Politecnico di Torino

Energy Management for IoT
01UDGOV

Master's Degree in Computer Engineering

Report lab 1
DPM Policy

Candidates:
Fabio Delbosco (S322244)
Tommaso Montedoro (S329567)

Referee:
Prof. Massimo Poncino

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction Chapter

The purpose of the first laboratory is to explore the basics of Dynamic Power Management by using a simulator to model a Power State Machine. The main objective is to examine different power management strategies and evaluate their performance. We organize the activity in three key sections:

1. Workload and PSM analisys
2. Timeout policy
3. History based policy

CHAPTER 2

Preliminary Concepts

The general idea of Dynamic Power Management (DPM) is to reduce power consumption by transitioning resources into a low-power "state" when they are underutilized.

Hardware Resources In general, hardware resources include cores, memories, device controllers, and sensors.

Low-Power States Low-power states refer to specific configurations where hardware components operate with reduced energy consumption while maintaining some level of functionality or readiness.

2.0.1 Power Consumption in Electronic Systems

- **Types of Power Consumption:**
  - *Dynamic:* Caused by logic switching in digital circuits, proportional to the operating frequency and the square of the supply voltage.
  - *Static:* Due to leakage currents, even when the system is idle.

2.0.2 Operating States and Power State Machines (PSMs)

- **Operating States:**
  - *Active:* The resource is operational and consuming full power.
  - *Idle:* The resource is ready but not actively used.
  - *Sleep/Off:* The resource is powered down or in a minimum-power state.
- **Power State Machine (PSM):**
  - An abstract model representing a resource's power behavior.
  - Nodes represent operating states.
  - Edges represent transitions between states, with associated energy and time costs.

2.0.3 Technologies and Techniques for Power Management

- **Clock Gating:** Disabling the clock in inactive parts of the circuit to reduce dynamic power consumption.
- **Power Gating:** Cutting off the power supply to inactive modules to reduce both static and dynamic power consumption.

- **Dynamic Voltage and Frequency Scaling (DVFS):**
  - Adjusting operating voltage and frequency according to workload.
  - Effective for optimizing both dynamic and static power.
- **Threshold Voltage Scaling:** Modulating transistor threshold voltage to reduce static leakage.

2.0.4 Preliminary Simulations - Probability Distribution Analysis

In our energy analysis of two different workloads, the Probability Density Function (PDF) and the Cumulative Distribution Function (CDF) provide crucial insights into the behavior of the system.

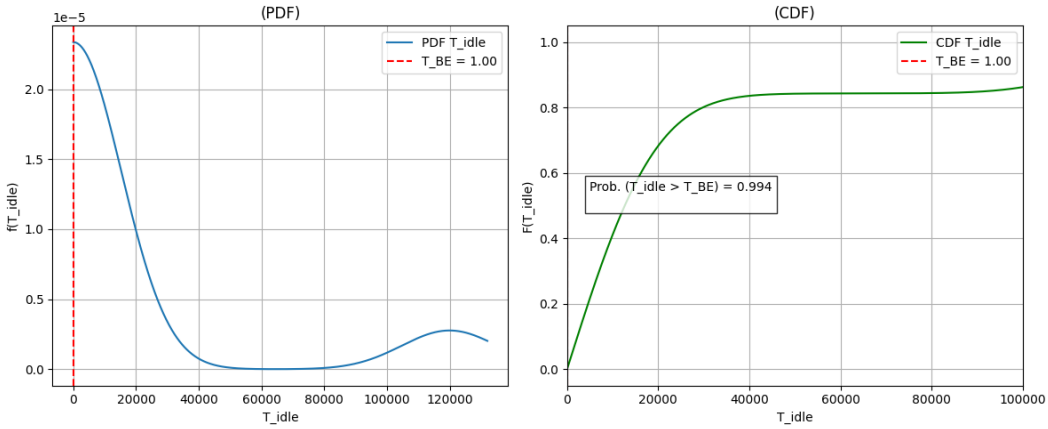


Figure 2.1: PDF and CDF for workload 1

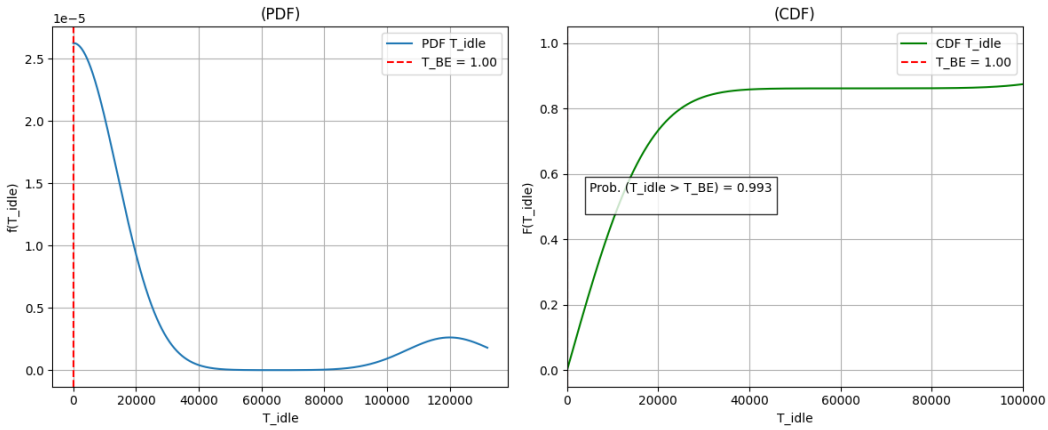


Figure 2.2: PDF and CDF for workload 2

Overall, the combined analysis of the PDF and CDF enables us to tailor Dynamic Power Management (DPM) strategies to the specific characteristics of each workload, thus ensuring both energy efficiency and robust system performance.

CHAPTER 3

DPM Analisys

In this chapter, we performed the simulation and evaluated the energy consumptions after active and inactive transitions. By using a PSM that can do transitions first from the Run state to the idle state and vice versa, and second from Run state to sleep.

