A Comparative Evaluation of Power-Down Algorithms

Yu Lim, Leo Lei
The University of Texas at Austin
{yulim,leolei}@utexas.edu

Abstract—**Dynamic power management (DPM) algorithms help reduce the power consumption of external devices which follow the ACPI standard at the cost of device performance. In modern systems, there exist a wide range of applications that present workloads which can often be reasonably modelled by specific statistical distributions. In this paper, we explore the energy efficiency and performance of five different DPM algorithms in the context of several of these workload distributions run across three hard disk device models. We design and implement our own simulation suite to mimic the hard disk devices running these algorithms. Our results suggest that under stochastic workloads, DPM algorithms will almost certainly save on energy consumption with acceptable performance drops, under the condition that the workload pattern permits shutting down and waking up devices. We also observe that no single algorithm excels across all different types of workloads in terms of energy efficiency and performance. Certain predictive algorithms have greater efficiency when workloads exhibit time trend patterns whereas simplistic algorithms perform consistently in a wide range of distributions.**

I. Introduction

A. Dynamic Power Management (DPM) Algorithms

Power management algorithms help in reducing energy consumption by efficiently utilizing system resources. Reducing energy consumption not only benefits individual users in terms of cost savings but also contributes to a broader effort to reduce the carbon footprint associated with computing activities. DPM plays a role in achieving these environmental goals by optimizing the power consumption of individual device components for different workloads at the expense of a decrease in performance. By their nature, DPM algorithms are implemented as part of the driver for some device so that whatever requests are sent from the kernel to the device are first intercepted by the DPM algorithm to decide how to best configure the device to save on energy consumption while minimizing performance drops.

Most modern hardware systems provide support for the Advanced Configuration and Power Interface (ACPI) standard written by Intel, first released in 1996 [1]. The ACPI standard provides an interface for ACPI-compliant operating systems to monitor and adjust various components in the hardware in relation to performance and energy consumption. The ACPI specification is split into three major categories to interact with the various hardware components in the system: D-states, C-states, and P-states. C-states and P-states describe the power and performance states of the CPU, and are primarily adjusted automatically by the underlying firmware, rather than the operating system's kernel. D-states apply to the individual devices in the system and are typically split from D0 to D3 with D0 indicating the device is fully-on in an operating state and D3 indicating the device has mostly powered off and may/may not be responsive to any incoming signals from the bus [2]. Different devices may have different specifications for the same D-state.

The focus of this paper is on Dynamic Power Management (DPM) algorithms that will target managing devices, rather than working directly with the CPU. Specifically, we are interested in analyzing the behavior of power management algorithms when applied to hard-disk drive (HDD) devices. Thus, we are more interested in the D-states that are provided by the ACPI standard rather than D-states and C-states. For both practicality and simplicity, we group these device power states into three primary states that our DPM algorithms can manage: active, standby, and sleeping. Our motivation for choosing HDDs as our device of interest for this study is that modern HDDs can be accurately represented by the three primary power states above which we are interested in with simple transitions between these power states (spinning up and spinning down).

B. Providing a Baseline Comparison for DPM Algorithms

In order to consolidate and compare various DPM algorithms found across the literature, we first needed to settle on several metrics for analyzing the performance and energy efficiency of these algorithms, many of which have been implemented and tested by previous literature [1].

For each algorithm, we consider these metrics in the context of running a workload W on some device D. While D is running W, it could be in one of three power states: an active state handling some request, a standby state not performing any useful work, or a sleeping state (powered off). We assume that device D cannot exist in any power state outside of active, standby, or sleeping. Modern HDDs typically operate in these three states and transitioning from the active state to standby state typically takes much less time than transitioning from an active or standby to a sleeping state (the difference is on the scale of microseconds to seconds [3]). Some older HDDs only exist in either a active or a sleeping state regardless of whether there are incoming requests or not (HDD A in Table I). Workload W is defined to run for some time length of L milliseconds. In each millisecond of W, there is either one new incoming request (which we consider a busy millisecond) or no request (which we consider an idle millisecond). We further define a consecutive interval of busy milliseconds as a busy time interval and a consecutive interval of idle milliseconds as an idle time interval. Thus, W is split into alternating busy and idle time intervals. We denote T_i as the number of consecutive milliseconds in the i^{th} time interval with incoming requests or with no incoming traffic. To differentiate between busy and idle intervals of incoming requests, we introduce $T_{i,busy}$ and $T_{i,idle}$ where only one of the values is positive while the other is 0.

