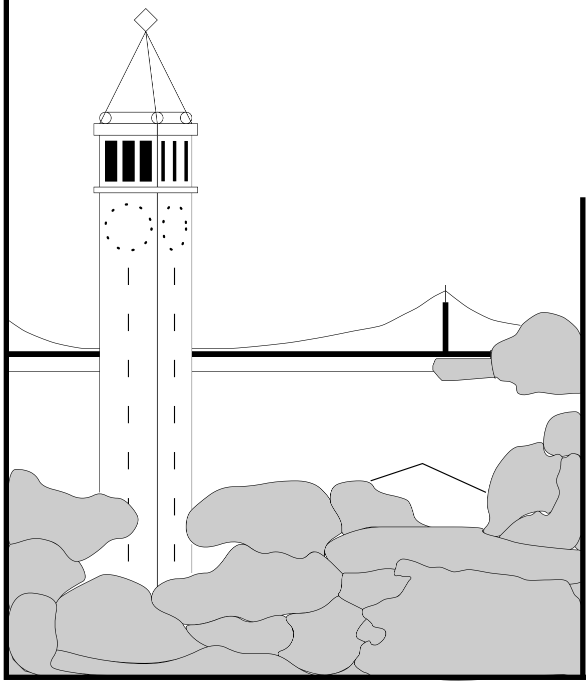
Software Strategies for Portable Computer Energy Management

Jacob R. Lorch *Alan Jay Smith*



Report No. UCB/CSD-97-949

May 1997

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Software Strategies for Portable Computer Energy Management*

Jacob R. Lorch† Alan Jay Smith†

May 30, 1997

Abstract

Limiting the energy consumption of computers, especially portables, is becoming increasingly important. Thus, new energy-saving computer components and architectures have been and continue to be developed. Many architectural features have both high performance and low power modes, with the mode selection under software control. The problem is to minimize energy consumption while not significantly impacting the effective performance. We group the software control issues as follows: *transition*, *load-change*, and *adaptation*. The transition problem is deciding when to switch to low-power, reduced-functionality modes. The load-change problem is determining how to modify the load on a component so that it can make further use of its low-power modes. The adaptation problem is how to create software that allows components to be used in novel, power-saving ways. We survey implemented and proposed solutions to software energy management issues created by existing and suggested hardware innovations.

1 Introduction

Limiting energy consumption has become an important aspect of modern computing. The most important reason for this is the growing use of portable computers, which have limited battery capacities. Another reason is that high energy consumption by desktop computers translates into heat, fan noise, and expense. One way to reduce energy consumption is to simply use components that consume less power. Another way is to use components that can enter low-power modes by temporarily reducing their speed or functionality. This paper will discuss the software problems that arise from such hardware features, and what solutions have been proposed to deal with these issues. The aim of this paper is not to discuss hardware techniques for reducing power, but to discuss software techniques for taking advantage of low-power hardware that has already been designed.

Category	**Description**
Transition	When should a component switch between modes?
Load-change	How can a component's functionality needs be modified so it can be put in low-power modes more often?
Adaptation	How can software permit novel, power-saving uses of components?

Table 1: Categories of energy-related software problems

We classify the software problems created by power-saving hardware features into three categories: transition, load-change, and adaptation. The transition problem involves answering the question, "When should a component switch from one mode to another?" The load-change problem involves answering the question, "How can the functionality needed from a component be modified so that it can more often be put into low-power modes?" The adaptation problem involves answering the question, "How can software be modified to permit novel, power-saving uses of components?" Each of the software strategies we will consider addresses one or more of these problems.

Different components have different energy consumption and performance characteristics, so it is generally appropriate to have a separate energy management strategy for each such component. Thus in this paper we will generally consider each component separately. For each component, first we will discuss its particular hardware characteristics, then we will discuss what transition, load-change, and adaptation solutions have been proposed for that component. The components whose software power management problems are most significant are the secondary storage unit, the processing unit, the wireless communication unit, and the display unit, but we will also briefly discuss other components.

