Quantitative Comparison of Power Management Algorithms

Yung-Hsiang Lu, Eui-Young Chung, Tajana Šimunić, †Luca Benini, Giovanni De Micheli
Computer Systems Laboratory, Stanford University, USA
{luyung, eychung, tajana, nanni}@stanford.edu
†Dip. Informatica, Elettronica, Sistemistica, Università di Bologna, Italy
†lbenini@deis.unibo.it

Abstract

Dynamic power management saves power by shutting down idle devices. Several management algorithms have been proposed and demonstrated effective in certain applications. We quantitatively compare the power saving and performance impact of these algorithms on hard disks of a desktop and a notebook computers. This paper has three contributions. First, we build a framework in Windows NT to implement power managers running realistic workloads and directly interacting with users. Second, we define performance degradation that reflects user perception. Finally, we compare power saving and performance of existing algorithms and analyze the difference.

1. Introduction

Dynamic power management (DPM) is an effective approach to reduce power consumption without significantly degrading performance [2]. DPM shuts down devices when they are not being used and wakes them up when necessary. When a device is not used, it is called *idle*; otherwise, it is called *busy*. DPM algorithms observe request patterns and predict the length of idle periods. Idle periods can be defined in different ways [8]. In this paper, we consider an idle period as "time with no requests waiting for service". The device is in a *working* state when it can serve requests with higher power consumption, P_w (Table 1 summaries all symbols.). The device is *sleeping* when it consumes less power, P_s ($P_s < P_w$), and cannot serve requests. Shutting down and waking up a device usually cause performance degradation and require extra energy. Therefore, DPM algorithms shut down a device only when an idle period is long enough to justify performance degradation and state-transition energy.

This paper compares DPM algorithms for controlling the power states of hard disk drives on a desktop and a notebook computers. Hard disks are of particular interest for power management due to three reasons. First, hard disks may consume up to 20% of total energy in a computer [7]. Recent studies find that hard disks will remain major power consumers in the near future [10] [12]. Second, hard disks have large performance and power overhead because of mechanical inertia. Spinning down or up disk plates takes several seconds, equivalent to hundreds of millions of instructions in modern computers. Finally, hard disks are not always needed when computers are running if the physical memory contains all the information needed; for example, caching may avoid unnecessary spin-ups [5].

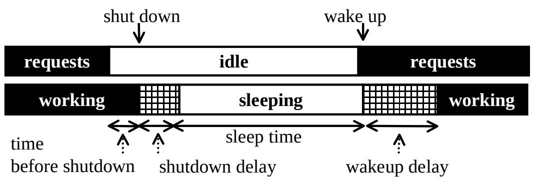

Figure 1: State Transitions

Our study has three major contributions: a framework to implement power managers (PM), definition of user-perceived performance, and quantitative comparison of algorithms. In the past, DPM algorithms were evaluated mainly by simulation. In contrast, we build a framework for comparing DPM algorithms running realistic workloads on a commercial operating system, Windows NT, and directly interacting with users. Consequently, users can perceive performance degradation while power is saved. Although some algorithms support multiple sleeping states, we use only one sleeping state in the implementation as a common denominator for fair comparison.

2. Foundation for Algorithm Comparison

2.1. Idle Periods

Figure 1 shows requests and the power states of a device. An idle period ($T_{idle}[i]$) occurs between two periods with requests, also called busy periods. The device is shut down after it enters $T_{idle}[i]$; it does not enter the sleeping state immediately due to shutdown delay (T_{sd}). Some DPM algorithms do not shut down a device immediately after an idle period starts; instead, they wait until they are confident that $T_{idle}[i]$ is long enough to save power. This waiting time is called "time before shutdown" (T_{bs}). Later, when requests arrive, the device is woken up and enters the working state after wakeup delay (T_{wu}). The energy consumed during the shutdown and wakeup delay are denoted as E_{sd} and E_{wu}.

