Energy-Efficient Real-Time Scheduling in Sensor Networks via DVFS on Raspberry Pi

Nihaal Zaheer
nzaheer@iastate.edu
CprE 558 Section 2

Varun Advani
vsadvani@iastate.edu
CprE 558 Section 2

Abstract— **This project explores the dynamics of energy-efficient real-time scheduling in sensor networks, focusing on Dynamic Voltage and Frequency Scaling (DVFS) strategies implemented on a Raspberry Pi 3B+. The primary objective is to investigate the trade-offs between energy efficiency and system performance under varying workloads and CPU governor settings. We utilized a set of distinct workloads, including sensor-based tasks, video playback, algorithmic computations (Dijkstra's Algorithm and Matrix Multiplication), to emulate real-time processing scenarios.**
The study extensively analyzed the impact of different CPU governors - powersave, ondemand, conservative, userspace, and performance - on key metrics such as CPU Load, Average Frequency, and Latency. Our methodology involved recording these metrics under each workload-governor combination, followed by an in-depth comparative analysis. We also introduced an estimated energy consumption model, alongside a novel Performance Efficiency Index, to evaluate the governors' efficacy in balancing energy use with performance.
Our results indicate significant variations in energy consumption and performance across the governors, with the 'powersave' governor demonstrating the highest energy efficiency at the potential expense of increased latency. Conversely, the 'performance' governor excelled in minimizing latency but with higher energy demands. The 'ondemand' governor presented a balanced profile, adapting dynamically to workload changes. These insights are crucial for optimizing real-time systems in sensor networks, providing a framework for selecting appropriate DVFS strategies based on specific performance or energy efficiency requirements.

I. Introduction

In the realm of sensor networks, particularly those deployed in real-time applications, energy efficiency and performance are critical factors. These networks often operate in environments where power resources are limited, making effective energy management essential. This study investigates the use of Dynamic Voltage and Frequency Scaling (DVFS) on a Raspberry Pi 3B+ as a method to optimize energy consumption in such networks without compromising their real-time performance capabilities.

The primary challenge addressed in this project is the inherent trade-off between energy efficiency and computational performance. Real-time systems, especially those reliant on sensor data, require prompt and reliable data processing, which typically demands high computational power and, consequently, higher energy consumption. DVFS provides a solution by adjusting the processor's voltage and frequency in response to the workload, potentially reducing energy use while maintaining performance.

Our investigation centers on various CPU governors, namely powersave, ondemand, conservative, userspace, and performance, which are essentially different DVFS algorithms within the Raspberry Pi environment. These governors dictate how the CPU's frequency is modulated, impacting both energy usage and computational efficiency. We simulate a range of workloads, including sensor data processing, video playback, and complex computational tasks like Dijkstra's Algorithm and Matrix Multiplication, to evaluate the effectiveness of each governor.

II. Project Objectives and Scope

The primary goal of this project is to explore and optimize energy efficiency in real-time systems, specifically focusing on sensor networks operating on the Raspberry Pi 3B+ platform. The study aims to investigate the balance between energy consumption and performance efficiency under varying workloads, utilizing Dynamic Voltage and Frequency Scaling (DVFS) via different CPU governors.

A. System Model

The problem addressed in this study is commonly found in the Internet of Things (IoT) and embedded systems, where energy efficiency is crucial for the longevity and reliability of devices. These systems often comprise sensor networks that constantly gather and process data. The Raspberry Pi 3B+, a typical representative of such systems, is used as the primary model for our experiments. In these environments, managing the power consumption of the CPU while ensuring adequate processing capabilities is a critical challenge. Our study simulates real-world scenarios where a single device (the Raspberry Pi) is responsible for both data acquisition from sensors and processing tasks, without the involvement of external cloud servers.

B. Problem Statement

The core issue addressed is the efficient management of CPU resources in terms of power consumption and processing capability. Specifically, the project seeks to determine the most effective CPU governor for various types of workloads in a real-time sensor network environment, balancing the need for prompt data processing with the constraint of limited energy resources.