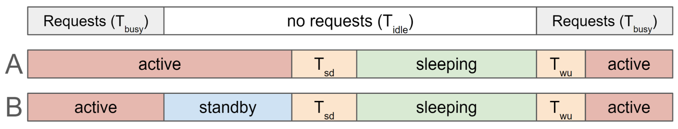

Figure 1: Possible power states for two HDDs. HDD A does not have an idle state while HDD B has an idle state.

All of the algorithms that we analyze in this paper make use of the shutdown and startup transition times of D (in seconds), denoted by T_{sd} and T_{wu} respectively. We assume that these values are different for different HDD devices and generally remain constant for each device independent of workload and time. Each HDD also has a power rate (in Watts) associated with each power state: P_{active} for the active state, $P_{standby}$ for the standby state, and P_{sleep} for the sleeping state (see Figure 1). Additionally, the transition states of D for shutting down and waking up have power P_{sd} and P_{wu} respectively. For any HDD, $P_{standby} \geq P_{sleep}$. As a consequence of this, every HDD has a unique time threshold α such that if $T_{i,idle} > \alpha$, we would save more power by shutting down D and if $T_{i,idle} < \alpha$, we should keep D in standby mode. Let $\alpha = T_{sd} + x$ where x is some non-negative amount of time. We can calculate x (and thus, α) with the following inequality:

$$\begin{aligned}
&\alpha \cdot P_{standby} + T_{wu} \cdot P_{active} \geq T_{sd} \cdot P_{sd} \\
&\qquad + x \cdot P_{sleep} + T_{wu} \cdot P_{wu} \\
\implies &(T_{sd} + x) \cdot P_{standby} + T_{wu} \cdot P_{active} \geq T_{sd} \cdot P_{sd} \\
&\qquad + x \cdot P_{sleep} + T_{wu} \cdot P_{wu} \\
\implies &T_{sd} \cdot P_{standby} + x \cdot P_{standby} + T_{wu} \cdot P_{active} \\
&\qquad \geq T_{sd} \cdot P_{sd} + x \cdot P_{sleep} + T_{wu} \cdot P_{wu} \\
\implies &x \cdot P_{standby} - x \cdot P_{sleep} \geq T_{sd} \cdot P_{sd} + T_{wu} \cdot P_{wu} \\
&\qquad - T_{sd} \cdot P_{standby} - T_{wu} \cdot P_{active} \\
\implies &x \cdot (P_{standby} - P_{sleep}) \geq T_{sd} \cdot (P_{sd} - P_{standby}) \\
&\qquad + T_{wu} \cdot (P_{wu} - P_{active}) \\
\implies &x \geq \frac{T_{sd} \cdot (P_{sd} - P_{standby}) + T_{wu} \cdot (P_{wu} - P_{active})}{P_{standby} - P_{sleep}}
\end{aligned} \tag{1}$$

It is possible that $P_{sd} \leq P_{standby}$. In this case, $\alpha = T_{sd}$ since as long as $T_{i,idle} \geq T_{sd}$, we can shutdown and save on energy consumption.

C. *Preliminary Predictions*

In this paper, we analyze the energy efficiency and performance of five unique DPM algorithms, each detailed in the next section (Section 2). In regards to energy efficiency, we anticipate that either the Markov Model or the Logistic Regression will consume the least amount of energy. Especially, for our exponentially-distributed workload, we know that logistic regression and Markov Chains in practice perform well in predicting exponential distributions and their ability to effectively reduce power consumption is supported by the results in the literature. We also expect the Timeout Algorithm to perform well, and potentially even outperform these predictive algorithms due to its simplicity and applicability to a wide range of workloads. Finally, we expect to see the L-shaped algorithm perform well in the Short-Long Busy-Idle workload but poorly on the other two workloads due to there being no correlation between the length of consecutive busy and idle intervals.

II. Algorithms

In this section, we introduce interesting DPM Algorithms found throughout past literature as well as propose a novel DPM algorithm building upon some of the insights we have collected. We categorize the algorithms into two broad categories: simple and predictive. We cover two simple algorithms - Timeout and L-Shape - and three predictive algorithms - Exponential Moving Average, Markov Chain, and Logistic Regression.