2.2. Break-Even Time

As we explained earlier, in order to compensate the shutdown and wakeup overhead, a device has to stay in the sleeping state long enough. We call this minimum duration the *minimum sleeping time* (T_{ms}). Since $E_{sd}+E_{wu}+P_s \cdot T_{ms} = P_w \cdot (T_{ms}+T_{sd}+T_{wu})$, we can find T_{ms} in Equation 1. The minimum length of an idle period to save energy is the *break-even time* for idle period (T_{be}). It includes the shutdown and wakeup delay in addition to T_{ms}, so $T_{be} = T_{ms}+T_{sd}+T_{wu}$. A shutdown command saves power only if $T_{idle}[i] > T_{bs} + T_{be}$.

$$T_{ms} = \frac{E_{sd} + E_{wu} - P_w \cdot (T_{sd} + T_{wu})}{P_w - P_s} \qquad (1)$$

$$T_{be} = \frac{E_{sd} + E_{wu} - P_s \cdot (T_{sd} + T_{wu})}{P_w - P_s} \qquad (2)$$

2.3. Performance Metrics

DPM trades off between power and performance. Several ways were proposed to quantify performance, such as total or average waiting time; however, total or average waiting time can be misleading. Using these metrics, a system that causes a total of 50 seconds of waiting in five hours is better than one that causes 60 seconds of waiting in five hours. However, if the former requires a user to wait for 40 seconds in a one-minute period while the latter requires at most 10 seconds in one minute, most users think the second system has better performance. Traditional total waiting time does not reflect this discrepancy. Studies in psychology show that waiting time can affect user behavior [18]. In this paper, we use two objective performance metrics: long waiting or repetitive waiting "within short time intervals" to reflect the perception of performance degradation.

Symbol	Meaning
T_{sd}	shutdown delay
T_{wu}	wakeup delay
T_{be}	break-even time for an idle period
T_{ms}	minimum sleeping time to save energy
T_{bs}	time before shutdown
T_{ss}	average time in sleeping state
$T_{idle}[i]$	current idle period, candidate for shutdown
$t_{idle}[i]$	predicted value of $T_{idle}[i]$
$T_{busy}[i]$	busy period before $T_{idle}[i]$
$W_S[i]$	starting time of a waiting period
$W_E[i]$	ending time of a waiting period
τ	timeout value
E_{sd}	energy to shut down
E_{wu}	energy to wake up
P_s	power in sleeping state
P_w	power in working state
N_{sd}	number of shutdowns
N_{wd}	number of wrong shutdowns

Table 1: Symbols and Meanings

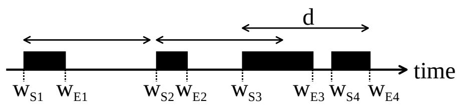

Figure 2: Waiting due to Power Management

Figure 2 gives an example of four waiting periods. A waiting period (W) starts at W_S and ends at W_E. Our first performance metric is the largest total waiting time in a duration of length d called W_d. It is obtained by finding the sum of waiting time in a sliding window of size d. A window may contain multiple waiting periods, such as $W[3]$ and $W[4]$ in the third window; a window can also contain part of a waiting period, such as $W[3]$ by the second window. In general, W_d can be calculated by Equation 3.

$$W_d = \max_t \sum_{\substack{i \text{ such that} \\ W_S[i] \geq t \\ W_E[i] \leq t+d}} (W_E[i] - W_S[i]) \qquad (3)$$

This equation finds all waiting periods inside a window of size d and calculates the sum of these periods. By adjusting t, the starting point of a window, it then finds the window that contains the longest total waiting time.

If a waiting period is partially covered, we consider only the part within the window; for simplicity, equation 3 does not include partially covered waiting periods.

The second performance metric is the longest shutdown sequence in which the time between two adjacent shutdowns is less than a threshold th. This metrics measures the number of consecutive waiting periods that are close and cause delay repetitively. It is the largest m for which there is a sequence $W[i], W[i+1], \dots W[i+m]$ such that the following conditions hold:

$$\begin{array}{ll} W_S[k+1] & -W_E[k] \leq th \\ W_S[i] & -W_E[i-1] > th \\ W_S[i+m+1] & -W_E[i+m] > th \end{array} \qquad (4)$$

where $k \in [i, i+m-1]$

3. Algorithms and Parameter Selection

We compare algorithms originally designed for various applications, such as X-servers and hard disks; these algorithms are listed in Table 2. They assume different characteristics of applications. For example, T_{sd} and T_{wu} are fairly small for X-server but large for hard disks due to mechanic inertia. We compare them in the same environment and study the deviations from their originally intended applications. In this section, we briefly explain these algorithm and the parameters recommended by their authors.