C. Objectives and Scope

The objectives of this project are as follows:

- To measure and analyze the impact of different CPU governors (powersave, ondemand, conservative, userspace, and performance) on CPU Load, Average Frequency, and Latency under various workloads.
- To develop a model for estimating energy consumption and create a Performance Efficiency Index, providing a quantitative framework for evaluating governor efficiency.

- To provide insights and recommendations on governor selection for real-time systems based on specific performance and energy efficiency requirements.

The scope of this study is confined to the Raspberry Pi 3B+ platform, utilizing a predefined set of workloads (sensor data processing, video playback, and computational algorithms) to simulate real-time processing scenarios. The findings are intended to inform the design and optimization of similar real-time systems in sensor networks and IoT environments.

III. Solution Methodology

The approach involves empirically evaluating the system's response under various workloads using specific metrics: CPU Load, Average Frequency, and Latency. This empirical data is supplemented by theoretical modeling to estimate energy consumption and to develop a Performance Efficiency Index. These models are crucial for quantifying and comparing the energy-performance trade-offs associated with each governor.

A. Algorithms, Protocols, and Architectures

The study investigates five CPU governors, each representing a unique algorithm for Dynamic Voltage and Frequency Scaling (DVFS) within the Linux kernel of the Raspberry Pi 3B+.

1) CPU Governors: Five CPU governors are analyzed, each employing a specific DVFS strategy on the Raspberry Pi 3B+:

Powersave Governor:

```
powersave():
    set_freq(min_freq)
```

Prioritizes minimal power usage by maintaining the lowest CPU frequency.

Ondemand Governor:

```
ondemand(current_load):
    freq = (current_load > threshold) ?
           max_freq : adjust_freq(curr_load)
    set_freq(freq)
```

Adjusts CPU frequency based on system load, balancing power and performance.

Conservative Governor:

```
conservative(current_load):
    freq = (current_load > up_threshold) ?
           increase_freq() : decrease_freq()
    set_freq(freq)
```

Modulates frequency in smaller increments for gradual changes.

Userspace Governor:

```
userspace(user_freq):
    set_freq(user_freq)
```

Allows manual control over the CPU frequency.

Performance Governor:

```
performance():
    set_freq(max_freq)
```

Maximizes performance by setting the CPU to its highest frequency.

2) Dijkstra's Algorithm Implementation: Dijkstra's Algorithm is used to model a computationally intensive task. It involves finding the shortest path in a graph, representative of complex algorithmic processing in real-world applications.

Pseudo-code for Dijkstra's Algorithm:

```
Dijkstra(Graph, source):
   create vertex set Q
   for each vertex v in Graph:
       dist[v] ← INFINITY
       add v to Q
   dist[source] ← 0

   while Q is not empty:
       u ← vertex in Q with min dist[u]
       remove u from Q

       for each neighbor v of u:
           alt ← dist[u] + length(u, v)
           if alt < dist[v]:
               dist[v] ← alt
   return dist[]
```

3) Matrix Multiplication: Matrix multiplication is used as another CPU-intensive workload. The operation is fundamental in many high-performance computing applications.

Pseudo-code for Matrix Multiplication:

```
MatrixMultiply(A, B):
   let C be a new matrix of size AxB
   for i from 1 to rows(A):
       for j from 1 to columns(B):
           C[i,j] ← 0
           for k from 1 to columns(A):
               C[i,j] += A[i,k] * B[k,j]
   return C
```

4) Energy Consumption Model: The energy consumption is estimated using a refined model:

$$E = \beta \cdot \text{Load} \cdot (\text{Freq})^2 + \gamma \cdot \text{Load} \tag{1}$$

Here, E is the energy consumption, β and γ represent the energy cost coefficients, and Freq is the CPU frequency.

5) Performance Efficiency Index: The Performance Efficiency Index (PEI) is defined to assess the trade-off between energy efficiency and speed:

$$\text{PEI} = \frac{1}{\text{Latency} \times E} \tag{2}$$

PEI inversely correlates with the product of task Latency and energy consumption E.

6) System Architecture: The system includes:

- A Raspberry Pi 3B+ as the main computational unit.
- Sensor modules for data collection.
- Various computational tasks to simulate different operational scenarios.