*This material is based upon work supported in part by the National Science Foundation under grants MIP-9116578 and CCR-9117028, by NASA under Grant NCC 2-550, by the State of California under the MICRO program, and by Apple Computer Corporation, Intel Corporation, Sun Microsystems, Fujitsu Microelectronics, Toshiba Corporation, and Sony Research Laboratories.

†Computer Science Division, EECS Department, University of California, Berkeley, California, 94720-1776

This paper is organized as follows. Section 2 discusses general issues in developing and evaluating solutions to the problems we have discussed. Sections 3, 4, 5, and 6 talk about specific problems and solutions involving the secondary storage unit, the processing unit, the wireless communication unit, and the display unit, respectively. Section 7 considers other, miscellaneous, components. Section 8 talks about strategies that deal with the system itself as a component to be power-managed. Finally, Section 9 concludes.

2 General Issues

2.1 Strategy types

We call a strategy for determining when to switch from one component mode to another a *transition strategy*. Transition strategies require two sorts of information about a component: knowledge about its mode characteristics and information about its future functionality requirements. By mode characteristics we mean the advantages and disadvantages of each mode the component can be in, including how much power is saved by being in it, how much functionality is sacrificed by entering it, and how long it will take to return from it.

Knowledge about mode characteristics is generally easily obtained and unchanging with time. On the other hand, information about future functionality requirements is difficult to obtain, is often inaccurate, and does change. Thus, the most difficult part of a transition strategy is continually predicting future functionality requirements. For this reason, transition strategies are sometimes called *prediction strategies*.

A common prediction tactic is to assume that the longer a component has been inactive, the longer it will continue to be inactive. Combining this prediction method with knowledge about mode characteristics will then lead to a length of time t such that whenever the component is inactive in a certain mode for longer than t, it should be placed in a lower-power mode. Such a length of time is called an *inactivity threshold*, and a strategy using one is called an inactivity threshold strategy. However, as we will see, there are other ways of predicting functionality requirements to determine when transitions should be made.

We call a strategy for modifying the load on a component in order to increase its use of low-power modes a *load-change strategy*. An example of such a strategy is disk caching, which can reduce the load on a hard disk and thereby reduce its power consumption. Note that modifying component load does not always mean reducing it; sometimes an effective way to reduce power consumption is to merely reorder service requests. For instance, the hard disk will consume less power if one makes a disk request immediately before spinning the disk down than if one makes the request immediately after spinning it down.

We call a strategy for allowing components to be used in novel, power-saving ways an *adaptation strategy*. An example of such a strategy is modifying file layout on secondary storage so that magnetic disk can be replaced with lower-power flash memory. File layout modification is important since flash memory has no seek time and requires that old data be explicitly erased before it can be overwritten.

2.2 Levels of energy management

Energy management can be done at several levels in the computer system hierarchy: the component level, the operating system level, the application level, and the user level. The end-to-end argument [63] suggests that this management should be performed at the highest level possible, because lower levels have less information about the overall workload with which to make decisions. However, certain types of strategy are inappropriate for the highest levels. Most strategies are inappropriate for the user, since the user lacks knowledge about power consumption of each component, is unable to make decisions within milliseconds or faster, and is generally unwilling to make frequent energy management decisions. The application level has the problem of many applications operating independently. No one application knows the functionality requirements of all the others, so none of them can effectively implement a global transition strategy. Also, some load-change strategies do not work if they are implemented independently by multiple entities, such as a strategy that attempts to reorder disk accesses to fit some pattern. Another problem with the application level is that, because of the abstractions presented by the operating system level, applications lack certain information about the state of the machine that would help them make energy management decisions, such as when the hard disk motor is about to turn off. For these reasons, most energy management is performed at the operating system level. The user typically just makes a few high-level decisions, such as putting the computer to sleep before he or she takes a long break or using less backlight brightness when running off of a battery. Applications typically just reduce their use of components, such as by using low-power instructions or by transmitting as little data as possible over wireless links.