TIMEOUT: Timeout algorithms are simple and widely used; they assume that if a device is idle longer than τ, it will remain idle for a long time ($T_{idle} > \tau \Rightarrow T_{idle} > \tau + T_{be}$) with up to 95% confidence level [8]. Timeout algorithms wait for τ before shutdown, so $T_{bs} = \tau$. Microsoft Windows Control Panel allows users to set τ as small as 60 seconds. We use a filter device driver [14] and can choose any τ value; we use thirty seconds and two minutes in this study.

The first adaptive timeout (ATO1) algorithm adjusts τ by considering the value of $\frac{T_{idle}[i-1]}{T_{wu}}$ [6]. When the ratio is small, τ increases; when the ratio is large, τ decreases. We start with 30 seconds for τ and use $(\alpha_m, \beta_m, \rho) = (1.5, 0.5, 0.1)$ because they produced better results. Golding suggests updating τ asymmetrically: increasing τ by one second or decreasing by half [8] (ATO2). In our experiments for [8], τ is limited to 1 to 120 seconds. Another approach adjusts τ according to $T_{busy}[i]$ [13] (ATO3). If $T_{busy}[i]$ is small, τ decreases; if $T_{busy}[i]$ is large, τ increases. We use 120 seconds for the initial value of τ with 1 Hz sampling and 2 seconds for the adjustment factor.

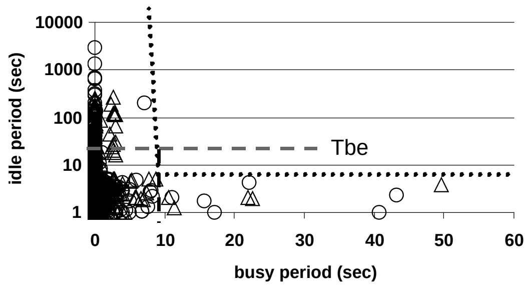

Figure 3: $T_{busy}[i]$ vs. $T_{idle}[i]$ of two different users in our experiments.

Algorithm	Features	Applications
[6] (ATO1)	adaptive timeout	hard disk
[8] (ATO2)	adaptive timeout	hard disk
[13] (ATO3)	adaptive timeout	hard disk
[17] (LS)	L-shape	X-server
[9] (EA)	exponential average	telnet
[15] (DM)	discrete-time Markov	hard disk
[3] (SW)	sliding windows	hard disk
[16] (CM)	continuous time Markov	real-time inputs
[20] (SM)	time-index semi-Markov	hard disk
[11] (CA)	competitive	spin-block
[4] (LT)	learning tree	hard disk

Table 2: DPM Algorithms Compared

L-SHAPE: If a short busy period is frequently followed by a long idle period, their scatter plot will form an "L-shape" (LS) as shown in Figure 3. If a busy period is short enough, the following idle period is likely to be long ($T_{idle}[i] \geq T_{be} \Leftrightarrow T_{busy}[i] \leq T_{threshold}$). Consequently, when request patterns form an L-shape, the device should be shut down after a short busy period [17].

EXPONENTIAL AVERAGE: Hwang and Wu use the predicted and the actual lengths of the previous idle period to predict the length of the current idle period [9]. They use exponential average (EA) for predicting T_{idle} by $t_{idle}[i] = a \cdot T_{idle}[i-1] + (1-a) \cdot t_{idle}[i-1]$, which is equivalent to

$$t_{idle}[i] = (1-a)^{i+1} t_{idle}[0] + \sum_{k=0}^{i} a(1-a)^k T_{idle}[i-k]$$

where $t_{idle}[i]$ is the prediction of $T_{idle}[i]$. It is an average of previous idle periods with exponential weights. If $t_{idle}[i] > T_{be}$, the device is shut down. This algorithm limits $t_{idle}[i]$ such that it cannot exceed $c \cdot t_{idle}[i-1]$ where c is a constant greater than one. We use 0.5 for a and 2 for c; we ignore all $T_{idle}[i-1] < 0.1$ second so that t_{idle} are not affected by short idle periods. This is an on-line filtering because $T_{idle}[i-1]$ is known before computing $t_{idle}[i]$. We use 0.1 second so that our implementation has only negligible difference from the original algorithm.