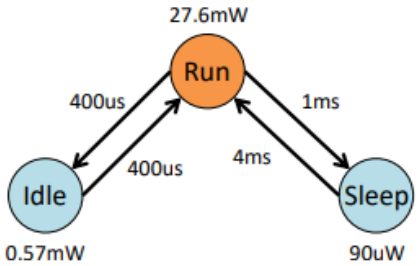


Figure 3.1: PSM

3.0.1 Energy-Performance Trade-offs

- **Benefits of DPM:**
  - Energy savings by extending idle times or reducing performance when possible.
- **Costs of DPM:**
  - Energy and time required for transitions between states.
  - Impact on performance, such as additional latencies or reduced throughput.
- **Break-even Time (TBE):**
  - The minimum idle time required to compensate for transition costs and achieve net energy savings.

3.1 Part 1 - Default Timeout Policy

The first part of the experience is based on a simplified version of the PSM that has only two states: Run and Idle state. The DPM doesn't allow transitions to the Sleep state, which is disabled.

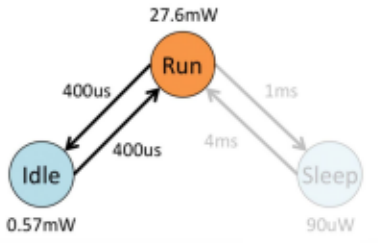


Figure 3.2: PSM with only two states

We compile the DPM simulator and run the simulations with two different workloads, using the the default Timeout Policy. The kind of worklad we work on is like an IoT device:

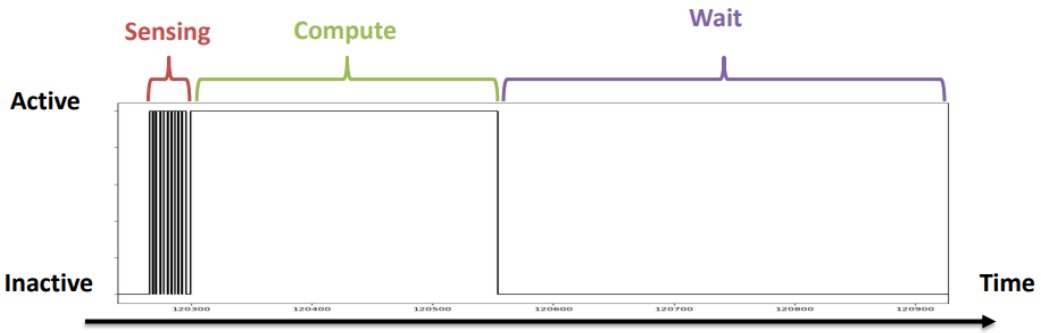


Figure 3.3: typical IoT workload

3.1.1 Workflow

First of all, we have to set our environment, by Make command we can create the object files and then could start the simulation by typing this command:

```
$ ./dpm_simulator -t 20 -psm example/psm.txt -wl ../workloads/workload_1.txt
```

By changing the value of the -t parameter we can set a new value for the timeout, thus analyzing the DPM behavior in different situations.

```
[psm] State Run: power = 27.6000mW
[psm] State Idle: power = 0.5700mW
[psm] State Sleep: power = 0.0900mW
[psm] Run -> Idle transition: energy  = 0.0100mJ, time = 0.4000ms
[psm] Run -> Sleep transition: energy  = 0.0200mJ, time = 1.0000ms
[psm] Idle -> Run transition: energy  = 0.0100mJ, time = 0.4000ms
[psm] Sleep -> Run transition: energy  = 2.0000mJ, time = 4.0000ms
```

Figure 3.4: PSM info

In this situation the system evaluates if it has to do a Run to Idle transition, it has to wait for a certain amount of time, called Timeout Time, after the end of it's work. As we said in 3.0.1, at this

point it could be useful to find the Break-even Time:

$$T_{\text{be}} = \begin{cases} T_{\text{tr}} & \text{se } P_{\text{on}} \geq P_{\text{tr}}, \\ T_{\text{tr}} + T_{\text{tr}} \frac{P_{\text{tr}} - P_{\text{on}}}{P_{\text{on}} - P_{\text{off}}} & \text{se } P_{\text{on}} < P_{\text{tr}}. \end{cases}$$

P_{on} is known, so we need to compute P_{tr}:

$$P_{\text{tr}} = \frac{E_{\text{tr}}}{T_{\text{tr}}}$$

Let's calculate T_{tr} and E_{tr}:

$$T_{\text{tr}} = T_{\text{on-off}} + T_{\text{off-on}} = 400\,\mu\text{s} + 400\,\mu\text{s} = 800\,\mu\text{s}$$

$$E_{\text{tr}} = E_{\text{on-off}} + E_{\text{off-on}} = 10\,\mu\text{J} + 10\,\mu\text{J} = 20\,\mu\text{J}$$

Now, compute P_{tr}:

$$P_{\text{tr}} = \frac{E_{\text{tr}}}{T_{\text{tr}}} = \frac{20\,\mu\text{J}}{800\,\mu\text{s}} = 25\,\text{mW}$$

Since $P_{\text{on}} = 27.6\,\text{mW} > P_{\text{tr}} = 25\,\text{mW}$, we have:

$$T_{\text{be}} = T_{\text{tr}} = 800\,\mu\text{s}$$

This break-even time T_{be} acts as a threshold: any idle time equal to or longer than T_{be} allows reducing power consumption without encountering any performance penalties.

```
Policy : 0[sim] Active time in profile = 2.924000s
[sim] Inactive time in profile = 1079.652000s
[sim] Tot. Time w/o DPM = 1082.576000s, Tot. Time w DPM = 1082.576500s
[sim] Total time in state Run = 2.932500s
[sim] Total time in state Idle = 1079.576000s
[sim] Total time in state Sleep = 0.000000s
[sim] Timeout waiting time = 0.008500s
[sim] Transitions time = 0.068000s
[sim] N. of transitions = 170
[sim] Energy for transitions = 0.0017000000J
[sim] Tot. Energy w/o DPM = 29.8790976000J, Tot. Energy w DPM = 0.6979953200J
```

Figure 3.5: Simulation with $\text{T}_{TO} = 20ms$

We tested many cases to find the best T_{TO} value in terms of energy performance. Thanks to a Python script we were able to obtain the trend of energy consumption as a function of the variation of T_{TO}.