One way to get the advantages of application-level management without most associated disadvantages is to use application-aware adaptation [34, 54]. In such a system, each application explicitly tells the operating system what its future needs are, such as when it plans to access the hard disk next. Also, the operating system notifies each application whenever is a change in the state of the system relevant to energy management decisions. Thus, if an energy management strategy has to be implemented at the operating system level, for instance because it cannot be implemented by multiple separate entities, it can still get information about the needs of an application from the definitive source: the application itself. Furthermore, if an energy management strategy is best implemented at the application level, it can be performed using machine state information normally confined to the operating system. For example, an application learning that the hard disk motor is about to turn off can figure out what disk data it will access in the near future and ask for it to be read into memory, thus prolonging the time the motor can thereafter be kept off. Unfortunately, it is seldom the case that appplications have the necessary knowledge or the necessary sophistication to take advantage of external notifications.

2.3 Strategy evaluation

When evaluating power management strategies, there are several points to remember. First, the effect of a strategy on the overall system power consumption is more important than its effect on the particular component it concerns. For example, a 50% reduction in modem power sounds impressive, but if the modem only accounts for 4% of total power consumption, this savings will only result in a 2% decrease in total power.

Second, it is important to use as the baseline the current strategy, rather than the worst possible strategy. For example, it would not be sufficient to simply know that a new strategy causes the disk motor to consume 85% of its maximum possible power. If the current strategy already caused it to be off 80% of the time, this would represent a small power reduction, but if the current strategy only turned it off 20% of the time, this would represent a significant power reduction.

Third, maximum battery lifetime is not necessarily what users want—they want to maximize the amount of work they can accomplish before the battery runs out, not simply the amount of time the computer can remain running before the battery runs out. For example, consider a strategy that halves the CPU speed and increases battery lifetime by 50%. If the sluggish response time makes papers take 10% longer to write, it is not reasonable to call the new strategy a 50% improvement just because the machine stays on 50% longer. The user can only write 36% more papers with one battery, so the strategy is really only a 36% improvement. Thus, to completely evaluate a new strategy, one must take into account not only how much power it saves, but also how much it extends or diminishes the time tasks take.

Component	**Hyp. 386**	**Duo 230**	**Duo 270c**	**Duo 280c**	**Avg.**
Processor	4%	17%	9%	25%	14%
Hard disk	12%	9%	4%	8%	8%
Backlight	17%	25%	26%	25%	23%
Display	4%	4%	17%	10%	9%
Modem	n/a	1%	0%	5%	2%
FPU	1%	n/a	3%	n/a	2%
Video	26%	8%	10%	6%	13%
Memory	3%	1%	1%	1%	2%
Other	33%	35%	28%	22%	30%
Total	6 W	5 W	4 W	8 W	6 W

Table 2: For various portable computers, percentage of total power used by each component when power-saving techniques are used [45, 49]

Fourth, when evaluating a strategy, it is important to consider and quantify its effect on components it does not directly manipulate. For example, a strategy that slows down the CPU may cause a task to take longer, thus causing the disk and backlight to be on longer and consume more energy.

Fifth, to be completely accurate, one also has to consider that battery capacity is not constant. Battery capacity can vary depending on the rate of power consumption [57] and on the way that rate changes with time [77]. Thus, it may be important to understand not only how much a strategy reduces power consumption, but also how it changes the function of power consumption versus time. Also, it means that computing battery lifetime is more difficult than just dividing a rated energy capacity by total power consumption. In practice, this issue is seldom considered.