The first DPM algorithm we consider is the timeout algorithm [1]. This algorithm assumes that if a device stays idle for longer than a time period γ, it will remain idle for a period of time greater than α with 95% confidence. Thus, when the driver senses that the device has already been idle for time γ, it will put it to sleep. More advanced timeout algorithms will adjust γ based on the ratio $\frac{T_{i-1,idle}}{T_{wu}}$, where $T_{i-1,idle}$ is the amount of time spent idling in the previous sleep-wake cycle. This ratio is one metric that can balance the need for energy efficiency and user convenience. For instance, a user should generally expect to wait for a device to wake up after an hour of not using it. On the other hand, they would be irritated if they must wait a few seconds for the device to wake up when it has only been idle for a short period of time.

We can also create a distribution of idle times $T_{i,idle}$ for the most recent few cycles and adjust our γ based on the mean or any similar relevant statistic. For instance, if the last 3 idle times were relatively short, we can assume the next idle time will be short as well and thus we would want to increase γ. Although timeout algorithms are relatively simple, they are one of the more common algorithms seen in practice for controlling hard-disks due to their simplicity and their surprising effectiveness [4].

A. *Markov Algorithm*

We can model devices and requests to devices as a stochastic process, and thus we can treat energy efficiency as an optimization problem [1]. For example, one way we can approach this optimization problem is with Markov Chains - a stochastic model that describes a sequence of events in which the probability of each event depends only on the state attained in the previous event. A key property of Markov processes is that they are memory-less. This means that the probabilities of transitioning to the next event only depends on our current event and not any previous ones.

We detailed above that the state of a disk can be split into several states, such as wake-up, shutdown, standby, sleep, and active. We can then utilize the Markov model to predict the probability that at a given moment the driver should wake the device or put it to sleep. Since, in general, we can view the probability of these states as continuous random variables, we can utilize a more advanced variant known as Semi-Markov models. Semi-Markov models are not memory-less with respect to time; i.e. the transition probabilities change depending on how long the device has remained in a given state. This provides a more accurate model of the real-world, since the longer a device is idle up to a certain point, the more likely it will stay idle for a long period of time. When the Markov algorithm first runs,there is a short trial period where data is collected for the probability transition matrix.

For our implementation specifically, we specified a look-back period of the last L idle periods ($T_{i-1,idle}$ to $T_{i-L,idle}$). For $T_{i,idle}$, we maintain a list of length L with the j^{th} element being a 1 if $T_{j,idle} \geq \alpha$ and 0 otherwise. Our input into the Markov algorithm at $T_{i,idle}$ is a hash number of this list in the form of a binary string. Based on this hash, our Markov algorithm predicts the probability $p(T_{i,idle} \geq \alpha)$. If the probability is greater than 50%, we call for a shutdown. Otherwise, we keep the device in standby mode because we expect that $T_{i,idle} < \alpha$.

B. Exponential Algorithm

The exponential algorithm utilizes an exponential moving average formula to more accurately predict the length of the next idle period [1]. This is done by averaging the past N idle-period lengths where N is some look-back parameter we can adjust in the algorithm. If we define $T_{i+1,idle}$ as our prediction for the length of the next idle period, then we have:

$$T_{i+1,idle} = a \cdot T_{i-1,idle} + (1-a) \cdot T_{i,idle} \qquad (2)$$

where a is a smoothing ratio calculated by $a = \frac{2}{1+N}$. This allows us to give the more recent idle periods a higher weight in our exponential average, and thus a bigger influence on our next predicted idle period.

C. Logistic Regression Algorithm

The premise of the Markov Algorithm is promising, but we believe it under-performs in a wide range of applications since it only considers previous idle periods and some workloads may have idle periods influenced by preceding busy periods. We propose generalizing this model to account for more input information in order to determine if the current idle period is greater than threshold α.

Like the Markov Algorithm, we again, only have one input parameter into the algorithm, look-back period L, which we can tune for different workloads. We will use L for more than one calculation in this algorithm. For T_{idle}, the Logistic Regression Algorithm will calculate three features for the regression.

1) The first feature is a ratio $\frac{c_L}{L}$ where c_L is the number of idle intervals with $T_{j,idle} \geq \alpha$ among the last L idle intervals.
2) The second feature is the z-score of the length of $T_{i-1,busy}$ relative to the last L busy periods.
3) The third feature is the z-score of the length of $T_{i-2,idle}$ relative to the last L idle periods.

