Response Time Analysis using Linux Kernel Completely Fair Scheduler for Data Intensive Task

Mrs. Sunita Dhotre[1], Miss. Rucha Shankar Jamale[2], Dr. Suhas .H Patil[3]
[1]Associate Professor, Department of Computer Engineering, Bharati Vidyapeeth Deemed University, College of Engineering, Pune, India
[2]M.Tech, Department of Computer Engineering, Bharati Vidyapeeth Deemed University, College of Engineering, Pune, India
[3]Professor, Department of Computer Engineering, Bharati Vidyapeeth Deemed University, College of Engineering, Pune, India

Abstract*--The modern Linux-based systems with updated operating system are used over a huge range. They provide numerous facilities and features which eventually ease our day to day requirements. However, while providing these variant attributes Response Time of the system increases which affects the systems overall performance.*
This paper focuses on estimation and analysis of response time by designing scheduler driven DVFS scheme using a Data Intensive Task. The proposed research defined a solution with respect to Operating System track which is to invoke Dynamic Voltage & Frequency Scaling (DVFS) techniques in Linux scheduler Completely Fair Scheduler (CFS) in collaboration with frequency change which eventually improves response time and system overall performance.

Keywords--*Response Time, Completely Fair Scheduler (CFS), Dynamic Voltage and Frequency Scaling (DVFS), Governors, Sched.*

I. INTRODUCTION

Many supercomputers [1] have Linux as their primary operating system. Various open source communities use Linux as their principal operating system.Due to increasing Linux users, Linux kernels CPU schedulers are enhanced with better performance and effectivity.The Linux operating system[2] adapts different applications requirements such as multimedia, games, video and audio applications and the most important internet browsing. For modern Linux operating systems[28], the CPU frequency and software complexity keep on increasing which needs a high amount of energy. The experiences of the user are therefore significantly affected by the overall response time which is an unstable factor.
An appropriate balance between response time and change in frequency is obligatorily required for better performance and productivity. Overall there should be a proper balance between modern technology implementations and its performance. A failure to this may lead significant degrade in the quality of experience. The modern operating system executes several applications simultaneously; energy management has always remained a challenge. To address this problem, several conventional Power Management schemes[3] are developed to provide efficient battery lifetime by supervising the energy.

The authors R. C. Garcia, J. M. Chung[4] proposed a scheme for performance estimation in smartphone by invoking DVFS and CFS separately. DVFS scheme affects positively in Smartphone's response time performance because DVFS works at Central Processing Unit and change in operational frequency at CPU indirectly affects access speed and responsiveness for execution of task. CFS has a remarkable impact on execution of task at Kernel level.
The proposed scheme works as a response time estimator to analyze above effects under different load conditions. The proposed system works on Optimus G Smartphone [5]. The working here is mainly explained with respect to various load situations apparently, run in the background which helped to analyze Response Time Estimation and Response Time Performance. The two most important terms which helped the result computation are Instantaneous event unit and Time measurement unit. Hence the proposed system actually captured the variation in response time during a change in CPU frequency and applications running background.

J. Lozi, J. Funston, F. Gaud, V. Qu, and A. Fedorova explained in there paper[6] performance, bugs in Linux Scheduler are removed and fixed with the help of conservative testing techniques and performance debugging tools. Traditional testing techniques are ineffective at understanding small but serious kind of bugs as they are evasive.
Load balancing [29] concept is explained in this paper with Load balancing is the expensive technique which requires iterations of runqueues. Constant modifications of cache data structure, synchronization, and costly misses are carried out.This leads the utilization of scheduler at the maximum amount. Moreover, due to this some of the runqueues remains unbalanced. The cores become idle which eventually leads to bad performance.
To tackle this situation, the runqueues should be balanced in a smart way. To achieve this CFS is used for balancing the runqueue in a smarter way. CFS balances the runqueue on the load basis as well as weight basis.

The bugs in the scheduler lead the CPU core [7] to remain ideal where the threads will not face the problem of energy wastage. The authors proposed in the paper that the scheduling task of dividing the CPU cycles amongst the thread is an unsolved problem. The complexity of the above problem increases when more threads are executing simultaneously. The bugs into the Linux schedulers like waiting threads which leads to remain thread idle have a big impact on the runqueues and as a result of the runqueue get stuck when the CPU goes idle. The authors understand the problem and presented various tools who catch and fix the bugs at the same time. The authors identified four bugs in Linux Kernel CFS scheduler:-the Group Imbalance Bug, scheduling Group Construction Bug, the Overhead-on-Wakeup Bug, the Missing Scheduling Domain Bug.

