] -

SCHED_NORMAL /SCHED_OTHER : It is used for regular tasks.

SCHED_FIFO : Uses the First-In-First- Out Scheduling Policy.

SCHED_BATCH : It is used for running the tasks for longer time without preempting it .

SCHED_IDLE: it is used to avoid the tasks to get into the priority.

SCHED_RR : Similar to SCHED_FIFO, but uses the Round Robin scheduling algorithm. It is impossible to get the ideal CPU in reality, but the CFS tries to imitate such ideal processor in system [16]. For scheduling the process CFS uses the process priority and timeslice [11][17]. timeslice is defined as the total amount of time taken by process to run. The process which is having the large timeslice is considered as higher priority process.

Time slice [16][31] is defined as-

$$\text{Timeslice}=\text{share}\times\text{period} \qquad (1)$$

in equation (1), the period is the total time slice that is used by scheduler for all tasks. The minimum period is 20ms. To calculate the priorities of tasks, CFS uses task weight represented by load.weight and divides runtime by tasks weight which is saved as vruntime in a RBtree.The nice value given to each process according to user’s perspective determines the priority of process. The ratio of time taken by any processor is determined by the difference between the nice values of runnable process and the nice value of process itself. To decide the balance among multiple tasks CFS inaugurated the concept of “virtual runtime (vruntime)” [13]. Virtual runtime elucidate as the total amount of time provided to given task. The task which is having small virtual time means it has higher priority and will schedule first. The virtual runtime can be considered as a weighted time slice, which is represented by following equation- [11][12]

$$virtualruntime += \frac{(delta\_exec)(\text{default weight of process})}{se(load.weight)} \qquad (2)$$

From the equation (2) of virtualruntime , delta_exec is the total amount of execution time of task, default weight of process means the unit value of weight and load.weight is weight of task/entity[19][31]. The weight of runnable processes is decided by their priority. By assigning the proper weights to processes the CFS maintains the fairness. In CFS share is calculated as –

$$\text{share} = \frac{\text{weight of schedulable entity}}{\text{total weight of all entities in CFS runqueue}} \qquad (3)$$

This scheduler also maintains the fairness for those processes which are waiting for I/O events to occur. Instead of maintaining these processes in run queue, the Completely Fair Scheduler maintains the time order Red-Black tree (RBTree) [31] in a view to decide the task to schedule next on CPU. each node in tree is task/process in the system, and key value of the node represent the virtual runtime of the specific task. According to the definition of Red-black tree, left most node has smallest key value, which means that this task has highest priority with smallest virtual runtime and vice versa.

Hence CFS has to take left most tasks for processing and once the task is processed then it is removed from tree.

Completely Fair Scheduler algorithm

Implementation of pick_next_task_fair() of CFS schedule class [10] [11][31]–

1. It first examine whether there is any running task in cfs_rq or not.
2. If the running task is there in cfs_rq the it calls pick_next_entity() method which implicitly call rb_entry() to obtain the next task with smallest vruntime value.
3. The method sets_next_entity conduct updation in values of sched_entity of node/ next task which is obtained in step 2 .

4. To return the obtained next task group CFS algorithm calls group_cfs_rq. This grouping feature attains fairness among users as well as groups rather than just among the tasks.

DVFS (Dynamic Voltage and Frequency scaling)

The DVFS holds a set of governors namely Performance, Powersave, userspace, Conservative and Ondemand[15] [24]which allows to conser the CPU power in Linux kernel. These CPU Frequency Scaling Governors allows the drives to set the target frequency. Dynamic frequency Scaling [16] mechanism is applied for using the CPU efficiently.

- Performance Governor- This sets the processor clock speed to the maximum to allow the maximum performance. This governor does not allow saving the power of system but it can allow the dynamic changing frequencies.
- Powersave Governor - This governor sets the CPU to the lowest available frequency however a range of frequencies can be adjusted.
- Userspace Governor - The frequency is set manually in this governor. It does not dynamically change the frequency.
- Ondemand Governor - This governor was introduced in the Linux Kernel 2.6.10. It dynamically changes the CPU frequency depend on the utilization of Processor.
- Conservative Governor – Based on utilization of processor this governor dynamically adjusted the frequency by gradually increasing the frequency level.

These governors operate at a particular fixed frequency and have a disconnected design with the Completely Fair Scheduler. Linux CFS is a strictly fair scheduler having various insertion points which can govern the CPU Frequency [17].