As we can see for both workloads, if we increase the T_{TO} we have a worsening on the energy saved. For workload 2 we have a linear worsening, while for workload 1 we have a worsening up to about $T_{TO} = 2.5$s and then it remains stable.

3.1.2 Timeout and Power Consumption Relationship

The relationship between the timeout value T_{TO} and power consumption can be summarized as follows:

- Reducing the timeout value generally decreases power consumption.
- This behavior depends on the energy and time required for transitions between *Run* and *Idle* states.
- In this specific case, transition costs (energy and time) are very small, so frequent short transitions do not introduce any significant overhead.

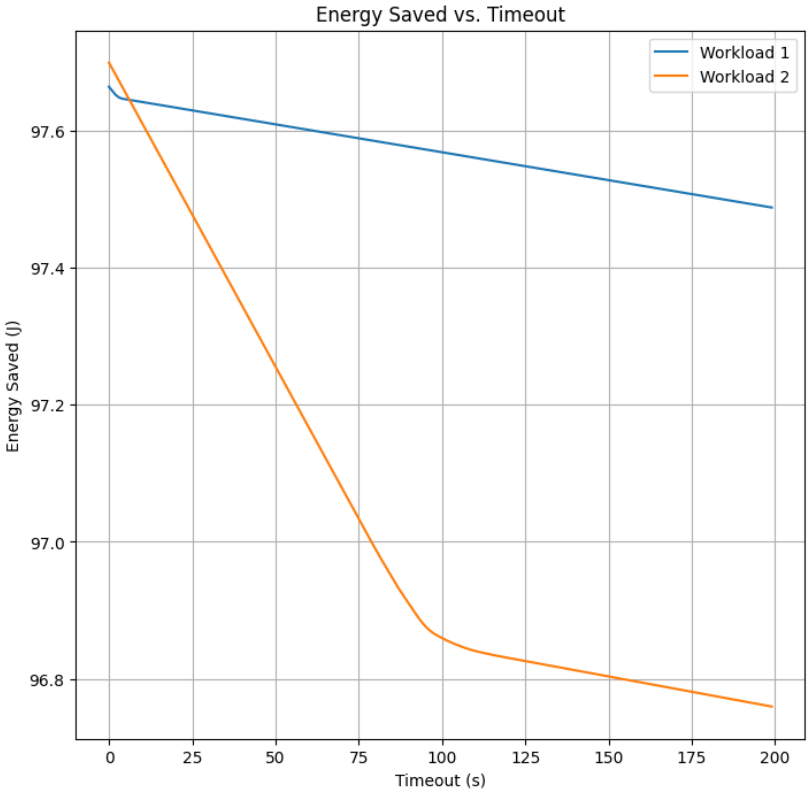
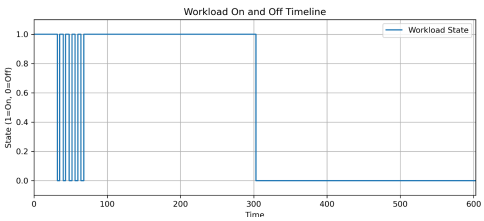
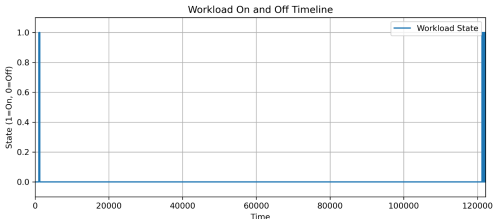


Figure 3.6: Energy saved comparison



(a) workload 1



(b) workload 2 (secondi??)

Figure 3.7: behavior of workloads

3.1.3 Transition Costs Analysis

- **Transition Details:**
  - *Run* $\rightarrow$ *Idle:* energy = 0.0100 mJ, time = 0.4000 ms.
  - *Idle* $\rightarrow$ *Run:* energy = 0.0100 mJ, time = 0.4000 ms.
- **Power During Transitions:**

$$\text{Transition Power} = \frac{\text{Energy Consumed}}{\text{Transition Time}}$$

  - *Run* $\rightarrow$ *Idle:* Power = 25.0000 mW $<$ 27.6000 mW (Run power)
  - *Idle* $\rightarrow$ *Run:* Power = 25.0000 mW $<$ 27.6000 mW (Run power)

- **Conclusion:** Performing a *Run* → *Idle* → *Run* transition is always energy-efficient, even for very short inactive windows.

3.1.4 Limitations (Not Considered)

- The model does not account for potential delays caused by *pre-wakeup* scenarios.
- **Pre-wakeup Issue:** If an *Idle* → *Run* transition occurs later than needed, this may result in performance losses or delays.
- **Focus:** This analysis focuses solely on power consumption, ignoring the above limitations.

3.1.5 Impact of Timeout on Transitions

- **Low Timeout (TTO):**
  - A low timeout increases the number of transitions (*Run* → *Idle* → *Run*).
  - Beneficial in this case, as it minimizes power consumption, but could be counterproductive in scenarios with higher transition costs.
- **High Timeout (TTO):**
  - If the timeout exceeds the longest inactive time, no transitions to *Idle* are performed.
  - In such cases, the system consumes as much power as if no DPM (Dynamic Power Management) were used:

$$E_{DPM} = E_{noDPM}$$

3.1.6 Workload Comparison

Both workloads exhibit a similar trend: as the timeout increases, the energy savings decrease almost linearly. For Workload 2, the decrease is rapid up to about 100 ms and then becomes smoother. In fact, the selection of the optimal timeout value depends on the specific workload; as shown in Figure 3.6, small T_{TO} values yield the maximum energy savings for both workloads.