The authors Sunwook Bae, Hokwon Song, Changwoo Min, Jeehong Kim, Young Ik Eom [8] established that for interactivity of processes I/O perfecting and process scheduling is applied when runtime overhead is observed at interactive process. By adding topmost flag set as false at booting time, the authors customized task_struct.
The observations and literature studied indicate that CFS is not connected with the CPU frequency change. CFS can also be linked to the DVFS algorithm as the response time enhances by a change in frequency. This leads to the design of a DVFS Scheme with a scheduler governor where the responsive time of processes and tasks are minimized.

In this paper [9], authors J. Wei, E. Juarez, M. J. Garrido, and F. Pescador implemented Energy based fair queuing (EFQ) scheduling algorithm. EFQ is used for maximizing the user experience in battery limited mobile systems. EFQ relies on energy oriented scheduling algorithms which support balanced energy usage and effective time restraint compliance. This paper shows how exactly EFQ is more flexible than Linux scheduler.
This article improves the working of EFQ and plays a vital role by maximizing the user experience in battery oriented mobile devices. The main work here deals with contributing traditional fair queuing algorithm regarding energy domain. The analysis is done by the help of test bench tool [30] which is created based on Linux scheduler to verify the proposed algorithm here. Due to this new testbench EFQ properties are analyzed appropriately with ease and no flaws. Also, EFQ scheduler here is compared with Linux default scheduler to show its advantage on enhancing user experience in battery limited mobile devices.
A fair queuing traditional algorithm is introduced in the energy domain. The relation between the acquired time of CPU and energy consumed is explained. Real time, Batch, and Interactive tasks are considered here. As per the operating system concepts, energy wastage is observed when no task is scheduled and the interactive process and real time process utilize only half CPU bandwidth. For overcome the energy problem author used DVFS scheme. All the results are carried out by using test bench benchmarking tools.

In this paper [10], J. Wei, R. Ren, E. Juarez, and F. Pescador explained the implementation of Energy base Fair Queuing (EFQ) Linux based scheduling algorithm. EFQ is an improvement over traditional fair queuing algorithm. The main characteristic of EFQ is proportional power share into the system.
This paper concentrates on improving the implementation of EFQ algorithm with the help of testbench Pthread in several ways.

- MiBench an open source benchmark suite is also used to program the task under test. Three tasks are programmed here interactive, batch and real time and these tasks are tested under EFQ scheduling algorithm.
- Hardware metering measures the power consumption of selected benchmarks and the obtained outputs in terms of energy values is given as an input to Pthread based testbench
- The total power consumption also includes energy used by I/O operations so that the overall systems power sharing ability can be achieved to some greater extent.
- The Linux Nice value table which maps the priority of the task wise in collaboration with its Kernels load weight; is redefined with the precise allocation of power share.

The algorithm provides a robust response time for different tasks. In addition to the existing SCHED_FIFO, SCHED_RR, SCHED_NORMAL, SCHED_OTHER a new scheduling policy is introduced. The sched_entity structure is modified to add the EFQ related variables viz. weight, share, packet size and warp parameters. For tracing the energy consumption energy measurement, related variables are added. The nice levels are mapped to the static global priority by adding difference value of 120. The authors have carried out the work in the Linux kernel scheduling files fair.c in the Linux Kernel Directory /kern/sched. The results are gathered using the performance tool MiBench.

III. LINUX KERNEL COMPLETELY FAIR

Completely Fair Scheduler(CFS) is the default scheduler of Linux Kernel. Ingo Molnar [32] introduced CFS in Linux Kernel 2.6.23. The key role of CFS is to eliminate the unfairness from the system by allocating a fair amount of CPU to each runnable process. Completely Fair Scheduler [11] deals with Ideal multi-tasking CPU which means CPU with 100% power and can execute each task at an equal speed, in parallel, each at 1/nr_running speed. The CFS [12] [13] tries to eliminate unfairness from the system. In a system, CFS keeps track of fair share of the CPU which is allocated to every process. Hence, CFS runs an equitable clock at a fraction of real CPU clock speed.

CFS is the default scheduler of Linux kernel[14]; recently all Android smartphone use CFS scheduler. The ultimate aim of Completely Fair Scheduler is to offer the fair amount to all the tasks directly proportional to their weights. In algorithm of CFS weight of every task is chosen by each tasks nice value, when the nice value of an individual task is decreased by one, then the weight of the task is raised by 1.25 times.