Compute-Intensive Task

Compute-Intensive is any task or application of computer which needs a lot of CPU/Computation[24]. These tasks are spends more time in executing the codes so also known as CPU bound processes in the operating system (OS). Linux scheduling policies attempt to achieve two goals such as fast response time and high throughput. So, in order to evaluate the performance measurements of scheduler-driven frequency scaling scheme the Compute-intensive tasks are implemented. The change in frequency is estimated by executing the Compute Intensive Task to enhance the performance of the system.

IV. CONTRIBUTION OF RESEARCH

This thesis works on designing the scheduler driven frequency scaling scheme and executing compute intensive task for minimizing the response time and to achieve efficient performance. The analysis of change in frequency will be carried out by running Compute- Intensive Task which utilizes the system performance. Hence the objectives of proposed system are-

- The current array of cpufreq governors is replaced with a new governor.
- To invoke DVFS methods through CFS algorithm to design the scheduler-driven frequency scaling by creating a new patch of the algorithm in existing algorithm of CFS.
- Analysis of Frequency and response time by executing Compute Intensive application.
- Comparative analysis of proposed shed governor with existing governors by graphical representation of frequency vs. response time for compute intensive task.

V. PROPOSED SYSTEM

This research works on designing the scheduler-driven frequency scaling scheme to enhance the system peroformance. The analysis of change in frequency will be carried out by running Compute- Intensive Task which utilizes the system performance. Considering that the scheduler in the kernel plays a vital role in today's multi-core operating systems for estimating the performance. The proposed system aims to obtain the connectivity among the scheduler and DVFS scheme in order to minimize the Response time of the process and provide the better system performance.

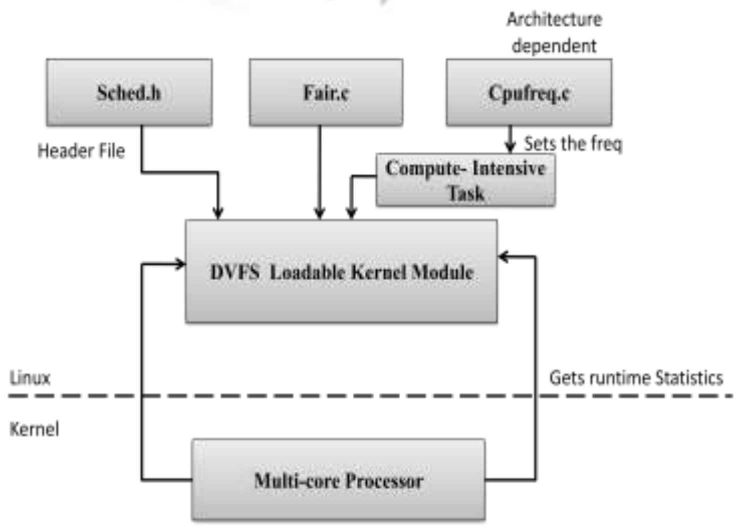

Fig. 1 : System Level Implementation

As shown in fig.1 , the Existing DVFS algorithm is loaded in the kernel module along with the existing set of governors. The modifications are done in CFS header file sched.h and CFS Code fair.c which adds a new governor 'sched' having new cpufreq. The cpufreq subsystem of Linux provides an interface to manage the CPU frequency. In Linux the governors scale the frequency dynamically and in controlled way. The new capacity of the CPU is generated at various points within CFS including Load Balance, and finally a call is made to update the capacity of the CPU which then converts the new minimum capacity request into the CPU Frequency. Then the Compute-Intensive task is executed by setting different governors on DVFS loadable kernel for the analysis of change in CPU frequency and to estimate the Response time accordingly.