In conclusion, there are four things one must determine about a component power management strategy in order to evaluate it: how much it reduces the power consumption of that component; what percentage of total system power, on average, is due to that component; how much it changes the power consumption of other components; and how it affects battery capacity through its changes in power consumption. The first, third, and fourth require simulation of the strategy; the second requires a power budget describing the average power consumption of each system component. In the next subsection, we will give some such budgets.

2.4 Power budget

Table 2 shows examples of average power consumption for the components of some portable computers when power-saving techniques are used. This ta-

Hard disk	Maxtor MobileMax 251350	Road Warrior Slimline	Toshiba MK2720	WD Portfolio
Capacity	1.35 GB	815 MB	1.35 GB	1.0 GB
Idle power	0.9 W	0.9 W	1.4 W	0.95 W
Standby power	0.23 W	0.5 W	0.35 W	0.20 W
Sleep power	0.025 W	0.15 W	0.15 W	0.095 W
Spin-up time	1 sec	5 sec	5 sec	6 sec
Spin-up energy	4.4 J	17.5 J	19.5 J	30 J

Table 3: Characteristics of various hard disks [51, 59, 70, 73]

ble shows measurements taken only when the computers were running off battery power, since power consumption is generally unimportant when the machine is plugged in. Note that power supply inefficiency is not treated as a separate category, but rather as a "tax" on all power consumed by each component. So, for instance, if the power supply system is 80% efficient, then instead of attributing 20% of power consumption to the power supply we increase the effective power consumption of each component by 25%. The first machine is a hypothetical 386DXL-based computer [49]. The next three examples describe measurements of Macintosh PowerBook Duo machines [45].

The power budget of Table 2 indicates the magnitude of possible power savings. For instance, since the hard disk consumes only 8% of total power on the Duo 280c given its current power-saving methods, better techniques for managing hard disk power could save at most 8% of total system power, increasing battery lifetime by at most 9%. With power management active, the main consumers of power include the backlight, processor, video system, and hard disk. Thus, these are the components for which further power-saving methods will be most important.

Note that these breakdowns are likely to change as time progresses [29]. For instance, wireless communication hardware will probably eventually appear in most portable computers, at first adding about 1 W to total power consumption. Hardware improvements will decrease the power consumption of various other components, but this rate of decrease will be different for different components. Evidence of changing breakdown percentages can be seen from trends in modern portable computers [19]. Later models of portable computers seem to spend a greater percentage of their power consumption on the hard disk than earlier models. Presumably, this is because later models have substantial relative savings in other components' power but not as much savings in hard disk power. These forecasts suggest that as time progresses, power-saving techniques might become more important for the display and hard disk, and less important for the processor. Note also that most existing components have been designed almost exclusively for performance and capacity, with little thought for power consumption. Components designed with minimal power use as a primary goal might behave very differently.

3 Secondary Storage

3.1 Hardware features

Secondary storage in modern computers generally consists of a magnetic disk supplemented by a small portion of main memory DRAM used as a disk cache. Such a cache improves the overall performance of secondary storage. It also reduces its power consumption by reducing the load on the hard disk, which consumes more power than the DRAM.

Most hard disks have five power modes; in order of decreasing power consumption, these are active, idle, standby, sleep, and off [29]. In active mode, the disk is seeking, reading, or writing. In idle mode, the disk is not seeking, reading, or writing, but the motor is still spinning the platter. In standby mode, the motor is not spinning and the heads are parked, but the controller electronics are active. In sleep mode, the host interface is off except for some logic to sense a reset signal. Transitions to active mode occur automatically when uncached data is requested to be read or written. Transitions to standby and sleep modes occur when explicit external directives are received. Such allowance for external control allows software to directly influence the power consumption of the hard disk, forming the basis for many software power-saving strategies.

Having the motor off, as in the standby mode, saves power. However, when it needs to be turned on again, it will take considerable time and energy to return to full speed. If this energy is greater than the savings from having the motor off, turning the motor off may actually increase energy consumption. Turning off the motor also has a performance impact, since the next disk request will be delayed until the motor returns to full speed. Going to sleep mode is an analogous operation, although one in which the savings in power, as well as the overhead required to return to the original state, are greater. Table 3 quantifies some time and energy considerations for various hard disks.