These three indicators are then fed into a logistic regression model and the model outputs the probability that the current idle period $T_{i,idle}$ will remain idle long enough to activate a shutdown sequence. If the probability is above a certain threshold, which we designate as 0.5, it signals the HDD device to transition to sleep mode. We assume that the model has been trained on some similarly distributed workload beforehand and will not actively learn data online while it is running. This is practical because we often have a good sense of how the workload is distributed as well as its patterns in specific applications.

The z-score of the previous idle and busy periods paired with the ratio of idle periods possibly provide the model with some interesting information about the trend of the running workload at the current period in time. For example, if the previous busy period had an abnormally low z-score (it was much shorter than average) and the ratio of idle intervals is far away from 1, we might expect that the workload has just finished a heavy period of requests and we might expect there to be a long uninterrupted idle period following.

D. L-Shape Algorithm

This was an interesting algorithm we found in the literature with a niche use-case. Researchers observed that in specific applications, short busy periods were frequently followed by a long idle period [1]. The name comes from the fact that plotting the duration of these busy periods against the duration of idle periods on a scatter-plot creates an "L-Shape" pattern with shorter busy periods corresponding to longer idle periods. In other words, for some time threshold $\theta > 0$, if $T_{i,busy} \leq \theta$ then $T_{i+1,idle} \geq \alpha$ with strong likelihood. This motivates an approach similar to the Timeout Algorithm where if $T_{i,busy} \leq \theta$ then immediately initiate a shutdown for the period $T_{i+1,idle}$.

III. Simulation Methodology

All of the simulator code used to obtain the results in this paper can be found in our GitHub repository [5].

A. Environment

We designed and implemented our own simulation suite in Python to model the DPM algorithms proposed in the previous section. The goal of these simulations was not to mimic a one-to-one correspondence with the actual hardware implementations governed by the ACPI standard, but rather, to compare the behavior of these algorithms against one another under a variety of workloads.

In our simulation environment, we abstracted the hardware components and their corresponding power states into a simplified model. This model allowed us to represent the dynamic behavior of devices transitioning between busy, idle, and sleeping states, akin to the D-states defined by ACPI. Additionally, we incorporated workload patterns (described in the next section) to simulate realistic usage scenarios, varying the duration of busy periods to evaluate algorithm performance across different operating conditions.

We simulated three different HDDs to interface with the DPM Algorithms with their statistics listed in Table I. Hard Drive A is exactly modeled after the IBM DTTA-350640 [1], Hard Drive B is exactly modeled after the Seagate ST3500630AS [3], and Hard Drive C is loosely modeled after the Seagate ST2000VN003 [6], [7].

B. Generating Workloads

We attempt to characterize and generate four different workloads for our simulation that seem plausible in real-world scenarios through using different statistical distributions to generate the time intervals in the workload. We use non-uniform distributions to model our workloads because we expect that data traffic, in practice, often follows trends and somewhat predictable behavior. Otherwise, the premise behind certain predictive DPM algorithms (Markov and Linear Regression) falls apart. We also didn't consider certain unpractical workloads which no DPM algorithm would perform reasonably on. For example, a workload where there is a request every other millisecond would prevent any kind of shut-down sequence. No DPM algorithm would be able to save energy consumption in such a workload since powering down and waking up take much more than a millisecond (on the order of seconds in fact), although there may be other, more effective, ways to manage power in this scenario.

We represent a workload as an array of discrete 1 millisecond time intervals with each interval representing either a period with incoming requests or a period with no requests (as depicted in Figure 1). For each workload, we arbitrarily chose the workload to run for 10 hours. While the HDD is active and handling requests, it operates solely in the busy state, which is out of the scope of the control of our DPM algorithms, so in regards to workload characterization, we focus on the length of busy periods rather than the intensity of data traffic during a busy period.

1) Normal Distribution (μ, σ): This is the most simplistic and unrealistic workload. We assume that busy and idle time periods last for roughly the same length of time and generally fall around some average time length μ (in milliseconds). Thus, we generate $T_i = N(\mu, \sigma)$ where N outputs a random time interval length based on the Normal Distribution with mean μ and standard deviation σ. T_i alternates between busy and idle as each interval is generated. If T_i represents a busy period, this means that for the next T_i milliseconds, the device is handling requests (1 request per millisecond) and no requests if T_i represents an idle period.