The CFS algorithm uses Red Black Tree[15], in this tree the tasks are arranged in a tree form from left to right according to the increasing order of respective nodes virtual run times. Meanwhile, CFS executes its task initiating from left most leaf moving towards the right.

In CFS the ideal, precise multi-tasking CPU means the CPU which runs multiple processes [16] concurrently by dividing the power of processor (Fair share of processing time) among all runnable processes. That means if a single process is running in the system then it will get 100% CPU's power; if there are two runnable processes, then each process will execute on 50% of processor's power in this way the multiple runnable processes can execute simultaneously by sharing the fair amount of CPU.

CFS uses timeslice and process priority for process scheduling. Timeslice is defined as the total time taken by a process to execute and run. The priority decided by the help of timeslice period. If the process has big timeslice, it is assigned with the highest priority. The nice value given to each process according to user's perspective determines the priority of the process.

The time proportion received by the processor is the differernce between process and runnable processes niceness.

Following are the scheduling policies supported by CFS:-

- SCHED_NORMAL /SCHED_OTHER: It is used for regular tasks.
- SCHED_FIFO: It uses First-In-First- Out Policy
- SCHED_BATCH: It is used for running the tasks for longer time without preempting
- SCHED_IDLE: Processes with low priority use this policy
- SCHED_RR : It is alike to SCHED_FIFO, but Round Robin scheduling algorithm is used in this policy.

In CFS, processes (tasks) are given fair processing time, when time for any task is out of balance as compared to other task, then those out of balance tasks should given the processing time to execute, in this way CFS maintains the Fairness . So, to determine the balance among multiple tasks CFS introduces the concept of virtual runtime (vruntime). Virtual runtime defines 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. CFS maintains fairness for waiting processes by the help of Red-Black tree which decides the runqueue processes order. CFS maintains the time order RBTree [33].

The RB tree is self-balancing binary search tree supporting following features:

- Each node is black or red.
- Each leaf node is black.
- If the node is red, then is means both the children of the node are black.
- Every simple path forms a node to leaf node contains the equal number of back nodes.

The benefits of using RBtree in CFS are :

- It is a self-balancing tree, which means that there is no path from the root to leaf node is more than twice as long as any other.
- For searching the RBTree takes O(log n) time.

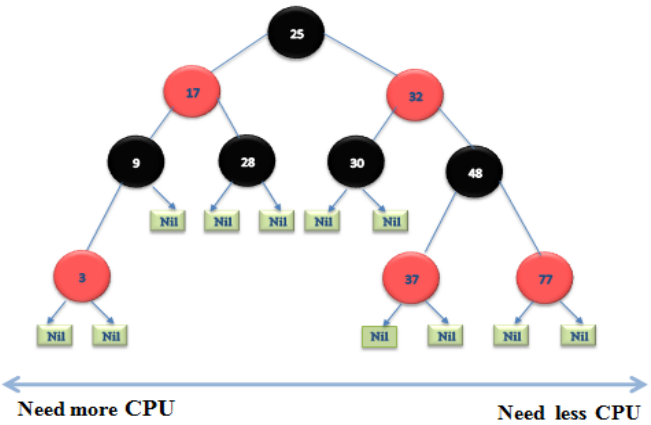

Fig 1: Red-black tree

The fig. 1 shows the Red-black tree, and each node in the tree is a certain task within the system, and virtual runtime is represented by the key value of particular task's node. According to the description of an RB tree, left most node has smallest key value, which eventually means that task with minimum key value has the highest priority with least virtual runtime and vice versa. Hence, CFS has to take left most tasks for processing, and once the task is processed, then it is permanently deleted from the RB tree.

IV. DYNAMIC VOLTAGE AND FREQUENCY SCALING

The power management scheme focuses on two aspects Dynamic Power Management (DPM) [17] and Dynamic Voltage and Frequency Scaling (DVFS) [18]. The DPM deals with executing the high workload at a maximum CPU speed while remaining workload at low power mode. The DVFS deals with executing processes at a low-performance setting regarding voltage and frequency.

DVFS techniques [19] are widely applied in smartphones to reduce power consumption by changing CPU core frequency and system voltage, and eventually, this result in variance in response time in smartphones while executing a precise application.

Many CPU Frequency Scaling Governors exist which allows the drives to set the target frequency. For the

efficient use of CPU dynamic frequency scaling mechanism is applied. These governors are embedded in patched Linux kernel System.