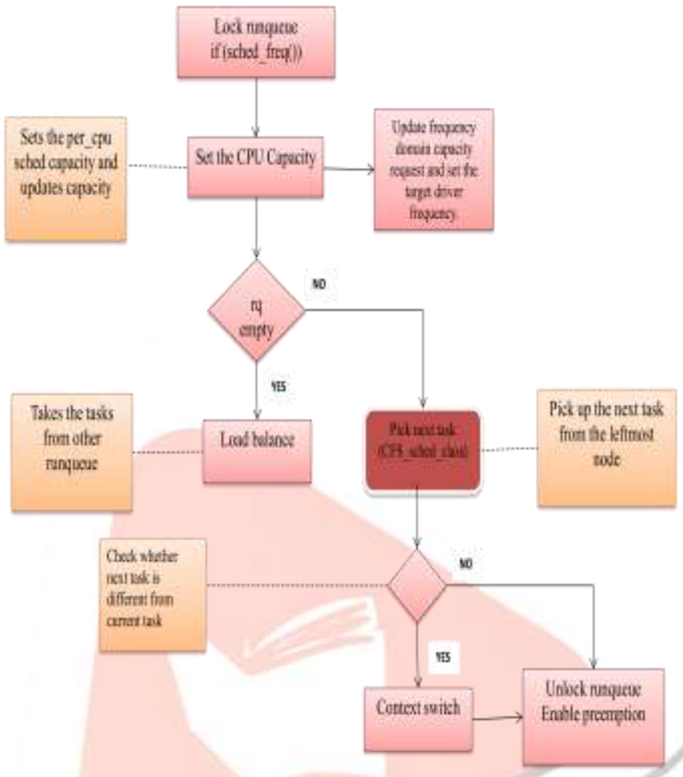
Lock runqueue if(sched_freq())
Sets the per_cpu sched capacity and updates capacity
Set the CPU Capacity
Update frequency domain capacity request and set the target driver frequency
rq empty
NO
YES
Takes the tasks from other runqueue
Load balance
Pick next task (CFS_sched_class)
Pick up the next task from the leftmost node
Check whether next task is different from current task
NO
YES
Context switch
Unlock runqueue Enable preemption

Fig. 2 : Flow Chart of Proposed System

Algorithm of Proposed System –

```
 { For each CPU :  cur_cpu}
1.    for all sd in sched_domains of cur_cpu do
2.    calculate the load of runqueue;
3.         if sd has idle cores then
4.                first_cpu = 1st idle CPU of sd
5.        else
6.                first_cpu  = 1st CPU of sd
7.        end if
8.    if cur_cpu  ≠ first_cpu then
9.        continue
10.   endif
11.   for all sched_group sg in sd do
12.   enqueue the tasks in runqueue;
   for the tasks that are new or waking up trigger the frequency switch
13.   if (task is new || tasks is wakedup) update capacity of (cpu(rq))
14.   sg.load=average loads of CPUs in sg
15.   for dequeue remove the task from the rbtree and update the fair scheduling status
16.         if (task is in sleep state) update capacity of (cpu(rq))
17.         Raise the target cpu's Operating Point Frequency;
18.   set the driver target frequency using cpu frequency table with new value
19.   end for
20.   busiest = overloaded sg with the highest load
21.         (or,  if inexistent) imbalanced sg with highest load
22.         (or, if inexistent) sg with highest load
23.   local = sg containing cur_cpu
```

```
24.    if busiest.load  ⩽  local.load then
25.       continue
26.    end if
27.    busiest_cpu = pick busiest cpu of sg
28.    try to balance load between busiest_cpu and cur_cpu
29.    if load cannot be balanced then
30.       exclude busiest_cpu, goto line 20;  end if
31.    end for
```

VI. RESULT ANALYSIS

For the experimentation the scheduler is configured in desktop mode and Compute Intensive task is executed. The Linux kernel 4.4.0-rc2 [30] is installed on Ubuntu 15.10 which contains DVFS Governors. The available frequency for Intel Core i5 Processor ranges from 2.5 GHz to 0.8 GHz in the steps of 2.5 GHz, 2 GHz, 1.8GHz, 1.6GHz, 1.4GHz, 1.2GHz, 1GHz and 0.8 GHz.

Table 1 : Experimental Setup

Kernel Name	Linux
Linux Kernel Version	rc 4.4.2
Operating System	Ubuntu 15.10
Processor	Intel x86 i5 Processor
CPU Cores	Quad Core Processor
CPU Frequencies	2.50GHz,2GHz,1.8GHz, 1.6 GHz, 1.4 GHz, 1.2 GHz, 1 GHz, 0.8 GHz