B. Illustrative Example

Task Set Description:

Task	**Period (ms)**	**WCET (ms)**
Task 1 (Sensor Data Processing)	$T_1 = 100$	$C_1 = 30$
Task 2 (Computational Task)	$T_2 = 200$	$C_2 = 80$

TABLE I: Description of Task Set

Schedulability Analysis using Rate-Monotonic Scheduling (RMS):

- The CPU utilization factor U for the task set is calculated as follows:
$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{30}{100} + \frac{80}{200} = 0.3 + 0.4 = 0.7 \tag{3}$$
- According to RMS theory, for two tasks, the system is schedulable if $U \leq 2 * (\sqrt{2} - 1) \approx 0.828$, therefore

the condition is satisfied for the given task set because $U = 0.7 \leq 0.828$, assuming ideal conditions where task switches and governor switching overheads are not considered.

Governor-Specific WCET Adjustments and Utilization Factors:

Powersave Governor:

- Frequency reduction factor: Approximately $\frac{600}{1400} \approx 0.43$ (assuming 600 MHz operation).
- Adjusted WCETs: $C_{1,\text{powersave}} = \frac{30}{0.43} \approx 69.77$ ms, $C_{2,\text{powersave}} = \frac{80}{0.43} \approx 186.05$ ms.
- Utilization Factor:

$$U_{\text{powersave}} = \frac{69.77}{100} + \frac{186.05}{200} \approx 1.628 \tag{4}$$

Performance Governor ($F_{max} = 1.4GHz$):

- Utilization Factor (using original WCETs):

$$U_{\text{performance}} = \frac{30}{100} + \frac{80}{200} = 0.7 \tag{5}$$

Ondemand and Conservative Governors (Assuming average frequency):

- Average frequency factor: Approximately $\frac{1000}{1400} \approx 0.71$.
- Adjusted WCETs: $C_{1,\text{ondemand}} = \frac{30}{0.71} \approx 42.25$ ms, $C_{2,\text{ondemand}} = \frac{80}{0.71} \approx 112.68$ ms.
- Utilization Factor:

$$U_{\text{ondemand}} = \frac{42.25}{100} + \frac{112.68}{200} \approx 0.985 \tag{6}$$

Tabulated Results:

Governor	Utilization	Schedulability
Powersave	1.628	Not Schedulable
Ondemand	0.985	Marginally Schedulable
Conservative	0.985	Marginally Schedulable
Performance	0.7	Schedulable

TABLE II: Schedulability and CPU Utilization under Different Governors

Analysis:

- The 'Performance' governor, operating at the highest frequency, offers the best schedulability with a utilization factor significantly below 1.
- The 'Powersave' governor, with a much higher utilization factor, indicates that the system is not schedulable, as it cannot complete all tasks within their time constraints.
- 'Ondemand' and 'Conservative' governors, with utilization factors close to 1, are marginally schedulable, meaning the system is just capable of handling the tasks under typical conditions, and can be scheduled under EDF but not under RMS without overload.

The above illustrative example highlights the significant impact of governor selection on real-time system performance in IoT environments. It demonstrates the critical balance between energy efficiency, task completion times, and schedulability, guiding the selection of an appropriate governor based on specific operational requirements and constraints.

IV. Implementation and Simulation Architecture

The setup aims to precisely evaluate the performance of different CPU governors under a range of workloads on a Raspberry Pi 3B+.

A. Hardware and Software Infrastructure

Hardware Setup:

- *Raspberry Pi 3B+*: Chosen for its 1.4GHz 64-bit A53(ARMv8) quad-core processor and ability to support various CPU governors, crucial for DVFS analysis.
- *Sensors*: Integration of DHT11 (for temperature and humidity), sound sensors, and PIR motion sensors to simulate real-time data acquisition tasks.

Software Environment:

- *Operating System*: Utilization of Raspbian OS, providing a stable and customizable environment for CPU governor manipulation and performance monitoring.
- *Governor Configuration*: Employing Linux kernel's CPU governor interface for switching between different governors, crucial for our comparative study.