Fig 2 gives the overall idea of User level governors and Kernel level governors [29] [30] [34]. DVFS schemes include governors like Ondemand governor, Performance governor, Conservative Governor, Powersave Governor and Userspace governor.

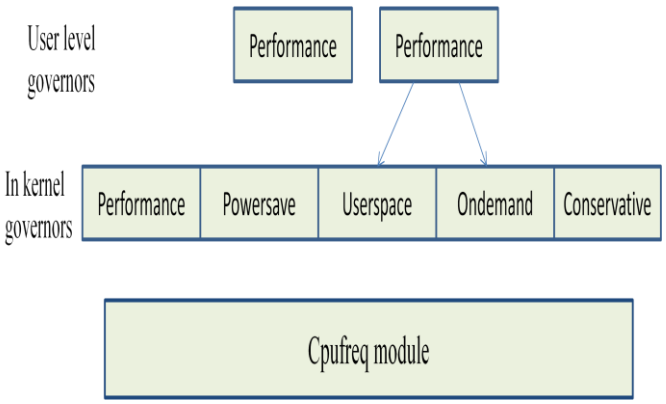

Fig 2: Governor Types

Ondemand governor [20] is the default governor of maximum Android-based smartphones. Ondemand governor was introduced in the Linux Kernel 2.6.10. Depending on the processor utilization it dynamically changes the processor frequency. The use of the processor is checked, and if the value exceeds the threshold, this governor set the frequency to the highest available value. If the utilization is less than the threshold, the governor steps down the frequency. The range of frequencies can be controlled by the governor and also the rate of checking the utilization of the system.

Performance governor sets the frequency to the highest frequency which is available. This allows the processor clock speed [21] to be set to maximum thus allowing maximum performance. No power savings are achieved which Performance Governor is used, but it allows changing the frequency.

In Conservative Governor Frequency is dynamically adjusted based on the processor utilization with a gradual increase in its value. The frequency of the processor utilization is checked and if its lies below or above the utilization thresholds, this governor steps up or down the frequency to the next available instead directly going to high or low.

Powersave Governor sets the processor to the lowest available frequency however a range of frequencies can be adjusted. The process runs at the slowest frequency. Therefore it takes the time to go idle.

In Userspace Governor Frequency is set manually in this governor. It does not dynamically change the frequency. Compare to all other governors Userspace is more customizable, it has a most efficient way for balancing between Performance and power of the system.

V. RESEARCH APPROACH

Previously many research are done with Energy efficiency, Energy contingent, Energy fidelity but considering Operating System domain to deploy battery constraints and performance are very scarce. Devices are getting smaller in size [22] with more amenities; hence it is crucial to maintaining a balance between battery capacity and different modern features. The power management schemes were introduced to challenge battery limitation, and they have more impact on memory, CPU, Network Bandwidth and Performance.

DVFS have its own set of different governors. Governors have a more controlled way for changing the CPU frequency.

The research aims at designing a scheduler enabled Dynamic Voltage & Frequency Scaling Scheme [23] [24]. The existing DVFS is invoked in the kernel module with the already existing governors.

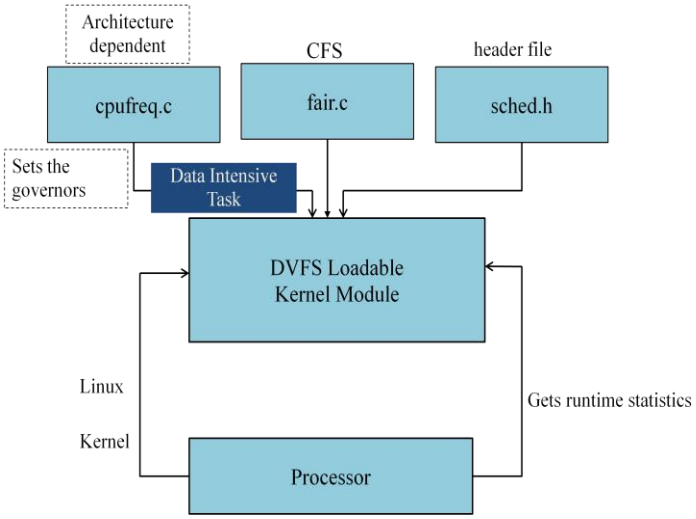

Fig 3: System Level Implementation

The main motive is to design a scheduler driven DVFS scheme. To achieve this, already existing DVFS techniques are loaded into Linux Kernel module as shown in Fig 3 This aspect helps to reduce extra power usage by setting lowest value for processors frequency. The modifications are done in the header file sched.h and cpufreq.c which adds a new governor. And by this method a new CFS enabled DVFS scheme is generated.