STOCHASTIC MODEL: Power management can be solved as an optimization problem when devices and requests are modeled as stochastic processes. This approach formulates power management as a constrained optimization problem; it provides the flexibility to trade off between power and performance. For example, power management can be modeled as a discrete-time Markov decision process (DM) [15]. The algorithm assumes stationary geometric distribution of request arrivals. It is extended to handle non-stationary requests in [3]. Non-stationarity is captured by sliding windows (SW); the algorithm interpolates pre-optimized look-up tables for shutdown decisions. The shutdown decision is evaluated each period, even when the device is in the sleeping state, thus causing computation overhead. For example, for a 10 W processor, SW could waste as much as 1800 J of energy during a 30-minute idle period just due to reevaluating shutdown decisions.

By modeling a device as a continuous-time Markov process, PM can change power states upon event occurrences, such as "request queue empty" events, instead of at discrete time intervals [16] (CM). State transitions are assumed to follow exponential distributions. This approach makes a decision as soon as certain events happen. Our measurements show no significant power saving since the algorithm tends to shut down too soon, thus incurring large wakeup costs and sometimes even missing shutting down on long idle periods.

Both discrete and continuous-time approaches model request arrivals and the power state transitions using memoryless distributions, which is not accurate in real situations. Semi-Markov approach [19] models transition times between power states with uniform distributions. As the request arrivals are better modeled using Pareto distribution, the semi-Markov approach is further generalized with time-indexed semi-Markov models in [20] (SM). Optimal power saving based on this model reevaluates the decisions during idle periods until either an request occurs or the device is shut down. When the device is sleeping, no decision evaluation is needed. Therefore, this algorithm has low computation overhead and shows the largest power saving in our study.

COMPETITIVE ALGORITHM: A "c-competitive" on-line algorithm (CA) can find a solution with cost less than c times the cost generated by an optimal off-line algorithm. It can be proven that 2-competitive power saving is achievable if $\tau = T_{be}$ [11]. In other words, this algorithm consumes at most twice the minimum power consumed by an off-line PM.

LEARNING TREE: Adaptive learning trees (LT) transform sequences of idle periods into discrete events and store them into tree nodes [4]. This algorithm predicts idle periods using finite-state machines similar to branch prediction used in microprocessors and selects a path which resembles previous idle periods. At the beginning of an idle period, it determines an appropriate sleeping state; this algorithm is capable of controlling multiple sleeping states.

4. Experiment Results

4.1. Experiment Environment

We use an environment built specifically for implementing and evaluating DPM algorithms [14]. It consists of two ACPI-compliant [1] computers. The first is a Pentium II desktop computer with an IBM DTTA 350640 hard disk; the other is a Pentium II notebook computer with a Fujitsu MHF 2043AT hard disk. Both are running a beta version of Microsoft Windows NT V5. We implemented filter drivers for each algorithm to control the power states of the hard disks, to record disk accesses and to analyze the performance impact and the power management overhead of each algorithm. Table 3 shows the parameters of the disks. T_{be} is 17.6 and 5.43 second for the IBM and Fujitsu disks respectively.

4.2. PM Overhead and Power Consumption

We recorded disk accesses for two users, one developing C programs and the other making presentation slides. These traces include disk accesses from user requests and operating system activities. Then the traces are replayed taking approximately 11 hours for each algorithm. We find that all algorithms spend less than 1% of computation time on power managers; hence, power management pays off when it is able to effectively reduce power consumption. These algorithms are compared by five figures:

- Power consumption (P), unit: Watt.

Model	P_s Watt	P_w Watt	T_{sd} sec	E_{sd} J	T_{wu} sec	E_{wu} J
IBM	0.75	3.48	0.51	1.08	6.97	52.5
Fujitsu	0.13	0.95	0.67	0.36	1.61	4.39