A possible technology for secondary storage is an integrated circuit called flash memory [7, 17, 37, 50, 75]. Like a hard disk, such memory is nonvolatile and can hold data without consuming energy. Furthermore, when reading or writing, it consumes only 0.15 to 0.47 W, far less than a hard disk. It has read speed of about 85 ns per byte, similar to DRAM, but write speed of about 4–10 μs, about 10–100 times slower than hard disk. However, since flash memory has no seek time, its overall write performance is not that much worse than that of magnetic disk; in fact, for sufficiently small random writes, it can actually be faster. Flash memory is technically read-only, so before a region can be overwritten it must be electrically erased. Such erasure is done one segment at a time, with each segment 0.5–128 KB in size and taking about 15 μs per byte to erase [75]. A segment can only be erased 100,000 to 1,000,000 times in its lifetime before its performance degrades significantly. The current cost per megabyte of flash is $2–4, making it about 17–40 times more expensive than hard disk but about 2–5 times less expensive than DRAM. Cáceres et al. [7] point out that the costs of flash and hard disk may become comparable in the future since the per-year increases in megabytes per dollar are about 25% for magnetic disk and 40% for flash; given these assumptions, the two prices could converge in 6–8 years. Flash memory offers great opportunities for secondary storage power savings if it can be substituted for the hard disk or used for caching. Before that, however, software must be designed to overcome the many limitations of flash memory, especially its poor write performance.

3.2 Transition strategies

Most transition strategies for magnetic disks are for predicting when it is appropriate to go to sleep mode. Most of those leave it to the disk controller to automatically return to idle mode when appropriate. Transition strategies also exist to decide when to turn off the entire hard disk, but these are all simple strategies based on inactivity timers, and have not been the subject of experimental scrutiny.

We know of no studies of transition strategies for placing disks in standby mode. One reason is that this is a relatively new feature. Another reason is that it is almost always better to enter sleep mode than standby mode. Sleep mode consumes less power, and since the time it takes to go from sleep to idle mode is dominated by the spin-up time of the motor, this transition takes no longer than that from standby to idle mode. The main advantage to standby mode is that on-disk cache contents are preserved, but since on-disk caches tend to be smaller than operating system disk caches this is not especially important.

3.2.1 Fixed inactivity threshold

The most common transition strategy for going into sleep mode is to enter that mode after a fixed inactivity threshold. When hard disks allowing external control over the motor were first developed, their manufacturers suggested this strategy be used with an inactivity threshold of 3–5 minutes. However, researchers eventually discovered that power consumption could be minimized by using inactivity thresholds as low as 1–10 seconds; such low thresholds save roughly twice as much power as a 3–5 minute interval [19, 42].

The greater power savings from using a smaller inactivity threshold comes at a cost, however: perceived increased user delay. Spinning down the disk more often means that the user will more often have to wait for an additional spin-up time before a disk request can be completed. The inactivity threshold yielding minimum disk power results in user delay of about 8–30 seconds per hour; some researchers believe this to be an unacceptable amount of delay [19]. Thus, the best disk spin-down policy is not necessarily the one that minimizes power consumption, but the one that minimizes power consumption while keeping user delay at a tolerable level. Note that no one has yet separated actual user delay from perceived user delay. A loss of 8–30 seconds per hour should be trivial, and were the disk spin-up entirely soundless, it is quite possible that the user would be able to tolerate far longer delays.

It is worth pointing out, although it should be obvious, that the time between disk accesses is not exponentially distributed; the expected time to the next disk access is generally an increasing function of the time since the last access. If the interaccess times for disk reference were not skewed in this manner, the correct strategy would use an inactivity threshold of either zero or infinity [27].