The proposed system focuses on estimation of response time analysis by designing scheduler driven DVFS scheme. Response Time Analysis of Linux Kernel Completely Fair Scheduler for Data Intensive Task is carried out by analysis of frequency change by the help of DVFS properties invoking in Linux kernel with the help of Data Intensive Task [25] [26].

To optimize the user experience the Completion time or Response time of a Process is the main focus of the work. For the given frequency limits the utility of CPU Scheduling Algorithm will be explored.

The new capacity of the CPU is generated at various points within CFS including Load Balance, and a call is to make finally to update the capacity of the CPU which then converts the new minimum capacity request into the CPU Frequency.

Frequency analysis is done by the help of Data-intensive task. Data-intensive tasks are used to describe applications that are I/O bound or with a need to process large volumes of data. This kind of claims most of their processing time to I/O and movement and manipulation of data. Data-intensive platforms use parallel computing approach combining multiple processors and disks to large computing clusters connected using high-speed communications switches and networks.

The response time analysis determines the schedulability of real-time systems on a fixed priority basis. The main objective of this study is to identify the points of interest with respect to frequency change within the Linux kernel for the response-time analysis.

Algorithm: Load Balancing

{ For each CPU cur_cpu} set curr->policy = SCHED_OTHER
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;
13: for the tasks that are new or waking up trigger the frequency switch
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 if (task is in sleep state) update capacity of (cpu(rq)) Raise the target cpu's Operating Point Frequency;
16: set the driver target frequency using cpu frequency table with new value
17: **end for**
18: **busiest = overloaded sg with the highest load**
(**or,** if inexistent) imbalanced sg with highest load
(**or,** if inexistent) sg with highest load
19: local = sg containing cur_cpu
20: **if** busiest.load ≤ local.load **then**
21: **continue**
22: **end if**
23: busiest_cpu = pick busiest cpu of sg
24: try to balance load between busiest_cpu and cur_cpu
25: **if** load cannot be balanced **then**
26: exclude busiest_cpu, goto line 19
27: **end if**
28: **end for**

The DVFS methods are invoked through CFS code in load balancing algorithm [6] [27] in Linux Kernel. For each scheduling domain (sd), the load balancing algorithm is executed. Only one core is balances the load, either its first core of scheduling domain or the first idle core whose free CPU cycles are used for load balancing technique. (Lines 2-9). Average scheduling load is calculated for every scheduling group (sg) of the scheduling domain given at (Line 10) which allows picking up of the busiest CPU based on heuristics. If the load of the busiest CPU is lower than local group's load, it is considered as balanced level (Line 20). Duplication of work is prevented by executing the load-balancing algorithm on the selected core for the given sd. Each core runs the load balancing algorithm based on a periodic clock tick. Few optimizations are in Linux kernel 2.6.21 Version onwards which avoids periodic waking up of sleeping cores. These cores enter a tickles idle state, who reduces the use of CPU Cycles.

Load balancing is an expensive procedure because it requires iterating over dozens of runqueues, and modification of remotely cached data structures, causing tremendously expensive cache misses and synchronization. As a result, the scheduler goes to great lengths to avoid executing the load-balancing procedure often. At the same time, not executing the processes may lead to unbalanced runqueues. When that happens, cores might become idle when there is work to do, which hurts performance.

To balance the runqueues smartly, accounting for the fact that the high priority thread does not need a whole core. To achieve this aim, CFS balances runqueues not only by weights but also on a metric called load, which is the combination of thread weight and average CPU utilization. If a thread does not use much of a CPU, its load will be decreased accordingly. Fig. 4 explains the Load Balancing technique which distributes its load among the four CPU cores. The fig shows Core 4 has maximum load hence it is the busiest CPU core.