Table 3: Disk Parameters

Algorithm	P	N_{sd}	N_{wd}	T_{ss}	T_{bs}
	desktop				
off-line	1.64	164	0	166	0
SM	1.92	156	25	147	18.2
CA	1.94	160	15	142	17.6
SW	1.97	168	26	134	18.7
$\tau = 30$	2.05	147	18	142	30.0
LT	2.07	379	232	62	5.7
ATO3	2.09	147	26	138	29.9
ATO1	2.19	141	37	135	27.6
ATO2	2.22	595	430	41	4.1
$\tau = 120$	2.52	55	3	238	120.0
DM	2.60	105	39	130	48.9
EA	2.99	595	503	30	7.6
always-on	3.48	-	-	-	-
	notebook				
off-line	0.33	250	0	118	0
SM	0.40	326	76	81	8.0
SW	0.43	191	28	127	13.4
CA	0.44	323	64	79	5.4
LT	0.46	437	217	56	6.1
ATO1	0.47	273	73	88	12.4
EA	0.50	623	427	37	3.0
$\tau = 30$	0.51	139	7	157	30.0
ATO3	0.52	196	48	109	24.5
DM	0.62	173	54	102	35.2
ATO2	0.64	881	644	19	2.3
$\tau = 120$	0.67	55	0	255	120.0
always-on	0.95	-	-	-	-

Table 4: Algorithm Comparison

- Number of shutdowns (N_{sd}).
- Number of wrong shutdowns (N_{wd}) that have sleeping time shorter than T_{ms} and actually waste energy.
- Average time in sleeping state (T_{ss}), unit: second
- Average time before shutdown (T_{bs}), unit: second

Table 4 orders the algorithms by power consumption. In this table, smaller values are better except T_{ss}. The first row contains the minimum power consumption without performance degradation; it is generated off-line with full knowledge about future requests. The last row shows the power consumption if no power management is applied. This table shows that SM, CA and SW can save nearly 50% of power on both platforms. Even though they have close power consumption on the mobile disk, they differ significantly in performance. CA and SM have more than twice wrong shutdowns (N_{wd}) compared with SW. For algorithms with similar power consumption, performance is an important factor for evaluation. The total waiting time is approximately proportional to the total number of shutdowns (N_{sd}). Users may notice substantially different performance degradation even for two algorithms that have similar values of N_{sd} if some wrong shutdowns occur repetitively within a short time interval.

4.3. Performance Measurement

Figure 4 (next page) draws the worst waiting time (W_d) for d between one to ten minutes. It shows that, in the worst case, CA requires users to wait for 98 seconds in a 10-minute duration on the desktop hard disk. The bottom of the figure is the waiting time by percentage. When the window size increases the percentage of waiting time decreases for all algorithms. When the window size is small, such as one minute, some algorithms may require users to wait for more than 50% of the time. This demonstrates the importance to measure the worst-case performance for small d. Traditional performance metric using the total waiting time cannot provide enough information for determining user perception of performance degradation. This figure has several "jumps" as d increases because the worst-case waiting time may change from one window to another. Figure 4 also shows that the waiting time is considerably shorter on the mobile hard disk. Figure 5 plots the longest shutdown sequence in which the time between two adjacent shutdowns is shorter than a threshold. In this figure, the X-axis is the threshold value and the Y-axis is the lengths of sequences. The arrow indicates that EA has a sequence of 27 waiting periods with less than one minute between two shutdowns. Users perceive delays every minute or even more frequently for 27 times.

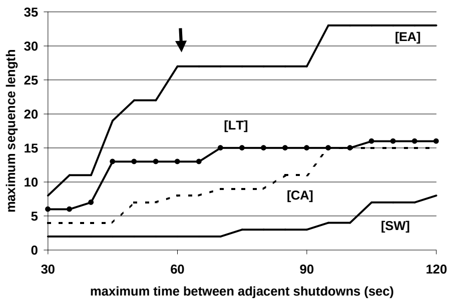

Figure 5: Maximum Length of Shutdown Sequences

5. Discussion

5.1. Correlation of Adjacent Busy and Idle Periods

LS uses the length of the previous busy period to predict the length of the current idle period by "L-shape" approximation. Figure 3 shows two traces we collected;