3.2.2 Changing inactivity threshold

There are several arguments for dynamically changing the inactivity threshold, not necessarily consistent with each other. The first argument is that disk request interarrival times are drawn independently from some unknown stationary distribution. Thus, as time passes one can build up a better idea of this distribution, and from that deduce a good threshold. The second argument is that the interarrival time distribution is nonstationary, i.e. changing with time, so a strategy should always be adapting its threshold to the currently prevailing observed distribution. The third argument is that worst-case performance can be bounded by choosing thresholds randomly. In other words, any deterministic threshold can fall prey to a particularly nasty series of disk access patterns, but changing the threshold randomly eliminates this danger.

If disk interarrival times are independently drawn from some unknown stationary distribution, as the first argument states, then no matter what this distribution, there exists an inactivity threshold that incurs a cost no more than $e/(e-1)$ times that of the optimal off-line transition strategy [36]. One could find this threshold by keeping track of all interarrival times so that the distribution, and thus the ideal threshold, could be deduced.

One algorithm of that type, using constant space, builds up a picture of the past interarrival time distribution in the following indirect way [39]. It maintains a set of possible thresholds, each with a value indicating how effective it would have been. At any point, the algorithm chooses as its threshold the one that would have performed the best. Incidentally, "best" does not simply mean having the least power consumption; the valuation might take into account the relative importance of power consumption and frequency of disk spin-downs specified by the user. This algorithm has been shown to perform well on real traces, beating many other practical algorithms.

Another strategy using a list of candidate thresholds is based on the second argument, that disk access patterns change with time [30]. In this strategy, each candidate is initially assigned equal weight. After each disk access, candidates' weights are increased or decreased according to how well they would have performed relative to the optimal off-line strategy over the last interaccess period. At any point, the threshold chosen for actual use is the weighted average of all the candidates. Simulations show that this strategy works well on actual disk traces. The developers of this strategy only considered using it to minimize power consumption; however, it could easily be adapted to take frequency of spin-ups into account.

Another dynamic strategy based on the second argument tries to keep the frequency of annoying spin-ups relatively constant even though the interaccess time distribution is always changing [18]. This strategy raises the threshold when it is causing too many spin-ups and lowers it when more spin-ups can be tolerated. Several variants of this strategy, which raise and lower the threshold in different ways, are possible. Simulation of these variants suggests that using an adaptive threshold instead of a fixed threshold can significantly decrease the number of annoying spin-ups experienced by a user while increasing energy consumption by only a small amount.

By the third argument, a strategy should make no assumptions about what the disk access pattern looks like, so that it can do well no matter when disk accesses occur. One such strategy chooses a new random threshold after every disk access according to the cumulative distribution function

$$\pi(t) = \frac{e^{t/c} - 1}{e - 1},$$

where c is the number of seconds it takes the running motor to consume the same amount of energy it takes to spin up the disk [36]. This strategy has been proven ideal among strategies having no knowledge of the arrival process. Note, however, that almost all transition strategies described in this paper do purport to know something about the arrival process, and thus are capable of beating this strategy. In other words, although this strategy does have the best worst-case expected performance, it does not necessarily have the best typical-case performance.

3.2.3 Alternatives to an inactivity threshold

Some transition strategies have been developed that do not use an inactivity threshold explicitly [19]. One such strategy is to predict the actual time of the next disk access to determine when to spin down the disk. However, simulations of variants of this strategy show that they provide less savings than the best inactivity threshold strategy, except when disk caching is turned off. This may be because filtering a pattern of disk accesses through a disk cache makes it too patternless to predict. Another strategy is to predict the time of the next disk request so the disk can be spun up in time to satisfy that request. However, no techniques proposed for this have worked well in simulation, apparently because the penalty for wrong prediction by such strategies is high. Despite the shortcomings of the non-threshold-based transition strategies studied so far, some researchers remain hopeful about the feasibility of such strategies. Simulation of the optimal off-line strategy indicates that such strategies could save as much as 7–30% more energy than the best inactivity threshold method.