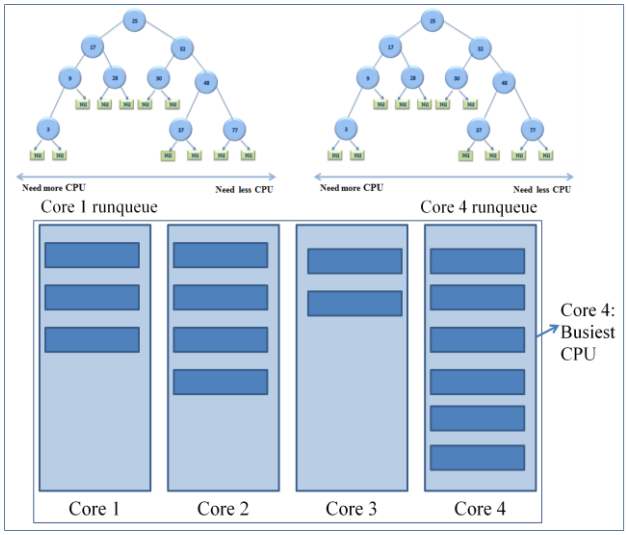

Fig 4: Load Balancing

VI. RESULTS

The practical work is carried out on Intel i5 processor including four cores. 4.4.0-rc2 Linux kernel is installed on Ubuntu 15.10.

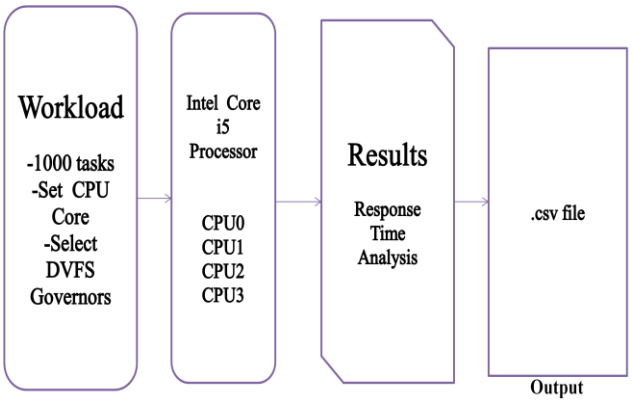

Fig 5: Benchmarking Methodology

Fig. 5 shows the benchmarking method used for a proposed system where the Data Intensive Task is executed for 1000 times with the help of shell script [31] by setting different governors and at the same time setting different CPU cores. Each governor has a different CPU utilization as well as diverse Time in state values. The Transition Table of all the governors also varies as the CPU cores get changed. The primary result proves that CFS utilizes and changes the CPU capacity which represents an average optimization in response time.

A. Analysis of Data Intensive Task

The Data Intensive Graph as shown in fig 6 proves that a Data Intensive Task is user oriented and varies according to respective governors. Average Response Time of the various governors is also reprented in the graph. The Graph also indicates that the patched governor gives minimum Response time. The other governors vary according to their independent behavior.

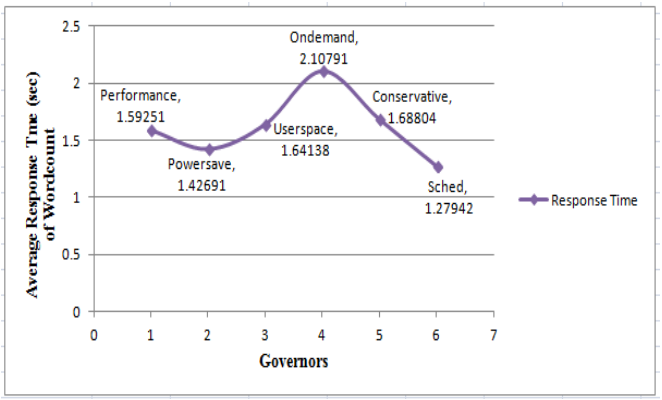

Fig 6: Data Intensive Task Graph

B. Comparison of Governors

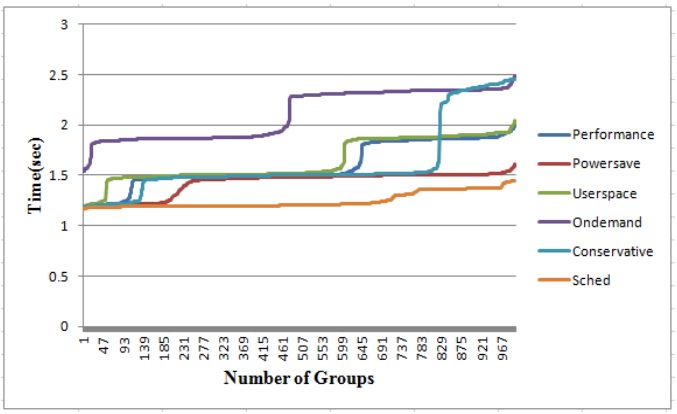

Fig 7: Comparison of Governors

The above fig. 7 analyze the in all behavior of DVFS governors and the Patched Sched governor. The analysis is carried out by executing the program by 1000 number of groups to visualize actual changes amongst the governors.