2) Exponential Distribution (μ, σ, λ): This workload aims to simulate a constant environment where there is, on average, one interval of requests interrupting every λ milliseconds. The benefit of using this distribution is that it is memory-less and thus, the probability of a request occurring in the future does not depend on how much time has already passed. Thus, for each idle period we calculate $T_{i,idle} = Exp(\lambda)$ which generates some random interval length based on the exponential distribution. Then, for each busy period, we calculate $T_{i,busy} = N(\mu, \sigma)$ as normal. This workload is closer to a real-world scenario where requests are expected to arrive regularly within some fixed time interval.

3) Short-Long Busy-Idle (μ, σ): This workload aims to simulate situations where short busy periods are generally followed by longer idle periods while long busy periods are generally followed by shorter idle periods. In other words, the length of an idle period is inversely proportional to the length of the preceding busy period. To model this characteristic, we first generate $T_{i,busy} = N(\mu, \sigma)$ and let $k = \frac{1}{T_{i,busy}}$. Then, we let $T_{i+1,idle} = N(k^2\mu, \sigma)$. The squared-factor for k was chosen to exaggerate the inverse relationship between consecutive busy and idle periods.

4) Periodic (μ, σ, k): This workload simulates a scenario where idle period lengths follow a periodic trend that alternate between increasing and decreasing length. In the real-world this may reflect changes in user-behavior as the time passes throughout the day transitioning from morning to afternoon to evening. This workload is of interest particularly for predictive algorithms such as the Markov and Logistic Regression Algorithms that typically excel in picking up patterns that are functions of time. To model this periodic behavior, we overlay a sine function on the current time interval stretched by a factor of $\frac{1}{k}$. Thus, we have $T_{i,idle} = \mu \cdot \sin(\frac{t}{k})$. The busy period intervals are generated the same as in the normally-distributed workload where $T_{i,busy} = N(\mu, \sigma)$.

C. Tools and Libraries Used

We used Python to implement the device features and algorithms as well as simulate these running together on the various workloads proposed above. Some third-party libraries used in the code include Pandas [8], NumPy [9], and Sklearn [10], [11].

IV. Evaluation

In this section, we split up our simulation results into three main categories, one for each hard disk drive simulated (Table I). For each hard drive, we ran all five DPM algorithms as well as a default algorithm on all four workloads proposed in the previous methodology section (Section 3). Each workload runs for a duration of exactly 10 hours and the default algorithm is simply keeping the disk either in active or standby state and not powering it up or down. We measured the energy efficiency of each algorithm in terms of watt-hours

Hard Drive	Storage (GB)	Sleeping Power (Watts)	Standby Power (Watts)	Active Power (Watts)	T_{sd} (Seconds)	T_{wu} (Seconds)	P_{sd} (Watts)	P_{wu} (Watts)
A	6.4	0.75	3.48	3.48	0.51	6.97	2.12	7.53
B	500	0.8	9.3	13	10	15	9.3	24
C	2000	0.25	2.8	3.7	10	8	12	30

TABLE I: Statistics of Model HDDs.

and the performance in terms of average wait time per request. That is, better performance corresponds to a lower waiting time per request. We assume that the default algorithm has no waiting time since it never transitions to sleeping mode whereas all other algorithms result in some additional waiting time when waking up (since they only know to wake up when the first request interrupts an idle period). Additionally, as we mentioned in Section 1, transitioning between the active and standby power states is in the order of microseconds [3], so we consider that transition time to be negligible.