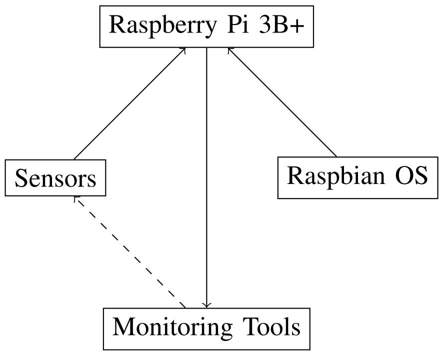


Fig. 1: Basic System Architecture

B. Simulation Tools and Workload Generation

Computational and Sensor Workloads:

- Implementation of a mixed workload environment, combining both sensor data processing and computationally intensive tasks to evaluate CPU governor performance under varied conditions.
- *Sensor Data Handling*: Custom Python scripts to simulate continuous sensor data reading and processing, mimicking typical IoT workloads.

Computational Tasks:

- *Video Playback*: Use of VLC Media Player for video-based tasks, providing a consistent and measurable workload for CPU performance evaluation.
- *Algorithmic Computation*: Implementation of Dijkstra's algorithm and matrix multiplication in Python to create CPU-intensive tasks, representing complex computational requirements.

C. System Monitoring and Performance Metrics

Real-Time Monitoring Tools:

- *psutil Library*: Utilized for real-time monitoring of system metrics, particularly CPU load, to assess the impact of governor changes.
- *Custom Python Scripts*: Developed to accurately measure and log system latency and average CPU frequency, capturing the nuanced performance differences under each governor.

Data Collection and Analysis:

- Performance data is collected and stored in a structured format for subsequent analysis.
- Python-based tools (including NumPy and Pandas) are employed for statistical analysis and visualization of the collected data, facilitating a detailed evaluation of each governor's performance.

V. Evaluation

A. Testing Methodology

Test Setup and Types:

- *Hardware Setup*: Raspberry Pi 3B+ integrated with DHT11, digital sound sensor, and PIR motion sensor.

- *Software Environment*: Raspbian OS with Python for scripting, ‘psutil‘ for system monitoring, and Linux tools for governor control.

Unit Tests:
- *Sensor Scripts*: Validated the accuracy and reliability of sensor data collection scripts.
- *Monitoring Scripts*: Tested the functionality of scripts used for recording CPU load, frequency, and latency.

Integration Tests:
- *System Integration*: Ensured seamless operation between the Raspberry Pi, sensors, and software components.
- *Governor Switching*: Verified the effective switching of CPU governors and the corresponding system response.

Real Tests:
- Executed a series of real-world task scenarios under each governor setting, encompassing both sensor processing and computational tasks.
- Recorded performance metrics for each scenario to evaluate the impact of governor choice on system behavior.

B. Detailed Procedure for Reproducibility

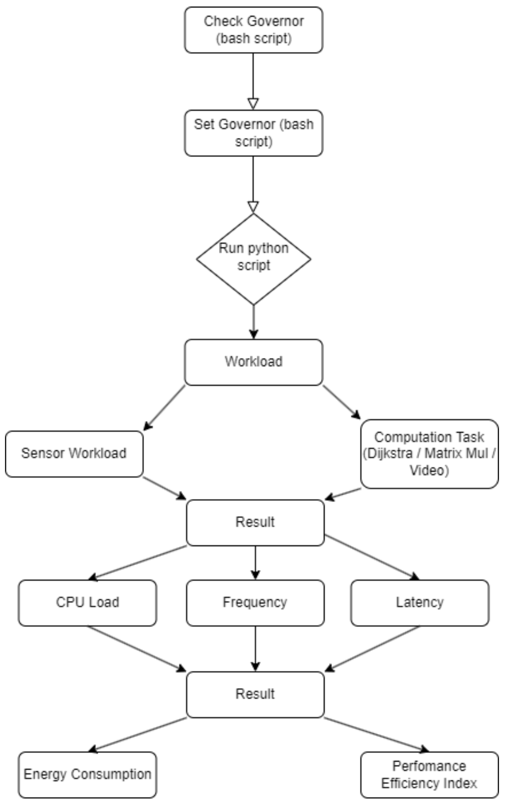

Check Governor (bash script)
Set Governor (bash script)
Run python script
Workload
Sensor Workload
Computation Task (Dijkstra / Matrix Mul / Video)
Result
CPU Load
Frequency
Latency
Result
Energy Consumption
Perfomance Efficiency Index

Fig. 2: Energy Consumption across Different Governors.