C. Analysis of Frequency and Response Time

In proposed system change in frequency results in a change in Response time. Performance parameter is greatly affected by the change in Response time. It eventually leads to a better performance. Table XI represents the Frequency and Response time of Performance and patched governor Sched. Table verifies that sched governor takes minimum frequency by 21.47% and minimum Response time by 80.33% compared to Performance governor. The exact difference between the response times of governors is 0.31309 sec and difference between frequencies is 1565775 kHz.

Governors	Frequency(khz)	Response Time(sec)
Performance	1993934	1.59251
Sched	428159	1.27942
Difference	1565775	0.31309

Table I: Frequency and Response Time of Performance and Sched governors

Fig. 8 depicts the Frequency vs. Response time graph. The two bars represent frequency and response time values of Performance and Sched governor respectively.

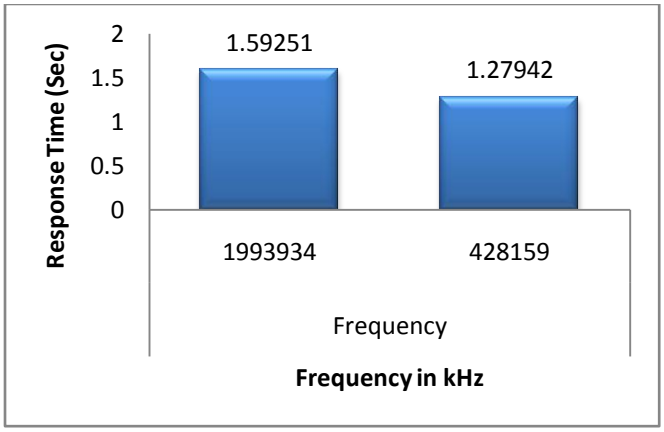

Fig. 8: Frequency vs. Response Time

VII. CONCLUSION

Modern Linux based devices have several advanced inbuilt features due to this, devices possess performance and battery limitation problem. The proposed research defined a solution with respect to Operating System track which is to invoke DVFS techniques in Linux scheduler CFS in collaboration with frequency change which eventually improves the performance and battery capacity. This aspect directly works on kernel level approach. This is a primary work which collaborates DVFS scheme with CPU frequency change

A new patched kernel is developed by adding a new patch to the CFS algorithm in load balancing session. The Sched governor is created here; the secondary analysis of the research involves deep analysis of Sched governor with respect to comparison of other five governors. For superior analysis graphical representation of Performance governor and sched governor is shown in the results. Considering the hardware complexity of Intel x86 i5 processor with Ubuntu 15.10 operating system as per experimental setup and excluding the daemon processes the average response time of sched governor is decreased by 60.69%; with respect to the highest computed average response time of Ondemand governor which is 100%.The Result Analysis proves that Sched consumes minimum response time compared to other governors which is 1.27942 sec and the average response time of sched governors compared to other governors is decreased by 60.69%.

VIII. FUTURE SCOPE

The proposed work contributes in implementing a DVFS driven scheme through CFS scheduler. This work can further be implemented on various hardware resources and processors. The Linux Schedulers can also be further developed by comprehensive literature, performance tuning and invoking new ideas to the architecture.

Response Time Analysis using Linux Kernel Completely Fair Scheduler for Data Intensive Task have various applications in wireless technology and modern operating system devices like smartphones, Gaming.

ACKNOWLEDGMENT

The authors would like to thank all the staff members of Bharati Vidyapeeth College of Engineering for their valuable inputs and support.

REFERENCES

[1] A. Gara *et al*., "Overview of the Blue Gene/L system architecture," in *IBM Journal of Research and Development*, vol. 49, no. 2.3, pp. 195-212, March 2005.

[2] Barabanov, Michael. *A linux-based real-time operating system*. Diss. New Mexico Institute of Mining and Technology, 1997.

[3] Le Sueur, Etienne, and Gernot Heiser. "Dynamic voltage and frequency scaling: The laws of diminishing returns." *Proceedings of the 2010 international conference on Power aware computing and systems*. 2010.