- For Workload 1: $E_{\text{saving}} = 97.65\%$
- For Workload 2: $E_{\text{saving}} = 97.75\%$

3.2 Part 2 - Timeout Policy w/ Sleep state

For the second part, we have to work with another simplified version of the PSM, because only the Sleep state is enabled, so we compute the same steps of the simulation of the previous point.

```
$ ./dpm_simulator -ts 20 -psm example/psm.txt -wl ../workloads/workload_1.txt
```

As we said, the sleep state is enabled so the Total time in state Idle is equal to zero, the Total time in state Sleep is equal to 1079.2394 s

$$T_{\text{be}} = \begin{cases} T_{\text{tr}} & \text{se } P_{\text{on}} \geq P_{\text{tr}}, \\ T_{\text{tr}} + T_{\text{tr}} \frac{P_{\text{tr}} - P_{\text{on}}}{P_{\text{on}} - P_{\text{off}}} & \text{se } P_{\text{on}} < P_{\text{tr}}. \end{cases}$$

P_{on} is known, so we need to compute P_{tr}:

$$P_{\text{tr}} = \frac{E_{\text{tr}}}{T_{\text{tr}}}$$

```
Policy : 1[sim] Active time in profile = 2.924000s
[sim] Inactive time in profile = 1079.652000s
[sim] Tot. Time w/o DPM = 1082.576000s, Tot. Time w DPM = 1082.576100s
[sim] Total time in state Run = 3.291700s
[sim] Total time in state Idle = 0.000000s
[sim] Total time in state Sleep = 1079.239400s
[sim] Timeout waiting time = 0.367700s
[sim] Transitions time = 0.045000s
[sim] N. of transitions = 18
[sim] Energy for transitions = 0.0181800000J
[sim] Tot. Energy w/o DPM = 29.8790976000J, Tot. Energy w DPM = 0.2061624660J
```

Figure 3.8: Energy saved comparison

Let's calculate T_{tr} and E_{tr}:

$$T_{\text{tr}} = T_{\text{on-off}} + T_{\text{off-on}} = 1\,\text{ms} + 4\,\text{ms} = 5\,\text{ms}$$

$$E_{\text{tr}} = E_{\text{on-off}} + E_{\text{off-on}} = 0.02\,\text{mJ} + 2\,\text{mJ} = 2.02\,\text{mJ}$$

Now, compute P_{tr}:

$$P_{\text{tr}} = \frac{E_{\text{tr}}}{T_{\text{tr}}} = \frac{2.02\,\text{mJ}}{5\,\text{ms}} = 404\,\text{mW}$$

Since $P_{\text{on}} = 27.6\,\text{mW} < P_{\text{tr}} = 404\,\text{mW}$, we have:

$$T_{\text{be}} = T_{\text{tr}} + T_{\text{tr}} \frac{P_{\text{tr}} - P_{\text{on}}}{P_{\text{on}} - P_{\text{off}}} = 73.41\,\text{ms}$$

3.2.1 Code Implementation

To run this new PSM, the simulator has been modified to enable transitions beyond the standard Run and Idle states. In the function `dpm_decide_state` in `src/dpm_policies.c`, instead of transitioning to the Idle state when the timeout expires, the system now transitions directly to the Sleep state. This change ensures that when the timeout runs out, the simulator immediately enters a deeper low-power state, which could lead to greater energy savings. (see appendix)

3.2.2 Results Analysis

At this point, we can clearly see that, from an energy-saving perspective, the Sleep PSM is the most efficient, achieving energy savings of 99.35% for Workload 1 and 98.78% for Workload 2. As expected, remaining in the Sleep state results in significantly lower power consumption than in the Idle state. However, it is important to note that the power cost for the transition, P_{tr}, differs greatly between these states: for instance, the Run→Idle transition consumes about 20 mW, while the Run→Sleep transition requires approximately 404 mW. Consequently, if the system were to perform frequent Run→Sleep→Run transitions, the cumulative transition cost could negate the benefits of operating in the lower-power Sleep state.

Nonetheless, if the majority of idle periods are shorter than the break-even time for the Sleep state—i.e., the duration for which the energy savings in Sleep offset the higher transition cost—the transition to Sleep remains advantageous. Therefore, the optimal policy choice must balance the energy savings achieved in each state against the overhead of the state transitions, taking into account the specific workload characteristics and the corresponding break-even times.

The answer to which policy should be preferred is provided in the next paragraph.