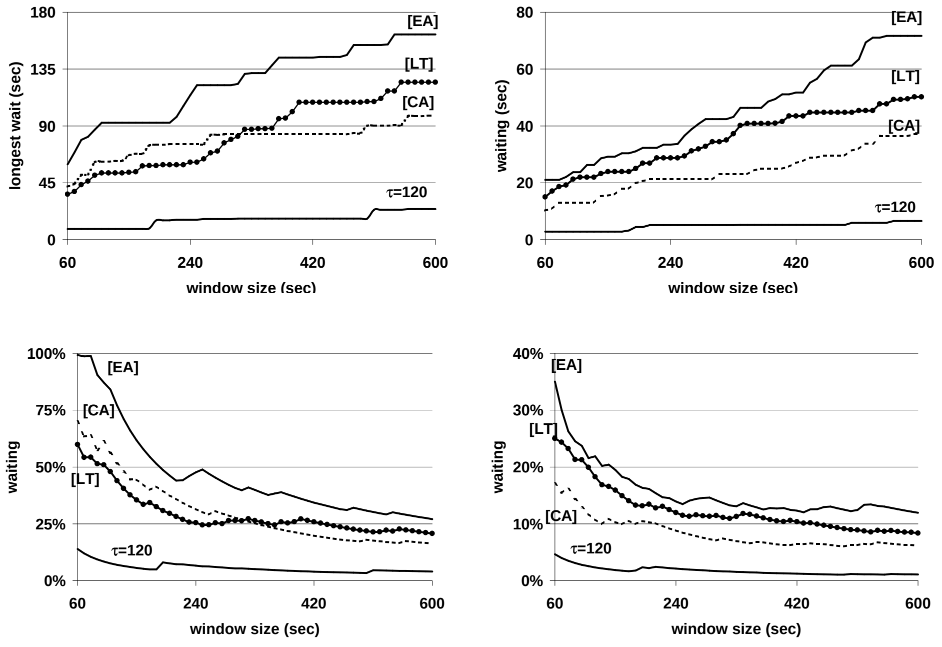

Figure 4: W_d on Desktop (left) and Mobile (right) Disks

the dash line encloses most requests into an L shape. If we consider only idle periods longer than 10 seconds and redraw the distribution in Figure 6, we find this approximation requires refinement. One long idle period (197 seconds, pointed by the arrow) follows a long busy period; L-shape algorithm will keep a hard disk in the working state during this period. Furthermore, a large group of short idle periods follow short busy periods enclosed by the circle. L-shape approximation is not sufficient to determine the length of an idle period.

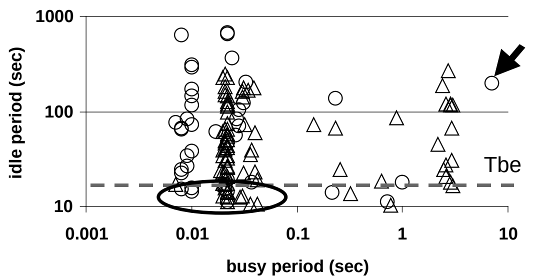

Figure 6: $T_{busy}[i]$ vs. $T_{idle}[i]$ (Zoomed In)

5.2. State Transition Time

Table 3 lists the average of T_{sd} and T_{wu} . Most algorithms treat them as constants for simulation; in reality, they are not constants. Figure 7 shows significant variation of T_{wu}. For the desktop hard disk, the average of T_{wu} is 6.97 seconds and the standard deviation is 0.65 second or 9.25%. For the mobile disk, T_{wu} is even more widely distributed; it is inappropriate to use a single value for T_{wu}. Markov models in DM are inexact approximations because our experiments show that T_{wu} is not exponentially distributed.

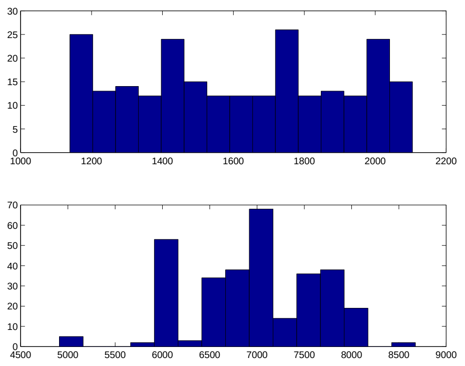
Figure 7: T_{wu} (ms) for Mobile (top) and Desktop Disks

5.3. Successive Wrong Shutdowns after a Long T_{idle}

EA assumes that a long idle period is followed by other long idle periods and a short idle period is followed by other short idle periods. When a long idle period is followed by short idle periods, however, the performance deteriorates considerably and generates long sequences in Figure 5. Consider an example, a user leaves the office and creates an idle periods of one hour. When the user returns, the hard disk will be shut down repetitively and made unusable during the first minute. In order to remedy this problem, a desirable algorithm should change its prediction sooner once successive wrong shutdowns happen. We do not use predictive wakeup because it consumes 96% more energy on the mobile disk without significant performance improvement.