3.3 Load-change strategies

Another way to reduce the energy consumption of a hard disk is to modify its workload. Such modification is usually effected by changing the configuration or usage of the cache above it.

One study found that increasing cache size yields a large reduction in energy consumption when the cache is small, but much lower energy savings when the cache is large [42]. In that study, a 1 MB cache reduced energy consumption by 50% compared to no cache, but further increases in cache size had a small impact on energy consumption, presumably because cache hit ratio increases slowly with increased cache size [76]. The study found a similar effect from changing the dirty block timeout period, the maximum time that cache contents are permitted to be inconsistent with disk contents. Increasing this timeout from zero to 30 seconds reduced disk energy consumption by about 50%, but further increases in the timeout delay had only small effects on energy consumption [42]. Another possible cache modification is to add file name and attribute caching. Simulation showed a moderate disk energy reduction of 17% resulting from an additional 50 KB of cache devoted to this purpose.

Prefetching, a strategy commonly used for performance improvement, should also be effective as an energy-saving load-change strategy. However, we know of no studies examining this latter benefit of prefetching. If the disk cache is filled with data that will likely be needed in the future before it is spun down, then more time should pass before it must again be spun up. This idea is similar to that of the Coda file system [64], in which a mobile computer caches files from a file system while it is connected so that when disconnected it can operate independently of the file system.

3.4 Adaptation strategies for flash memory as disk cache

Flash memory has two advantages and one disadvantage over DRAM as a disk cache. The advantages are nonvolatility and lower power consumption; the disadvantage is poorer write performance. Thus, flash memory might be effective as a second-level cache below the standard DRAM disk cache [50]. At that level, most writes would be flushes from the first-level cache, and thus asynchronous. However, using memory with such different characteristics necessitates novel cache management strategies.

The main problem with using flash memory as a second-level cache is that data cannot be overwritten without erasing the entire segment containing it. One solution is to ensure there is always a segment with free space for writing; this is accomplished by periodically choosing a segment, flushing all its dirty blocks to disk, and erasing it [50]. One segment choosing strategy is to choose the one least recently written; another is to choose the one least recently accessed. The former is simpler to implement and ensures no segment is cleaned more often than another, but the latter is better at preserving locality. Unfortunately, even the latter is not very good at preserving locality, since different blocks in the same segment may have very different access frequencies; one solution is to supplement the strategy with a copying garbage collector, such as that found in LFS [60], to choose recently used blocks of a segment about to be erased and write them into a segment that also contains recently used blocks and is not going to be erased.

Simulations have shown that using a second-level flash memory cache of size 1–40 MB can decrease secondary storage energy consumption by 20–40% and improve I/O response time by 30–70% [50]. Thus, using appropriate cache management policies seem to allow a flash memory second-level cache to reduce energy consumption and still provide equal or better performance than a system using a traditional cache. The simulations would have been more persuasive, however, if they had compared the system with flash to one with a DRAM second-level cache rather than to one with no second-level cache.

3.5 Adaptation strategies for flash memory as disk

Flash memory is a low-power alternative to magnetic disk. However, the large differences between flash memory and magnetic disk suggest several changes to file system management. Since flash has no seek latency, there is no need to cluster related data on flash memory for the purpose of minimizing seek time [7]. Since flash is practically as fast as DRAM at reads, a disk cache is no longer important except to be used as a write buffer [7]. Such a write buffer would make writes to flash asynchronous, thus solving another problem of flash memory: poor write performance. In fact, if SRAM were used for this write buffer, its permanence would allow some writes to flash to be indefinitely delayed [17, 75]. Finally, unlike magnetic disk, flash memory requires explicit erasure before a segment can be overwritten, a slow operation that can wear out the medium and must operate on a segment at a time. One solution to these problems is to use a log-structured file system like LFS [37, 60], in which new data does not overwrite old data but is instead appended to a log. This allows erasures to be decoupled from writes and done asynchronously, thus minimizing their impact on performance. A flash file system also needs some way to ensure that no physical segment is cleaned especially often. One way to do this is to make sure that physical segments containing infrequently modified data and ones containing frequently modified data switch roles occasionally [75].