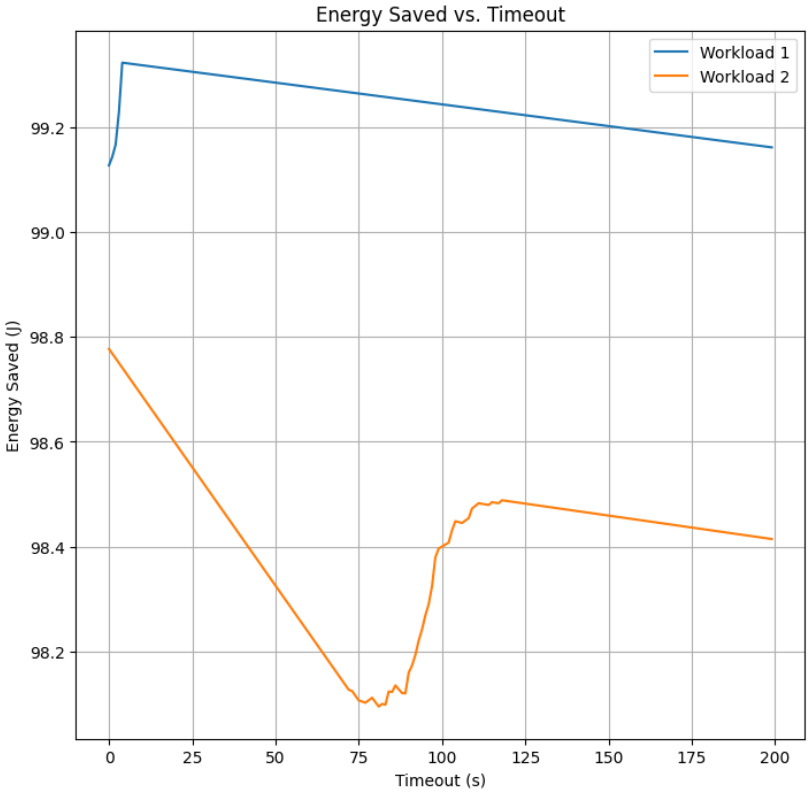


Figure 3.9: Energy saved comparison

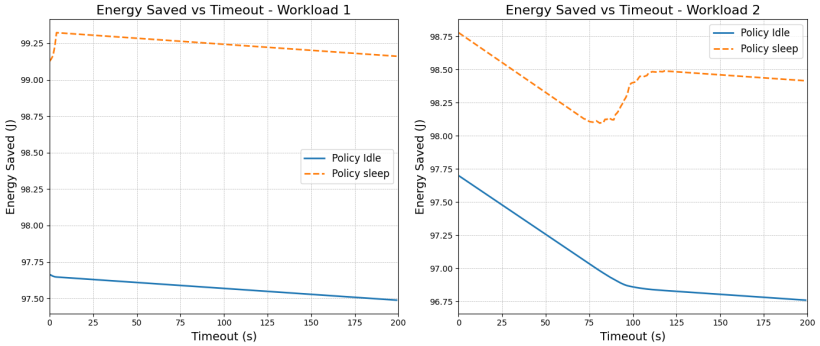


Figure 3.10: Energy saved comparison

3.3 Part 3 - Predictive History Policy

In this section, we present the implementation and analysis of the predictive history policies integrated into our Dynamic Power Management (DPM) simulator. These policies leverage historical information to predict the duration of upcoming inactive periods, thereby enabling more informed decisions regarding power state transitions.

3.3.1 Introduction

History-based policies, such as `DPM_HISTORY`, `DPM_HISTORY_A`, `DPM_HISTORY_I`, and `DPM_HISTORY_EXP`, utilize the durations of previous inactive intervals to estimate the upcoming idle time. In addition,

the oracle-based variant (`DPM_ORACLE`) uses perfect knowledge of the next work item arrival time to compute the predicted inactive time. (see appendix)

3.3.2 Simulator Modifications

To support the predictive history policies, several key modifications were made to the simulator

3.3.3 Scripts and Graphical Analysis

To evaluate and fine-tune the predictive history policies, several scripts were developed:

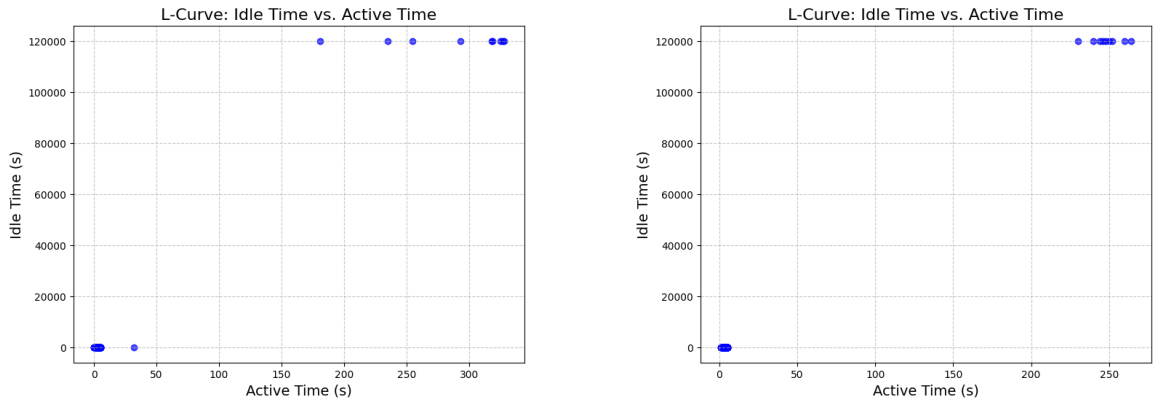
- **Parameter Grid Search:** Scripts iterate over candidate threshold values (and α for the exponential policy) to identify the combination that maximizes energy savings.
- **L-Curve Analysis:** An L-curve graph is used to plot the relationship between the active time (i.e., the time when the active period ends) and the subsequent idle time. This graph provides insight into the workload characteristics, helping to define appropriate thresholds.
- **Energy Savings vs. Threshold Plots:** Additional plots illustrate how energy savings vary with different threshold settings, providing a visual tool to support the parameter selection process.

3.3.4 L-Curve Analysis

The L-curve graph plots the idle period against the active time (the end time of an active interval) for each work item. Typically, the graph shows:

- A rapid decrease in idle times for short active periods, indicating bursts of activity.
- A plateau for longer active periods, suggesting more consistent idle durations.

This visual representation aids in setting the best thresholds for our policies: if the idle time consistently exceeds the break-even point after a certain active time, then a transition (either to Idle or Sleep) can be justified.



(a) workload 1

(b) workload 2

Figure 3.11: L-curve for both workloads

3.3.5 History-Based Policy Using Last Active Time

In this approach, the decision on the next power state is based on the time at which the last active phase ends. The underlying assumption is that the duration of the most recent active period provides useful information for predicting the upcoming idle period. More specifically, once the system completes its active phase, the time of its end, denoted as $t_{\text{active\_end}}$, is used to estimate the idle duration as follows:

$$T_{\text{pred}} = t_{\text{next}} - t_{\text{active\_end}},$$

where t_{next} is the arrival time of the next work item. This predicted idle time T_{pred} is then compared against two thresholds:

- $T_{\text{thresh\_idle}}$: if T_{pred} exceeds this lower threshold, it is beneficial to switch to the Idle state.
- $T_{\text{thresh\_sleep}}$: if T_{pred} exceeds this higher threshold, the transition to Sleep becomes more energy efficient.