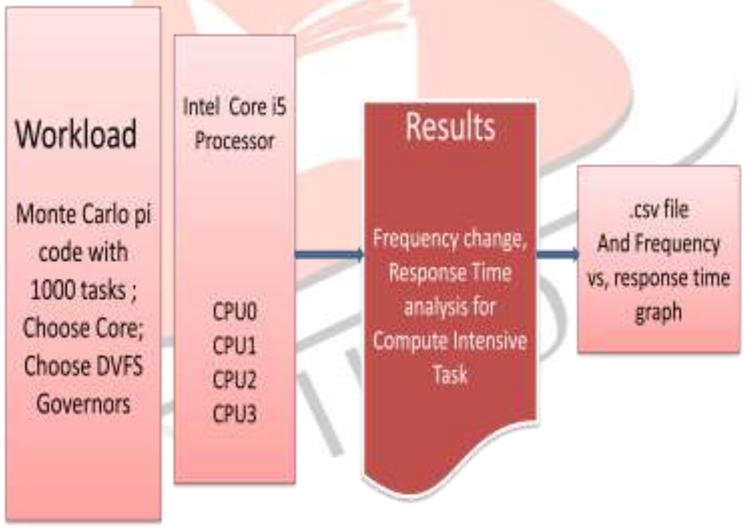
Workload
Monte Carlo pi code with 1000 tasks ; Choose Core; Choose DVFS Governors
Intel Core i5 Processor
CPU0
CPU1
CPU2
CPU3
Results
Frequency change, Response Time analysis for Compute Intensive Task
.csv file And Frequency vs, response time graph

Fig 3 : Benchmarking Methodology

Analysis of Frequency Change using Compute-Intensive Task

The proposed research work is carried out by implementing the Monte Carlo algorithm with calculation of Pi (π) value in multithreaded simulation as Compute Intensive task. The compute intensive task as Pi (π) calculation C program generally implemented by two algorithms, one is discrete integration method and another is Monte Carlo method.

C program for calculating pi value with Monte Carlo algorithm uses POSIX thread (Pthread) libraries which were available as part of GNU C compiler. It is multithreaded parallel compute intensive program to demonstrate the problem of unfairness and to evaluate the performance of scheduler driven DVFS scheme in multi-core environment. The program consists of Number of thread and number of iterations as the input parameters. This multithreaded program as compute-intensive task is utilized to get the better performance, and to prove the effectiveness of scheduler enabled DVFS scheme.

The compute-intensive task as pi (π) calculation c program with Monte Carlo simulation is chosen to get the desires result of Scheduler enabled DVFS scheme. It is concluded that [] the result will be precise for more number of thread and for more number of iterations. Hence, the numbers of threads are made very large to ensure that the program execution approximately takes few seconds for execution. The number of thread has huge impact on result as the experimentation used the time command to obtain system time as well as user time. As compute intensive task required more system time, in program user time is calculated to show the difference among user and system time.

The tests are produced under normal desktop environment with no other compute intensive task is running. The multithreaded pi calculation program is initiated through Schell script where the program is executed with the fixed priority setting to nice value 0. Schell script runs the program for large number of threads to get system time. First, the test is performed without the sched governor. Each simulation is done 1000 times by setting conservative, ondemand, powersave, performance and userspace for 4 cores, 3cores, 2cores and 1core. The frequency change is noted in time_in_state and trans_table .

The same results are produced by setting sched governor for all 4 cores to compare the response time and frequency change of scheduler enabled DVFS scheme for Compute-intensive task with other DVFS governor on DVFS loadable kernel. The CPU Frequency status provides the status of time_in_state table for each CPU core that is the total time spent by each CPU at the given frequency.

These results are shown in table-
Table 2. shows the average Response Time for all set of governors by executing Compute Intensive task 1000 times on DVFS loaded Linux kernel and proposed sched governor.

Table 2: Average Response Time for Compute Intensive Task

Governors ↓	**Avg. Response Time in sec.**
Conservative	0.32532
Ondemand	0.31899
Powersave	0.37137
Performance	0.3267
Userspace	0.46269
Sched	0.30326