Hardware Configuration:
1) Connect the DHT11, sound, and PIR sensors to the Raspberry Pi GPIO pins.
2) Ensure stable power supply and internet connectivity for software updates and monitoring.

Software Setup:
1) Install Raspbian OS on the Raspberry Pi.
2) Set up Python environment, install ‘psutil‘, and other required libraries.
3) Deploy sensor data collection and performance monitoring scripts.
4) Configure Linux CPU governor control tools.

Execution of Tests:
1) Initiate sensor data collection scripts to simulate continuous data processing.
2) Run computational tasks (e.g., video playback, Dijkstra’s algorithm).
3) Sequentially switch between different CPU governors, maintaining consistent test conditions.
4) Collect and log data for CPU load, frequency, latency, and calculate PEI and energy consumption post-test.

Data Analysis:
1) Use Python scripts to aggregate and analyze the collected data.
2) Apply statistical methods to interpret the performance metrics and their implications on CPU governor efficiency.

C. Data Visualization and Analysis

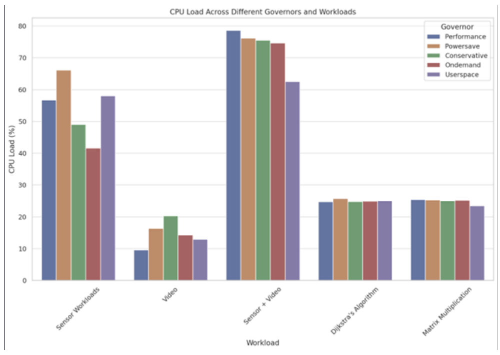


Fig. 3: CPU Load across Different Governors.

1) CPU Load and Average Frequency across Different Governors: Based on the data, we observe significant variations in CPU load among the different governors and workloads. The ‘Powersave‘ governor exhibits the highest average CPU load at 66.13% for sensor workloads, while for more intensive tasks like matrix multiplication, it maintains a CPU load of 25.28%. This suggests that while the ‘Powersave‘ governor reduces frequency to 600 MHz to conserve energy, it results in higher CPU utilization, which could lead to longer task completion times.

The ‘Performance‘ governor, with a consistent frequency of 1400 MHz, showcases an interesting pattern: it has a relatively high CPU load for sensor workloads at 56.66%, yet it handles intensive computational tasks like Dijkstra’s algorithm with a lower CPU load of 24.74%. This governor optimizes for speed and appears to be more efficient when the system is under a heavy computational workload.

Adaptive governors, ‘Ondemand‘ and ‘Conservative‘, show their ability to balance load and performance. Under sensor workloads, ‘Ondemand‘ lowers the CPU load to 41.63% with an average frequency of 1397 MHz, suggesting a more aggressive response to load changes. ‘Conservative‘ has a similar load of 49.07% but maintains the maximum frequency of 1400 MHz, indicating that it may not downscale the frequency as efficiently during lighter tasks.

‘Userspace‘, with manual frequency settings, shows a load of 58.03% for sensor workloads at 1400 MHz, and interestingly, a load of 12.95% for video playback at 800 MHz, which illustrates the impact of tailored frequency settings on CPU load management.

2) Latency across Different Governors and Workloads: The latency characteristics for each governor across diverse workloads reveal critical insights into their performance. The ‘Performance‘ governor demonstrates consistently low latency across all tasks, with a mean latency of 10.054 seconds for sensor workloads and only a slight increase to

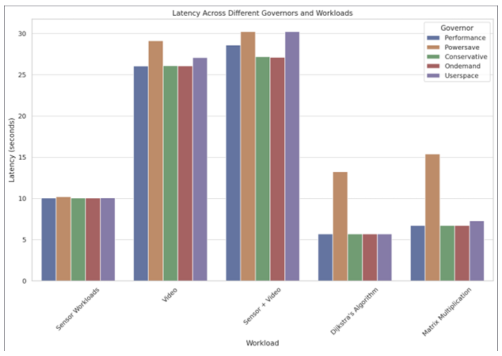


Fig. 4: Latency across Different Governors and Workloads.