Thus, the state decision is made according to:

$$\begin{cases} \text{Sleep} & \text{if } T_{\text{pred}} > T_{\text{thresh\_sleep}}, \\ \text{Idle} & \text{if } T_{\text{thresh\_idle}} < T_{\text{pred}} \leq T_{\text{thresh\_sleep}}, \\ \text{Run} & \text{otherwise.} \end{cases}$$

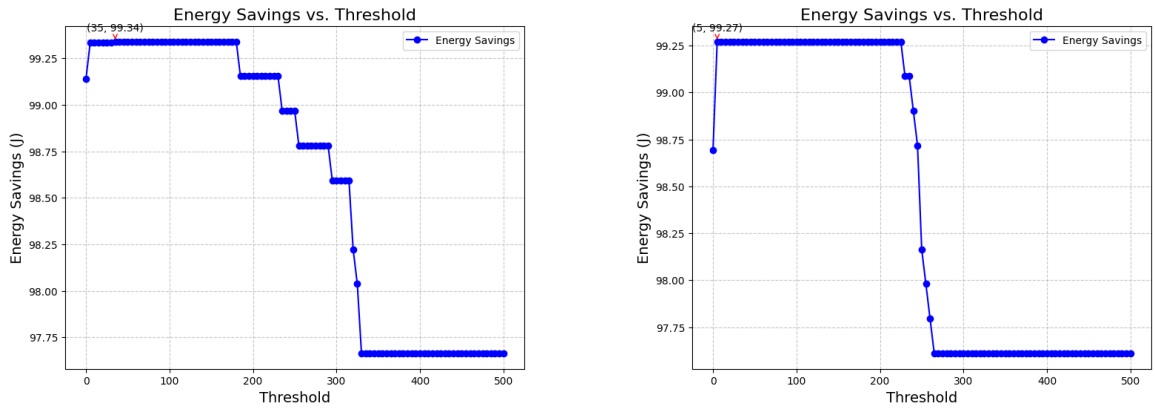
In our simulator, this policy is implemented in the function `dpm_decide_state` under the case corresponding to history-based policies that use the last active time (e.g., `DPM_HISTORY_A`). At the moment of decision, the simulator computes the predicted idle time by subtracting the time at which the active phase ended ($t_{\text{active\_end}}$) from the arrival time of the next work item (t_{next}). The thresholds $T_{\text{thresh\_idle}}$ and $T_{\text{thresh\_sleep}}$ are provided via the history parameters (`hparams.threshold[0]` and `hparams.threshold[1]`), which are set based on the energy and performance trade-offs of the system.

This policy allows the system to adapt its power state transitions based on recent activity patterns. However, its effectiveness depends on the correlation between the length of the active phase and the subsequent idle period. In workloads where this relationship is strong, the policy can achieve significant energy savings. Conversely, in scenarios with high variability, the prediction might be less accurate, and the thresholds must be carefully tuned.

In this analysis, the Pareto curve represents, for each value of α, the combination of thresholds that yields the maximum energy savings. From the resulting graph, it is evident that the optimal α for Workload 1 is 0.6, while for Workload 2 the best performance is achieved with $\alpha = 0.3$. This finding underlines that the optimal balance between recent and past history is highly dependent on the specific characteristics of the workload.

3.3.6 Analysis of History Policy Graphs

In our experiments, approximately 300 combinations of threshold values were simulated for the history-based policy. The results revealed that the most significant variations in energy savings occurred when threshold0 was held constant and threshold1 was varied. Consequently, the generated graphs plot energy savings as a function of threshold1, with threshold0 fixed at a constant value. This approach allows us to isolate the effect of the upper threshold on the policy performance, highlighting the sensitivity of energy savings to changes in threshold1. By examining these graphs, we can identify the optimal threshold1 value that maximizes energy savings for the given workload.



(a) workload 1

(b) workload 2

Figure 3.12: Energy Savings w Last Active Time

3.3.7 History-Based Policy Using Previous Inactive Time

In addition to using the last active time, our simulator also supports a policy that relies on the *previous inactive time* to predict upcoming idle durations. The rationale is that if the most recent inactivity was particularly long (or short), the next interval may exhibit similar behavior.

- **Policy Logic:** Once the system becomes inactive at time $t_{\text{inactive\_start}}$, we retrieve the duration of the last inactive period from our history array. This duration, T_{idle}^{n-1}, serves as the primary predictor for the current idle phase.
- **State Decision:** If T_{idle}^{n-1} is larger than a certain threshold $T_{\text{thresh\_sleep}}$, the system transitions to the **Sleep** state. If it is only greater than $T_{\text{thresh\_idle}}$ (but not the higher threshold), the system transitions to **Idle**. Otherwise, it remains in the **Run** state.

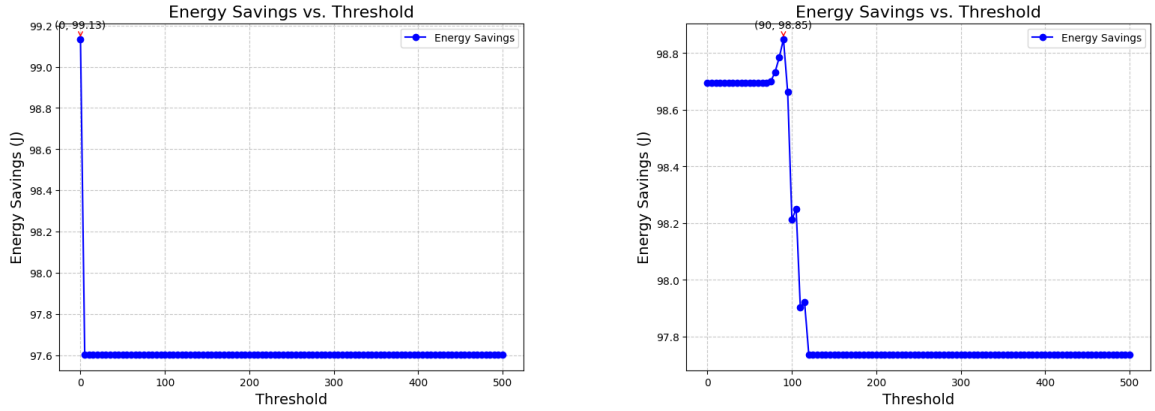
Mathematically, if $T_{\text{pred}}^{n} = T_{\text{idle}}^{n-1}$ represents the predicted idle duration based on the last observed inactivity, the policy can be summarized as:

$$\begin{cases} \text{Sleep} & \text{if } T_{\text{pred}}^{n} > T_{\text{thresh\_sleep}}, \\ \text{Idle} & \text{if } T_{\text{pred}}^{n} > T_{\text{thresh\_idle}}, \\ \text{Run} & \text{otherwise.} \end{cases}$$

Implementation Details. In the simulator, we maintain a *history array* that records the last few inactivity intervals. Specifically, when a new inactive phase begins, we perform:

1. *History Update:* At the end of each inactive interval, we call `dpm_update_history` with the duration of that interval.
2. *Policy Decision:* In `dpm_decide_state`, we retrieve T_{idle}^{n-1} (the most recent value from the history array) and compare it to the thresholds stored in `hparams.threshold`.