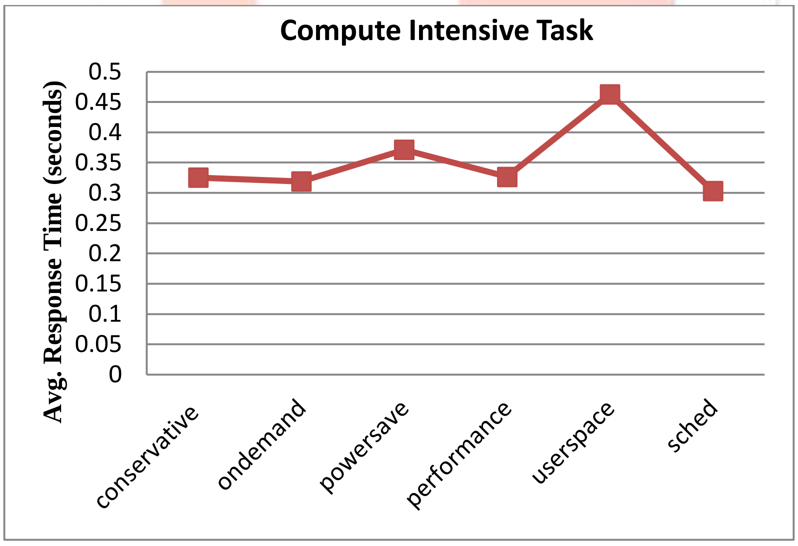

Fig 4 : Average Response Time taken by Compute intensive task for different governors

Fig. 4.represents the average response time taken by compute intensive task for all governors. Graphs show the comparison of all default governors with sched governor. The average system response time for sched governor is 0.30326 seconds which concludes that the proposed system with scheduler enabled frequency scaling scheme gives minimum Response time for compute intensive task as compared to existing governors.

During the whole experiment Hyper-threading is disabled, and results are analyzed for different cores. A Compute Intensive task is executed and the CPU Utilization is monitored. During the process, the time_in_state values at each frequency level are monitored.

The table 3 below shows the time_state values for quad core. The non-zero values in the last column indicate that there is a frequency switch in the patched kernel. Hence the performance and Scheduler governor are compared to prove the correctness of values.

Table 3 : time_in_state values for all set of governors with quad cores.

Number of Cores	**Quad Core**					
Frequency (KHz)	**Conservative**	**Ondemand**	**Performance**	**Powersave**	**Userspace**	**Sched**
2501000	193323	325808	412709	333036	333036	31649
2500000	2766	2786	2786	2786	2786	0
2000000	179308	209266	469140	450074	450074	4022
1800000	0	0	0	0	0	1
1600000	201659	201659	293736	289853	207922	2
1400000	0	0	0	0	0	1
1200000	0	0	0	0	0	1
1000000	0	0	0	0	0	9
800000	0	0	0	0	0	42

Table 4.below shows the average response time taken by compute intensive and according change in frequency for all governors. The frequency is taken as the maximum frequency with which particular governor works for the purpose of result analysis.

Table 4 : Avg. Response time and Maximum frequency taken by compute intensive task for all governors.

Governor	**Frequency**	**Avg. Response Time**
Conservative	193323	0.32532
Ondemand	325808	0.31899
Powersave	333036	0.37137
Userspace	333036	0.46269
Performance	412709	0.3267
Sched	31649	0.30326