[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] F. Lin and W. Ye, "Operating System Battle in the Ecosystem of Smartphone Industry," *2009 International Symposium on Information Engineering and Electronic Commerce*, Ternopil, 2009, pp. 617-621.

[6] Lozi, Jean-Pierre, et al. "The Linux scheduler: a decade of wasted cores." *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016.

[7] Karande, Poonam, S. P. Dhotre, and Suhas Patil. "Task management for heterogeneous multi-core scheduling." *Int. J. Comput. Sci. Inf. Technol* 5.1 (2014): 636-639.

[8] Sunwook Bae, Hokwon Song, Changwoo Min, Jeehong Kim, Young Ik Eom, "Eimos: Enhancing Interactivity in Mobile Operating System", Springer, "12th International Conference on Computational Science and its Applications – ICCSA 2012 Salvador de Bahia, Brazil,pp. 238-247, June 2012

[9] 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.

[10] 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.

[11] C. S. Wong, I. K. T. Tan, R. D. Kumari, J. W. Lam, and W. Fun, "Fairness and interactive performance of O(1)

and CFS Linux kernel schedulers," *Proc. - Int. Symp. Inf. Technol. 2008, ITSim*, vol. 3, no. 1, 2008.
[12] Kabugade, Rohan R., S. S. Dhotre, and S. H. Patil. "A Modified O (1) Algorithm for Real Time Task in Operating System."
[13] Kabugade, Rohan R., S. S. Dhotre, and S. H. Patil. "A Study of Modified O (1) Algorithm for Real Time Task in Operating System." *Sinhgad Institute of Management and Computer Application NCI2TM* (2014).
[14] Pabla, Chandandeep Singh. "Completely fair scheduler." *Linux Journal* 2009.184 (2009).
[15] P. Pawar, S. S. Dhotre, and S. Patil, "CFS for Addressing CPU Resources in Multi-Core Processors with AA Tree," *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, no. 1, pp. 913–917, 2014.
[16] Kumar, Avinesh. "Multiprocessing with the completely fair scheduler." *IBM developerWorks* (2008).
[17] Le Sueur, Etienne, and Gernot Heiser. "Dynamic voltage and frequency scaling: The laws of diminishing returns." *Proceedings of the 2010 international conference on Power aware computing and systems*. 2010.
[18] Choi, Kihwan, Ramakrishna Soma, and Massoud Pedram. "Dynamic voltage and frequency scaling based on workload decomposition." *Proceedings of the 2004 international symposium on Low power electronics and design*. ACM, 2004.
[19] Dhiman, Gaurav, and Tajana Simunic Rosing. "Dynamic voltage frequency scaling for multi-tasking systems using online learning." *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*. IEEE, 2007.
[20] Pallipadi, Venkatesh, and Alexey Starikovskiy. "The ondemand governor." *Proceedings of the Linux Symposium*. Vol. 2. No. 00216. sn, 2006.
[21] Noble, James L., et al. "Adjusting clock frequency and voltage supplied to a processor in a computer system." U.S. Patent No. 5,760,636. 2 Jun. 1998.
[22] Cuervo, Eduardo, et al. "MAUI: making smartphones last longer with code offload." *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010.
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] Shirazi, Behrooz A., Krishna M. Kavi, and Ali R. Hurson. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Society Press, 1995.
[28] R. Love, "Linux Kernel Development," 2nd Edition, Noval Press, ISBN 0-672-32720-1, 2005.
[29] Silberschatz, Abraham, et al. *Operating system concepts*. Vol. 4. Reading: Addison-wesley, 1998.
[30] Ezolt, Phillip G. *Optimizing Linux (R) Performance: A Hands-On Guide to Linux (R) Performance Tools*. Prentice Hall PTR, 2005.
[31] Bovet, Daniel P., and Marco Cesati. *Understanding the Linux Kernel: from I/O ports to process management*. "O'Reilly Media, Inc.", 2005
[32] Elboth, David. *The Linux Book*. Prentice Hall PTR, 2001
[33] Sobell, Mark G. *A practical guide to Linux commands, editors, and shell programming*. Prentice Hall Professional Technical Reference, 2005.
[34] I.Molnar, "Modular Scheduler Core and Completely Fair Scheduler [CFS]," http://lwn.net/Articles/230501.
[35] Red-black trees, "http://www.eli.sdsu.edu/courses/fall95/cs660/notes/RedBlackTree/RedBlack.html#RFToC1" October 1995.
[36] D.Brodowski, "CPUFreq Governors," https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt, Nov. 2013.