A. *Hard Disk Drive A Results*

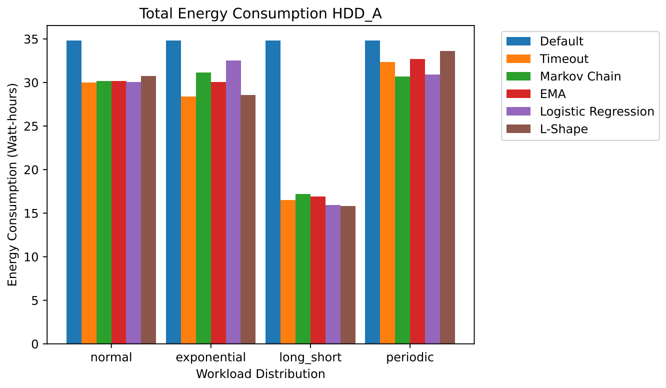

Figure 2: Energy consumption results for HDD A

Figure 2 shows the power consumption of all five DPM algorithms plus the default algorithm for each workload on HDD A. In all workloads, the default algorithm consumed more energy than all other DPM algorithms. For the Normally-Distributed workload, all DPM algorithms consumed roughly roughly $12-14\%$ less energy than the default algorithm with the Logistic Regression and Timeout algorithms consuming slightly less energy than the other DPM algorithms (30 Wh each). For the Exponentially-Distributed workload, the Timeout algorithm (28.4 Wh) and L-Shape algorithm (28.6 Wh) performed better than the other DPM algorithms by $7-12\%$ and the default algorithm by 18%. For the Long-Short workload, the L-Shape algorithm performed the best (15.8 Wh) out of all the other DPM algorithms and roughly 55% better than the default algorithm. The Logistic Regression algorithm was not far behind the L-Shape algorithm (15.9 Wh). For the Periodic workload, the Markov Chain algorithm (30.7 Wh) and the Logistic Regression algorithm (30.9 Wh) consumed between $6-9\%$ less energy than the other DPM algorithms and 12% less energy than the default algorithm.

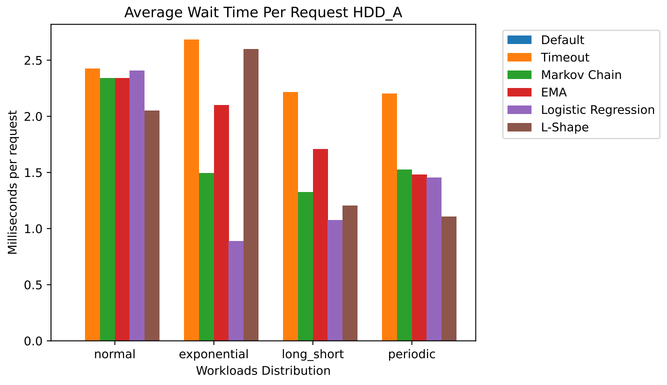

Figure 3: Performance results for HDD A

Figure 3 shows the average waiting time of all five DPM algorithms for each workload on HDD A. For the Normally-Distributed workload, the L-Shape algorithm had the lowest average waiting time per request out of all other DPM algorithms (2.05 ms per request). For the Exponentially-Distributed workload, the Logistic Regression algorithm had the lowest average waiting time per request (0.88 ms per request). For the Long-Short workload, the Logistic Regression algorithm had the lowest average waiting time per request (1.07 ms per request). For the Periodic workload, the L-Shape algorithm had the lowest average waiting time per request (1.11 ms per request).

B. *Hard Disk Drive B Results*

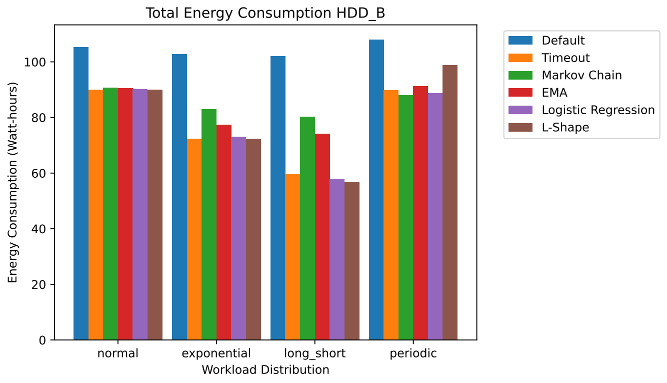

Figure 4: Energy consumption results for HDD B

Figure 4 shows the power consumption of all five DPM algorithms plus the default algorithm for each workload on HDD B. In all workloads, the default algorithm consumed more energy than all other DPM algorithms. For the Normally-Distributed workload, all DPM algorithms consumed roughly roughly $13.8 - 14.6\%$ less energy than the default algorithm with the Timeout and L-Shape algorithms consuming slightly less energy than the other DPM algorithms (89.9 Wh each). For the Exponentially-Distributed workload, the Timeout algorithm and L-Shape algorithm (72.32 Wh each) performed better than the other DPM algorithms by $6.5 - 12.8\%$ and the default algorithm by 29.6%. For the Long-Short workload, the L-Shape algorithm performed the best (56.6 Wh) out of all the other DPM algorithms and roughly 44.5% better than the default algorithm. The Logistic Regression algorithm was not far behind the L-Shape algorithm (57.8 Wh). For the Periodic workload, the Markov Chain algorithm (87.9 Wh) and the Logistic Regression algorithm (88.7 Wh) consumed between $2 - 11\%$ less energy than the other DPM algorithms and 18% less energy than the default algorithm.