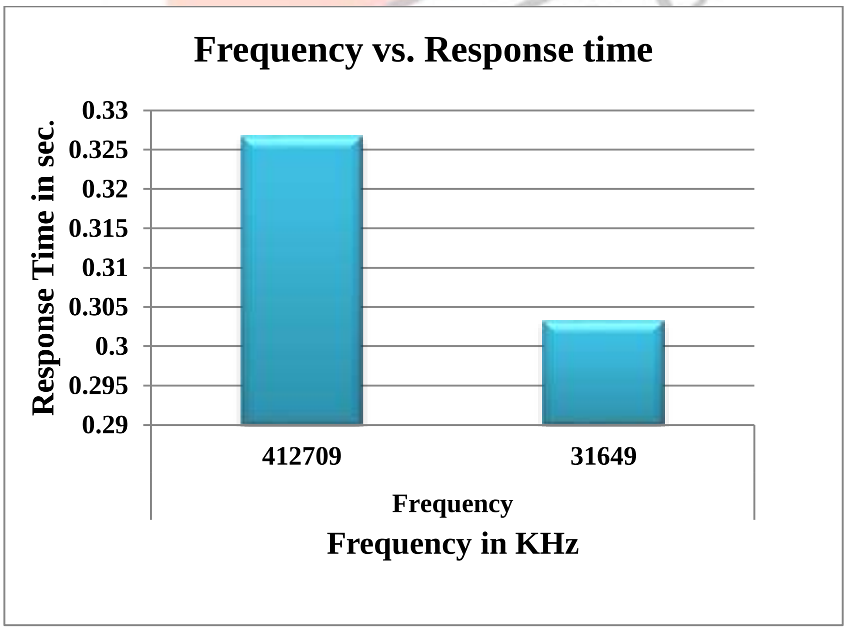

Fig 5 : Frequency vs. Response Time for compute Intensive task

Fig 5. shows the graph of frequency vs. Response Time for compute Intensive task. The proposed sched governor results are compared with performance governor to show that proposed system minimizes the Average Response time and accordingly frequency level. After executing compute intensive task Graph shows that performance governor runs with frequency 412709 KHz and takes average response time as 0.3267 seconds while the frequency at which sched governor runs is 31649 and average Response Time taken as 0.30326 seconds. Hence , comparison is made among performance and sched governor on the basis of frequency vs. Response Time graph which proves that the proposed sched governor takes minimum frequency level and average Response time for executing any Compute Intensive Task. So, proposed system ultimately enhances the performance of system.

VII. CONCLUSION

Completely Fair scheduler has disconnected design from frequency scaling algorithm, so CFS could not controls the CPU frequency. Proposed research work achieved the connection among CFS and Dynamic Frequency scaling scheme. The research work focuses on enhancing the performance of Linux system by analyzing frequency and Response time for scheduler-driven frequency scaling scheme with the help of Compute-intensive Task. It is proved that DVFS can be governed by scheduler related variables. Comparison is shown among existing governor and proposed governor- sched. Considering the hardware complexity and daemon processes of proposed experimental setup the compute intensive task is executed and the average response time is calculated. For sched governor the Avg. Response Time is 0.30326 seconds. By assuming that the userspace governor takes 100% Avg. Response Time it is calculated that sched governor takes 65.54% Response Time. Hence, it concludes that the proposed system with scheduler enables sched governor minimizes Avg. Response time by 34.46% as compared to existing governors.

Finally for various set of cores the frequency is analysed using time_in state table by executing compute-intensive task. Performance governor runs with frequency 412709 KHz and takes average response time as 0.3267 seconds for compute intensive task while the frequency at which sched governor runs is 31649 and average Response Time taken as 0.30326 seconds. Hence, comparison is made among performance and sched governor on the basis of frequency vs. Response Time graph which proves that the proposed sched governor minimizes frequency level by 7.66% as compared to existing frequency scaling scheme. Hence, by minimizing Avg, Response Time 34.46% and Frequency by 7.66 % it is proved that the proposed system with newly added 'sched' governor gives significant difference with existing governor and enhances the overall performance of system .

This is a preliminary work which forms the base for research in designing an operating system controlled Frequency scaling scheme which utilizes the CPU Frequency changes.Hence proposed system achieves the minimum Response time for scheduler driven frequency scaling scheme by executing Compute intensive task to enhance the overall performance of system.