26.071 seconds for video playback. These values underscore the governor's design to prioritize swift task execution.

Conversely, the ‘Powersave‘ governor shows significantly higher latency, particularly notable in computationally intensive tasks such as Dijkstra's algorithm and matrix multiplication, with mean latencies of 13.2523 second and 15.3913 seconds, respectively. This indicates a considerable performance compromise for energy savings, as the reduced frequency leads to longer task completion times.

Adaptive governors like ‘Ondemand‘ and ‘Conservative‘ offer an interesting balance, with ‘Ondemand‘ slightly outperforming ‘Conservative‘ in most tasks. For example, under ‘Ondemand‘, the latency for sensor workloads is a mean of 10.046 seconds, marginally lower than ‘Conservative‘'s 10.0633 seconds, suggesting a more efficient frequency scaling in response to workload demands.

The ‘Userspace‘ governor, which is subject to manual frequency control, shows latencies that are competitive with the ‘Performance‘ governor for lighter workloads but exhibit a notable increase to 27.100 seconds for video tasks when set to sub-optimal frequencies.

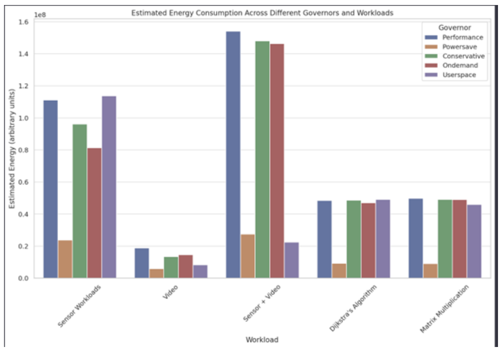


Fig. 5: Energy Consumption across Different Governors.

3) Energy Consumption across Different Governors: The estimated energy consumption, as illustrated in the plot, varies notably among the CPU governors and workloads. For instance, the ‘Performance‘ governor, which operates at the maximum frequency of 1400 MHz, exhibits the highest energy consumption across all workloads. Specifically, for sensor workload + video playback, it reaches approximately 1.54×10^8 energy units, reflecting its design priority for maximum throughput at the cost of power efficiency.

On the other end of the spectrum, the ‘Powersave‘ governor significantly reduces energy consumption, with sensor workloads consuming as low as 2.38×10^7 energy units. However, for computationally intensive tasks such as Dijkstra's algorithm, it shows an increase up to 9.26×10^7 energy units, which is still considerably less than what's observed under the ‘Performance‘ governor.

The ‘Ondemand‘ governor demonstrates adaptive energy consumption, dynamically adjusting based on the workload. For example, the energy consumption for sensor workloads is around 8.12×10^7 energy units, positioning itself as an intermediate between the ‘Performance‘ and ‘Powersave‘ governors.

The ‘Conservative‘ governor showcases similar adaptability but tends to consume slightly more energy than ‘Ondemand‘, indicative of its cautious approach to increasing frequency. For instance, in matrix multiplication tasks, it consumes roughly 4.92×10^7 energy units, marginally higher compared to ‘Ondemand‘.

The ‘Userspace‘ governor, with customizable settings, shows varied energy consumption patterns, largely dependent on the specific frequencies set by the user. Its energy consumption for video playback tasks is about 8.29×10^7 energy units, which suggests a middle ground approach, not fully optimized for power saving nor performance.