Advantages and Limitations. This policy adapts quickly to recent workload changes: if the previous idle period was unexpectedly long, the policy is more inclined to choose a deeper sleep state. However, if the system's behavior changes drastically (e.g., from short to very long idle intervals, or vice versa), the policy may mispredict the new idle duration and thus make a suboptimal power state decision. Consequently, fine-tuning the thresholds and combining them with additional history-based heuristics can significantly improve overall energy savings.



(a) workload 1

(b) workload 2

Figure 3.13: Energy Savings w Previous Inactive Time

3.3.8 Adaptive Exponential Policy

The coefficient α in the exponential prediction formula

$$T_{\text{pred}}^{n} = \alpha\, T_{\text{idle}}^{n-1} + \left(1-\alpha\right) T_{\text{pred}}^{n-1}$$

controls the relative weight of the most recent idle interval versus the historical predictions. Specifically:

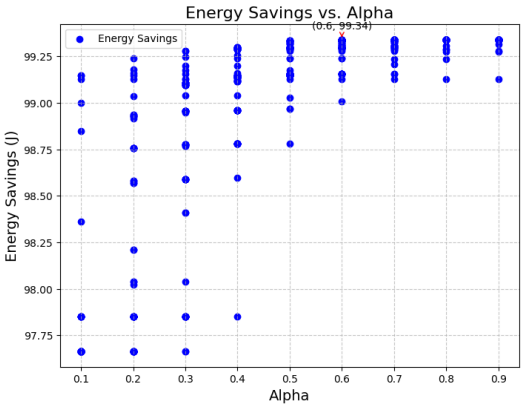
- If $\alpha = 0$, then $T_{\text{pred}}^{n} = T_{\text{pred}}^{n-1}$. In other words, recent history has no effect, and the prediction remains the same as in the previous step.
- If $\alpha = 1$, then $T_{\text{pred}}^{n} = T_{\text{idle}}^{n-1}$, which effectively ignores all previous predictions and relies solely on the last observed idle interval.

By choosing α between 0 and 1, we can balance between these two extremes. This exponential policy can effectively forecast idle periods in most scenarios, as it adapts to changes in the workload while still retaining some memory of past behavior. However, a critical case arises when a very long idle period suddenly occurs after a series of nearly uniform, shorter idle intervals. In such a situation, the exponential predictor may underestimate the idle time, leading to a suboptimal power state decision. Balancing α and defining suitable thresholds for transitioning to lower-power states thus remain crucial for maximizing energy savings while maintaining acceptable performance.

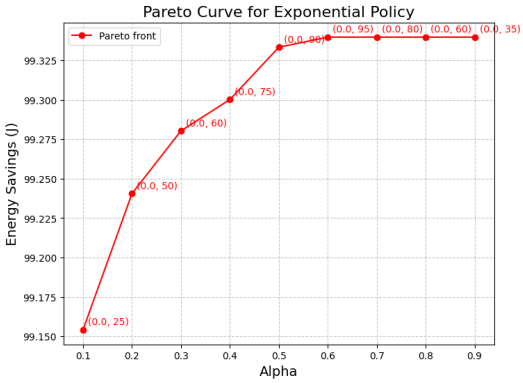
3.3.9 Optimal Alpha Selection via Pareto Curve Analysis

For the exponential prediction policy, the parameter α controls the relative weight given to the most recent idle interval versus the past predictions. Since, for a given α, multiple energy savings values are obtained due to the variation in threshold combinations, we plotted a Pareto curve to identify the optimal points.

In this analysis, the Pareto curve represents, for each value of α, the combination of thresholds that yields the maximum energy savings. From the resulting graph, it is evident that the optimal α for Workload 1 is 0.6, while for Workload 2 the best performance is achieved with $\alpha = 0.3$. This finding underlines that the optimal balance between recent and past history is highly dependent on the specific characteristics of the workload.

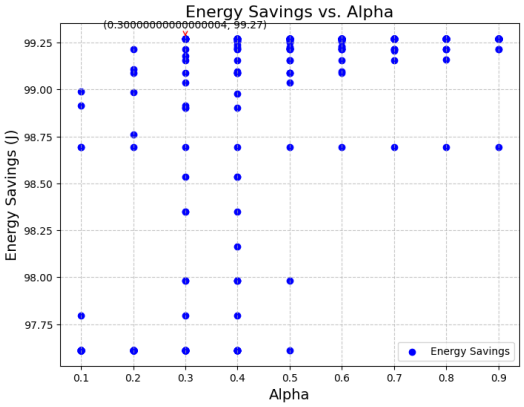


(a) workload 1

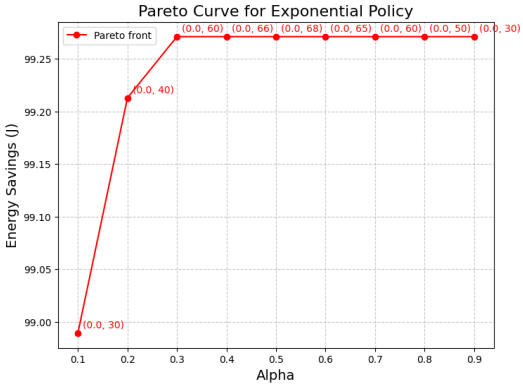


(b) workload 1 (pareto curve)

Figure 3.14: Energy Savings w Exp. Policy



(a) workload 2



(b) workload 2 (pareto curve)

Figure 3.15: Energy Savings w Exp. Policy

3.3.10 Oracle-Based Policy

The oracle-based policy, referred to as `DPM_ORACLE`, exploits perfect knowledge of the future to determine the optimal power state transition. In this approach, a global variable, `g_next_arrival`, is maintained and continuously updated with the arrival time of the next work item. When the system transitions from an active phase to an inactive phase, denoted by the time $t_{\text{inactive\_start}}$, the policy calculates the predicted inactive duration as follows:

$$T_{\text{pred}} = g_{\text{next\_arrival}} - t_{\text{inactive\_start}},$$

where $g_{\text{next\_arrival}}$ represents the exact arrival time of the next work item.