REFERENCES

[1] J. Wei, E. Juarez, M. J. Garrido, and F. Pescador, “Maximizing the user experience with energy-based fair sharing in battery limited mobile systems,” IEEE Trans. Consum. Electron., vol. 59, no. 3, pp. 690–698, 2013.
[2] J. Wei, R. Ren, E. Juarez, and F. Pescador, “A linux implementation of the energy-based fair queuing scheduling algorithm for battery-limited mobile systems,” IEEE Trans. Consum. Electron., vol. 60, no. 2, pp. 267–275, 2014.
[3] C. S. Wong, R. D. Kumari, and J. W. Lam, “Fairness and Interactive Performance of O(1) and CFS Linux Kernel Schedulers,” no. 1, 2008.
[4] R. C. Garcia, J. M. Chung, S. W. Jo, T. Ha, and T. Kyong, “Response time performance estimation in smartphones applying dynamic voltage & frequency scaling and completely fair scheduler,” Proc. Int. Symp. Consum. Electron. ISCE, vol. 2, no. 2, pp. 1–2, 2014.
[5] C. S. Wong, I. Tan, and R. Deena, “Towards Achieving Fairness in the Linux Scheduler,” pp. 34–43.
[6] S. Wang, “Fairness and Interactivity of Three CPU Schedulers in Linux,” pp. 2–7, 2009.
[7] S. M. Mostafa, H. Amano, and S. Kusakabe, “FAIRNESS AND HIGH PERFORMANCE FOR TASKS IN GENERAL PURPOSE MULTICORE SYSTEMS,” vol. 29, no. December, pp. 74–86, 2016.
[8] J. Lozi, J. Funston, F. Gaud, V. Qu, and A. Fedorova, “The Linux Scheduler : a Decade of Wasted Cores.”
[9] “Completely Fair Scheduler _ Linux Journal.” http://www.linuxjournal.com/magazine/completely-fair-scheduler .
[10] “Tuning the Task Scheduler _ System Analysis and Tuning Guide _ openSUSE Leap 42.” https://doc.opensuse.org/documentation/leap/tuning/html/book.sle.tuning/cha.tuning.taskscheduler.html.
[11] G. Cheng, “A Comparison of Two Linux Schedulers,” Master thesis, pp. 1–89, 2012.
[12] Yigui Luo, Bolin Wu, “A Comparison on Interactivity of Three Linux Schedulers in Embedded System”, Communications, Computers and Signal Processing(PacRim), 2011 IEEE Pacific Rim Conference, pp. 494-498, August 2011.
[13] Wei-feng MA, WANG Jia-hai, “ Analysis of the Linux 2.6 Kernel Scheduler” 2010 IEEE International conference on computer Design and Applications, pp.71-74, 2010
[14] Prajakta Pawar, SS Dhotre, Suhas Patil, “CFS for Addressing CPU Resources in Multi-Core Processors with AA Tree”, International Journal of Computer Science and Information Technologies, Vol. 5 (1), 913-917, 2014
[15] Hong Xu, Rong Tang, “Study and Improvements for the Real-time Performance of Linux Kernel”, 3rd International Conference on Biomedical Engineering and Informatics, pp. 2766-2769 , 2010
[16] “Power Management & DVFS.” http://www.arteris.com/power-management-dvfs .
[17] R. Ge, R. Vogt, J. Majumder, and A. Alam, “Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU,2010”
[18] Poonam Karande, SS Dhotre, Suhas Patil, “Illustration of Task Scheduling in Heterogeneous Quad-Core Processors”, International Journal of Engineering and Technology Research, Vol 03, Issue 08, Pages:1389-1393, May 2014
[19] Dilipkumar, Vora Shivani, M. Tech, and S. S. Dhotre. "Runtime CPU Scheduler Customization Framework for Real Time Operating System."
[20] Kabugade, Rohan R., S. S. Dhotre, and S. H. Patil. "A Modified O (1) Algorithm for Real Time Task in Operating System."

[21] Wenbo Wu, Xinyu Yao, Wei Feng, Yong Chen, "Research on Improving Fairness of Linux Scheduler", Proceeding of the IEEE International Conference on Information and Automation, China, pp. 409-414,August 2013
[22] Jyotish J., O. Sujisha, T. Gilesh, Thayyil B., "On the Fairness of Linux O(1) scheduler, Fifth International Conference on Intelligent Systems, Modelling and Simulation, IEEE Computer Society, 2014.
[23] P. Tanaji, S. Dhotre, and R. Shankar, "A Survey on Fairness and Performance Analysis of Completely Fair Scheduler in Linux Kernel," vol. 9, no. 44, pp. 495–502, 2016.
[24] P. T. Patil and P. S. Dhotre, "Response Time Analysis Using Linux Completely Fair Scheduler for Compute-Intensive Tasks," vol. 5, no. 2, pp. 377–380, 2017.
[25] R. Shankar, S. Dhotre, and P. Tanaji, "A Survey on Response Time Analysis Using Linux Kernel Completely Fair Scheduler for Data Intensive Tasks," vol. 9, no. 44, pp. 351–358, 2016.
[26] R. S. Jamale, S. Dhotre, and S. H. Patil, "Data Intensive Task Analysis using Dynamic Voltage and Frequency Scaling Governors," vol. 5, no. 2, pp. 457–461, 2017.
[27] A. Silberschatz, P.B. Galvin, G. Gagne, "Operating System Concepts," 7th Edition, JohnWiley & Sons Inc.,2005
[28] Richard Petersen, "The Complete Reference" Linux, Second Edition, Tata McGraw Hill.
[29] Daniel P. Bovet & Marco Cesati". Understanding the Linux Kernel, OReilly October 2000.
[30] "Inside the Linux 2." https://www.ibm.com/developerworks/library/l-completely-fair-scheduler/.
[31] P. Tanaji, S. Dhotre, and R. Shankar, "A Survey on Fairness and Performance Analysis of Completely Fair Scheduler in Linux Kernel," vol. 9, no. 44, pp. 495–502, 2016.
[32] "sched scheduler-driven CPU frequency selection [LWN." https://lwn.net/Articles/667281/ .
[33] "Improvements in CPU frequency management [LWN." https://lwn.net/Articles/682391/.