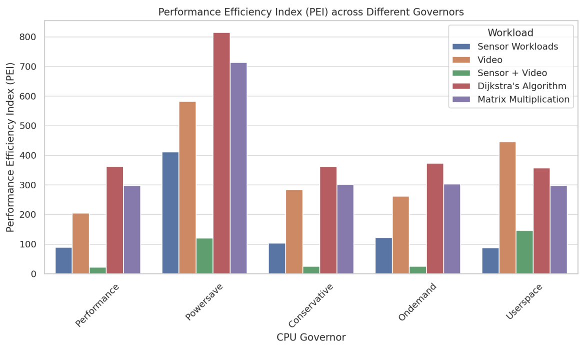


Fig. 6: Performance Efficiency Index (PEI) across Different Governors.

4) Performance Efficiency Index (PEI) across Different Governors: The PEI plot demonstrates a varied efficiency landscape across the spectrum of governors. The ‘Performance‘ governor, designed for speed, shows lower PEI values across computationally intensive tasks such as Dijkstra's Algorithm and Matrix Multiplication. This is indicative of the trade-off between quick task execution and energy consumption.

On the other end, the ‘Powersave‘ governor exhibits higher PEI values, especially in less demanding workloads like Sensor Workloads, suggesting that tasks can be completed with substantially lower energy consumption, albeit potentially at slower speeds. This aligns with the governor's goal to conserve energy, which may suit applications where time efficiency is less critical.

The ‘Ondemand‘ governor displays intermediate PEI values, reflecting its dynamic adjustment to CPU load and latency. For workloads that are sporadic or unpredictable, such as Sensor + Video, ‘Ondemand‘ offers a balance between responsiveness and energy use, optimizing performance as workload demands increase.

The ‘Conservative‘ governor typically trails the ‘Ondemand‘ in terms of PEI, suggesting a slightly less efficient response to workload changes. Its cautious frequency scaling can lead to increased task completion times, which might not be ideal for time-sensitive applications but could benefit scenarios where gradual changes in demand are expected.

Lastly, the ‘Userspace‘ governor showcases the impact of user-defined settings, with PEI values varying widely depending on how the frequency is managed for each task. When optimally configured, ‘Userspace‘ can achieve high performance efficiency, as seen in the Sensor Workloads.

VI. Conclusions

This study involves a comprehensive analysis of the effects of various CPU governors on the performance and energy consumption of a Raspberry Pi 3B+ under different workload conditions. The experiment leveraged the inherent capabilities of Dynamic Voltage and Frequency Scaling (DVFS) to adjust the CPU's operational parameters, aiming to discover an optimal balance between performance efficiency and power consumption.

Our contributions to this field of research include:

- A dataset capturing CPU Load, Average Frequency, and Latency across five distinct CPU governors under various workloads.
- A novel estimation of energy consumption and Performance Efficiency Index (PEI) that takes into account both the operational frequency and the CPU load.
- A systematic comparison of the governors' performance, providing a nuanced understanding of their behavior in real-time systems.

Findings from the experiment indicate that:

- The 'Performance' governor, while ensuring the lowest latency, resulted in the highest energy consumption across all workloads.
- The 'Powersave' governor significantly improved energy efficiency but at the expense of increased latency, which may not be suitable for time-sensitive tasks.
- Adaptive governors ('Ondemand' and 'Conservative') offered a middle ground, dynamically scaling frequency based on workload demands, hence providing a balanced approach.
- The 'Userspace' governor, with manual frequency settings, allowed for tailored performance tuning, indicating its potential for workload-specific optimization.

From these findings, we recommend:

- Further empirical studies to determine the exact power characteristics of CPU governors, which would enable more accurate energy consumption models.
- Development of advanced governor algorithms that can learn from workload patterns and predictively adjust frequencies for improved efficiency.
- Exploration of the impact of different governors in multi-core environments, as well as their effects on the thermal performance of the system.

Future experiments could focus on the integration of machine learning techniques to create smart governors that dynamically adapt to the system's workload patterns, potentially enhancing energy efficiency without compromising on performance. Additionally, considering the evolving landscape of IoT devices, further research could also explore the governors' behavior in a wider array of hardware configurations and operational environments.

VII. References

[1] Andrew B. Kahng et al. "Enhancing the Efficiency of Energy-Constrained DVFS Designs". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21.10 (2013), pp. 1769–1782. DOI: 10.1109/TVLSI.2012.2219084.