The decision mechanism compares this predicted idle time with two predefined thresholds:

- If $T_{\text{pred}} > T_{\text{BE\_sleep}}$, the system transitions to the **Sleep** state, as the predicted idle period is sufficiently long to amortize the higher transition cost.
- If $T_{\text{pred}} > T_{\text{BE\_idle}}$ but $T_{\text{pred}} \leq T_{\text{BE\_sleep}}$, the system enters the **Idle** state, which, while less energy-saving than Sleep, incurs a lower transition cost.
- Otherwise, if $T_{\text{pred}} \leq T_{\text{BE\_idle}}$, the system remains in the **Run** state.

This policy provides an ideal benchmark since it leverages perfect foresight of the upcoming idle duration. By comparing the predicted inactive time against carefully tuned thresholds, the oracle-based policy achieves a balance between energy savings and transition costs. Although such perfect prediction is rarely available in practice, it serves as a reference point against which more practical predictive policies can be evaluated.

(a) workload 1

(b) workload 2

Figure 3.16: Energy Savings w Exp. Policy

3.4 Conclusion

In this study, we analyzed several Dynamic Power Management (DPM) policies, with a focus on history-based approaches. Our results show that the history-based policies, particularly those based on the last active time, the exponential prediction method, and the oracle-based policy, are the most effective in maximizing energy savings.

Specifically:

- The policy based on *last active time* leverages the time at which the active phase ends to accurately predict the upcoming idle period, enabling efficient state transitions.
- The *exponential prediction policy* combines recent and past inactive times through the parameter α, allowing for a flexible balance that adapts to workload variations.
- The *oracle-based policy* benefits from perfect foresight of the next work item's arrival, which provides an ideal benchmark for energy saving.

However, the selection of the best absolute policy must consider not only energy savings but also the specific characteristics of the workload, the designer's requirements, and other fundamental factors such as the delay introduced by state transitions. In real-world scenarios, the optimal policy is highly workload-dependent and must strike a balance between energy efficiency and performance overhead.

In summary, while our history-based policies consistently outperform other approaches in terms of energy savings, a tailored solution that accounts for the delay, transition costs, and specific workload patterns is essential to achieve the best overall performance.

APPENDIX A

Code used for dpm implementation and pyhon plotting

I

A.1 List of Code Modules Used

Below is a list of the source files employed in this project along with a brief description of their purpose:

listings

```
1    switch (policy) {
2
3        case DPM_TIMEOUT_IDLE:
4            if(t_curr > t_inactive_start + tparams.timeout) {
5                *next_state = PSM_STATE_IDLE;
6            } else {
7                *next_state = PSM_STATE_RUN;
8            }
9            break;
10
11       case DPM_TIMEOUT_SLEEP:
12           if(t_curr > t_inactive_start + tparams.timeout) {
13               *next_state = PSM_STATE_SLEEP;
14           } else {
15               *next_state = PSM_STATE_RUN;
16           }
17           break;
18
19       case DPM_HISTORY_A:
20           /*Policy 1 the next Tidle = last active time*/
21           if(t_curr >= t_inactive_start){
22               if(history[DPM_HIST_WIND_SIZE-1] > hparams.threshold[1]){
23                   *next_state = PSM_STATE_SLEEP;
24               }
25               else if(history[DPM_HIST_WIND_SIZE-1] > hparams.threshold[0])
                     {
26                   *next_state = PSM_STATE_IDLE;
```

```
27                  }
28                  else
29                      *next_state = PSM_STATE_RUN;
30              }
31              else
32                  *next_state = PSM_STATE_RUN;
33              break;
34
35          /*predictive policy based on previuous inactive time*/
36          case DPM_HISTORY_I:
37              /*Policy 1 the next Tidle = last active time*/
38              if(t_curr >= t_inactive_start){
39                  if(history[DPM_HIST_WIND_SIZE-1] > hparams.threshold[1]){
40                      *next_state = PSM_STATE_SLEEP;
41                  }
42                  else if(history[DPM_HIST_WIND_SIZE-1] > hparams.threshold[0])
                        {
43                      *next_state = PSM_STATE_IDLE;
44                  }
45                  else
46                      *next_state = PSM_STATE_RUN;
47              }
48              else
49                  *next_state = PSM_STATE_RUN;
50              break;
51
52          case DPM_HISTORY_EXP:
53              /*Policy 4 the next Tidle = last active time*/
54              if(t_curr >= t_inactive_start){
55                  if(prev_state == 0)
56                      exp_pred(hparams, history, t_last_pred);
57                  if(*t_last_pred > hparams.threshold[1]){
58                      *next_state = PSM_STATE_SLEEP;
59                  }
60                  else if(*t_last_pred > hparams.threshold[0]){
61                      *next_state = PSM_STATE_IDLE;
62                  }
63                  else
64                      *next_state = PSM_STATE_RUN;
65              }
66              else
67                  *next_state = PSM_STATE_RUN;
68              break;
69
70          case DPM_ORACLE:
71              if(t_curr >= t_inactive_start) {
72                  psm_time_t predicted_inactive = g_next_arrival -
                        t_inactive_start;
73                  if(predicted_inactive > 73.41){ // calculated from the psm
74                      *next_state = PSM_STATE_SLEEP;
75                  }
76                  else if(predicted_inactive > 0.8){ // calculated from the psm
77                      *next_state = PSM_STATE_IDLE;
78                  }
```

```
79                  else {
80                      *next_state = PSM_STATE_RUN;
81                  }
82                  }
83              else {
84                  *next_state = PSM_STATE_RUN;
85              }
86              break;
```

Listing A.1: PSM code implementation