5.4. Algorithm Ordering and Device Parameters

Compared to the desktop hard disk, $T_{sd} + T_{wu}$ and $E_{sd} + E_{wu}$ are 70% and 91% less on the mobile hard disk. As a result, DPM algorithms can be more aggressive to shut down the mobile disk in order to save energy since the penalty is considerably less. The ordering on the desktop computer in Table 4 is similar to the ordering on the notebook computer except EA where it consumes less power than most other algorithms. This algorithm was originally designed for X-server and telnet, which have small shutdown and wakeup overhead because no mechanic device is involved. This example shows the importance to tune an algorithm for its intended application.

6. Conclusions

We built a framework to compare power management algorithms. To our knowledge, this is the first time DPM algorithms are compared while running realistic workloads and interacting with users. Our experience shows that waiting in a short duration and consecutive waiting, instead of total waiting time, directly affect user perception of performance. We quantitatively define performance metrics that reflect user experience. Our study concentrates on comparing power management algorithms for control the power states of computer hard disks and discusses several key issues in designing DPM algorithms.

7. Acknowledgments

This work is supported by MARCO, ARPA, GSRC, and NSF (CCR-9901190).

8. References

[1] ACPI. http://www.teleport.com/˜acpi.

[2] L. Benini, A. Bogliolo, and G. D. Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Transactions on VLSI Systems*, March 2000.

[3] E.-Y. Chung, L. Benini, A. Bogliolo, and G. D. Micheli. Dynamic Power Management for Non-Stationary Service Requests. In *Design Automation and Test in Europe*, pages 77–81, 1999.

[4] E.-Y. Chung, L. Benini, and G. D. Micheli. Dynamic power management using adaptive learning tree. In *International Conference on Computer-Aided Design*, pages 274–279, 1999.

[5] F. Douglis, R. Cáceres, F. Kaashoek, K. Li, B. Marsh, and J. A. Tauber. Storage Alternatives for Mobile Computers. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 25–37, 1994.

[6] F. Douglis, P. Krishnan, and B. Bershad. Adaptive Disk Spin-Down Policies for Mobile Computers. In *Computing Systems*, volume 8, pages 381–413, 1995.

[7] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the Power-Hungry Disk. In *USENIX Winter Conference*, pages 293–306, 1994.

[8] R. Golding, P. Bosch, and J. Wilkes. Idleness is not Sloth. In *USENIX Winter Conference*, pages 201–212, 1995.

[9] C.-H. Hwang and A. C. Wu. A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation. In *International Conference on Computer-Aided Design*, pages 28–32, 1997.

[10] Intel. Mobile Power Guide ’99.

[11] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive Randomized Algorithms for Nonuniform Problems. *Algorithmica*, 11(6):542–571, June 1994.

[12] J. R. Lorch and A. J. Smith. Software Strategies for Portable Computer Energy Management. *IEEE Personal Communications*, 5(3):60–73, June 1998.

[13] Y.-H. Lu and G. D. Micheli. Adaptive Hard Disk Power Management on Personal Computers. In *Great Lakes Symposium on VLSI*, pages 50–53, 1999.

[14] Y.-H. Lu, T. Šimunić, and G. D. Micheli. Software Controlled Power Management. In *International Workshop on Hardware/Software Codesign*, pages 157–161, 1999.

[15] G. A. Paleologo, L. Benini, A. Bogliolo, and G. D. Micheli. Policy Optimization for Dynamic Power Management. In *Design Automation Conference*, pages 182–187, 1998.

[16] Q. Qiu and M. Pedram. Dynamic Power Management Based on Continuous-Time Markov Decision Processes. In *Design Automation Conference*, pages 555–561, 1999.

[17] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen. Predictive System Shutdown and Other Architecture Techniques for Energy Efficient Programmable Computation. *IEEE Transactions on VLSI Systems*, 4(1):42–55, March 1996.

[18] M. T. Stokes. *Time in Human-Computer Interaction.* PhD. Thesis, Psychology Department, Texas Tech University, 1991.

[19] T. Šimunić, L. Benini, and G. D. Micheli. Event-Driven Power Management of Portable Systems. In *International Symposium on System Synthesis*, pages 18–23, 1999.

[20] T. Šimunić, L. Benini, and G. D. Micheli. Dynamic Power Management of Laptop Hard Disk. In *Design Automation and Test in Europe*, 2000.