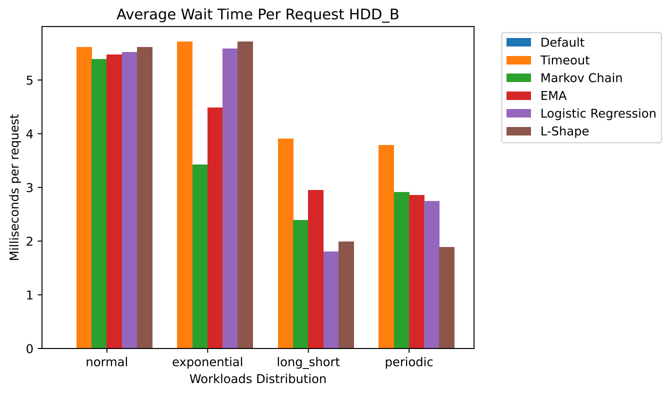

Figure 5: Performance results for HDD B

Figure 3 shows the average waiting time of all five DPM algorithms for each workload on HDD B. For the Normally-Distributed workload, the Markov Chain algorithm had the lowest average waiting time per request out of all other DPM algorithms (5.38 ms per request), with the other algorithms not far behind. For the Exponentially-Distributed workload, the Markov Chain algorithm, again, had the lowest average waiting time per request (3.42 ms per request). For the Long-Short workload, the Logistic Regression algorithm had the lowest average waiting time per request (1.80 ms per request). For the Periodic workload, the L-Shape algorithm had the lowest average waiting time per request (1.89 ms per request).

C. Hard Disk Drive C Results

Figure 6 shows the power consumption of all five DPM algorithms plus the default algorithm for each workload on HDD C. In all workloads, the default algorithm consumed more energy than all other DPM algorithms. For the Normally-Distributed workload, all DPM algorithms consumed roughly roughly $14 - 16\%$ less energy than the default algorithm with the Timeout and L-Shape algorithms consuming slightly less energy than the other DPM algorithms (24 Wh each). For the Exponentially-Distributed workload, the Timeout algorithm

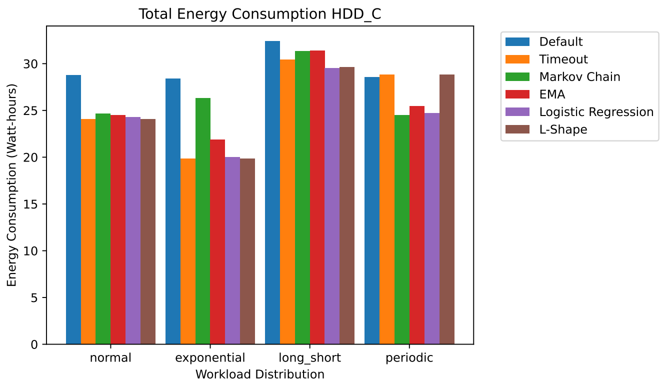

Figure 6: Energy consumption results for HDD C

and L-Shape algorithm (19.9 Wh each) performed better than the other DPM algorithms by $9 - 24.5\%$ and the default algorithm by 30%. For the Long-Short workload, the Logistic Regression algorithm performed the best (29.5 Wh) out of all the other DPM algorithms and roughly 8.9% better than the default algorithm. The L-Shape algorithm was not far behind the Logistic Regression algorithm (29.6 Wh). For the Periodic workload, the Markov Chain algorithm (24.5 Wh) and the Logistic Regression algorithm (24.7 Wh) consumed between $3.7 - 15\%$ less energy than the other DPM algorithms and 14.3% less energy than the default algorithm.