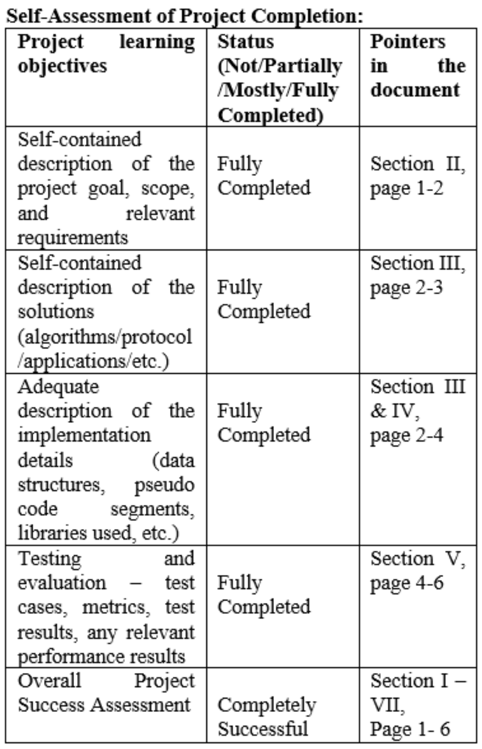
[2] Jabran Khan, Sebastien Bilavarn, and Cécile Belleudy. "Energy analysis of a DVFS based power strategy on ARM platforms". In: (June 2012). DOI: 10.1109/FTFC.2012.6231734.

[3] Siqin Liu and Avinash Karanth. "Dynamic Voltage and Frequency Scaling to Improve Energy-Efficiency of Hardware Accelerators". In: *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. 2021, pp. 232–241. DOI: 10.1109/HiPC53243.2021.00037.

[4] Roberto Medina and Liliana Cucu-Grosjean. "Work-in-Progress: Probabilistic System-Wide DVFS for Real-Time Embedded Systems". In: *2019 IEEE Real-Time Systems Symposium (RTSS)*. 2019, pp. 508–511. DOI: 10.1109/RTSS46320.2019.00051.

VIII. Appendix

A. Self-Assessment of Project Completion

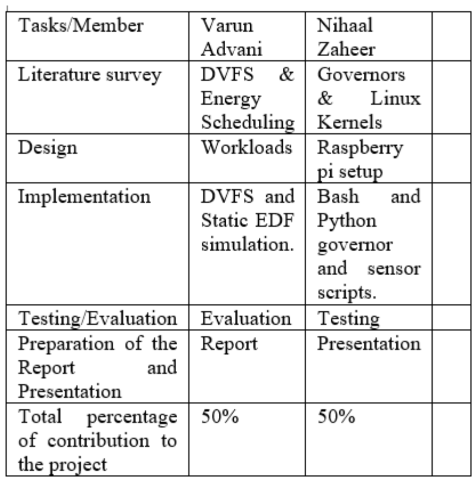


Self-Assessment of Project Completion:

Project learning objectives	**Status (Not/Partially /Mostly/Fully Completed)**	**Pointers in the document**
Self-contained description of the project goal, scope, and relevant requirements	Fully Completed	Section II, page 1-2
Self-contained description of the solutions (algorithms/protocol /applications/etc.)	Fully Completed	Section III, page 2-3
Adequate description of the implementation details (data structures, pseudo code segments, libraries used, etc.)	Fully Completed	Section III & IV, page 2-4
Testing and evaluation – test cases, metrics, test results, any relevant performance results	Fully Completed	Section V, page 4-6
Overall Project Success Assessment	Completely Successful	Section I – VII, Page 1- 6

Fig. 7: Self-Assessment of Project

B. Team Member Contributions



Tasks/Member	Varun Advani	Nihaal Zaheer	
Literature survey	DVFS & Energy Scheduling	Governors & Linux Kernels	
Design	Workloads	Raspberry pi setup	
Implementation	DVFS and Static EDF simulation.	Bash and Python governor and sensor scripts.	
Testing/Evaluation	Evaluation	Testing	
Preparation of the Report and Presentation	Report	Presentation	
Total percentage of contribution to the project	50%	50%	

Fig. 8: Team Member Contribution