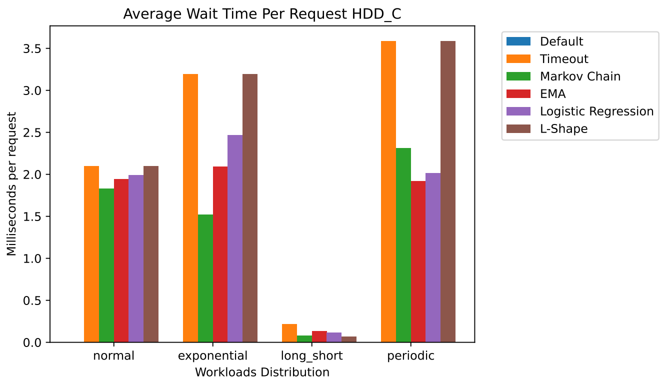

Figure 7: Performance results for HDD C

Figure 7 shows the average waiting time of all five DPM algorithms for each workload on HDD C. For the Normally-Distributed workload, the Markov Chain algorithm had the lowest average waiting time per request out of all other DPM algorithms (1.83 ms per request). For the Exponentially-Distributed workload, the Markov Chain algorithm, again, had the lowest average waiting time per request (1.52 ms per request). For the Long-Short workload, the L-Shape algorithm had the lowest average waiting time per request (0.07 ms per request). For the Periodic workload, the Exponential Moving Average algorithm had the lowest average waiting time per request (1.92 ms per request).

V. Conclusion

A. Results Summary

Our results suggest that in general, all of the DPM algorithms proposed were more energy-efficient than not having any power management at all, at least in the context of the workloads we proposed. For the Normally-Distributed workload, all of the DPM algorithms had relatively similar performance across all hard-disk drives. Out of the three predictive algorithms - Exponential Moving Average, Markov Chain, and Logistic Regression - the Logistic Regression algorithm seemed to perform the best on all hard disks across all workloads. Additionally, the average wait time per request for the Logistic Regression model was overall lower than those of the simplistic Timeout and L-Shape algorithms. In general, predictive algorithms saved more energy than the other algorithms on the workload distributions which followed time-series trends (Long-Short and Periodic workloads). This aligns with our initial prediction since the underlying mechanisms powering these algorithms are used in other data specifically for following time-series trends and patterns. On the Long-Short workload distribution, the L-Shape algorithm performed the best out of all the other algorithms in terms of energy efficiency and performance, which is also what we expected and found in the pre-existing literature. However, the L-Shape algorithm also had relatively good energy efficiency on the normal and exponential distribution workloads, which is suprising since the idle periods in these workloads do not have any correlation with the preceding busy periods; although, the L-Shape also exhibited poor performance compared to the other DPM algorithms on these workloads, so this may have just been due to the algorithm calling for frequent shut-downs.

B. Future Considerations

There are many paths that these results could be taken in the future. Our simulation is quite crude and does not directly interact with any hardware, which is an area that can be further explored. One major component of HDDs which we did not factor into account was the rotational speeds of different hard disks under varying request intensities. This factor would not impact the results in our study significantly since our workloads are quite simplistic. Our current workloads also assume that there is a fixed number of incoming requests per discrete time interval. Further analysis could be done on wider ranges of workloads with more complex distributions. More realistic workloads would have varying requests throughput during busy periods, which would likely result in the rotational speeds of the hard disk impacting energy consumption. Additionally, the performance of these algorithms may change significantly when run on physical hardware tests and under less idealistic workloads, especially considering that physical disks have varying rotational speeds which could impact wake up and shut down times. Additionally, our predictive Logistic Regression algorithm performed relatively well (often better than the other algorithms) in this study. Future studies could experiment with more predictive mechanisms involving machine learning, especially if the workloads of interest exhibit strongly predictable patterns.

References

[1] Y.-H. Lu, E.-Y. Chung, T. Šimunić, L. Benini, and G. De Micheli, "Quantitative comparison of power management algorithms," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 20–26. [Online]. Available: https://doi.org/10.1145/343647.343688

[2] "Device power states," https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/device-power-states, 2021.

[3] E. Otoo, D. Kotem, and T. Shih-Chiang, "Energy smart management of scientific data," https://escholarship.org/uc/item/30k0p2mp, 2009.

[4] Q. Wu and G.-z. Xiong, "Why simple timeout strategies work perfectly in practice?" in *Embedded Software and Systems*, Z. Wu, C. Chen, M. Guo, and J. Bu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 468–473.

[5] Y. Lim, "Hdd-energy-simulator," https://github.com/yellowfish15/HDD-Energy-Simulator, 2024.

[6] "Seagate ironwolf + rescue 1-4, 6, 8tb - product manual," https://www.seagate.com/content/dam/seagate/migrated-assets/www-content/product-content/ironwolf/en-us/docs/100807039r.pdf, 2022.

[7] "Data sheet," https://www.seagate.com/www-content/datasheets/pdfs/ironwolf-12tbDS1904-9-1707US-en_US.pdf, 2017.

[8] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[9] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.