Simulations of flash file systems using some of these ideas have found that they can reduce secondary storage power consumption by 60–90% while maintaining aggregate performance comparable to that of magnetic disk file systems [17]. However, at high levels of utilization, the performance of file systems using asynchronous erasure can degrade significantly due to the overhead of that erasure.

3.6 Adaptation strategies for wireless network as disk

Another hardware approach to saving secondary storage energy is to use wireless connection to a plugged-in file server instead. Offloading storage has the advantage that the storage medium can be big and power-hungry without increasing the weight or power consumption of the portable machine. Disadvantages include increased power consumption by the wireless communication system, increased use of network bandwidth, and higher latency for file system accesses. Adaptation strategies can help minimize the impact of the disadvantages but retain the advantages.

The general model for using wireless communication as secondary storage is to have a portable computer transmit data access requests to, and receive data from, a server. An improvement on this is to have the server make periodic broadcasts of especially popular data, so the portable computer will have to waste less power transmitting requests [33]. A further improvement is to interleave index information in these broadcasts, so a portable computer can anticipate periods of no needed data and shut its receiver off during them.

Another model for using wireless communication for storage is proposed in extensions to the Coda scheme [64]. In this model, the portable computer storage system functions merely as a large cache for the server file system. When the portable computer is not wired to the file system server, it services cache misses using wireless communication. Because such communication is slow and bandwidth-consuming, the cache manager seeks to minimize its frequency by hoarding files that are anticipated to be needed during disconnection.

A third model for using wireless communication for storage, used by InfoPad [4, 5], is to perform *all* processing on an unmoving server. In this model, the portable "computer" is merely a terminal that transmits and receives low-level I/O information, so the energy consumption for general processing and storage is consumed by plugged-in servers instead of the mobile device. In this way, portable storage and CPU energy consumption are traded for high processing request latency, significant network bandwidth consumption, and additional energy consumption by the portable wireless device.

The limiting factor in all of these cases is network bandwidth; what is practical depends on the bandwidth between the local system and the data source. A packet radio connection at 28.8 Kb/s is very different than the type of multi-megabit per second system that could be implemented within a building.

3.7 Summary

Because of the large percentage of portable energy consumption by the secondary storage system, a great deal of research has concerned reducing that energy consumption with software strategies. Generally, such transition strategies attempt to predict when the disk will be idle long enough that the disk can be placed in sleep mode. Load-change strategies generally tweak the caching policy, such as by increasing cache size or dirty block timeout period, to reduce disk energy consumption.

Using different components can also reduce secondary storage energy consumption, opening up new software issues in how to deal with their different characteristics. For example, flash memory can be used as a disk cache to save 20–40% of secondary storage power consumption. Also, flash memory can be substituted for the hard disk itself to reduce that power consumption by 60–90%, but doing so requires using a file system that hides the poor write performance of flash. Yet another option is to use wireless communication with an immobile file server; schemes for making this feasible include periodic broadcasting of frequently accessed data, using the portable hard disk as a cache for the main file system, and offloading all processing and storage needs to a remote server.

A great deal of research remains to be done on software techniques to reduce storage system power consumption. First, all of the study results for power-saving techniques are based on reasonably short and restricted traces of disk activity; more research is needed to ensure that the results are applicable to a wide range of workloads. Second, these traces were generally taken from machines that were plugged in, despite the possibility that disk access patterns are different when machines are running on battery power; more studies are needed on how the strategies work when battery power is used. Third, since different strategies generally work best for different workloads, more work is needed on dynamic techniques that adjust their mode of operation depending on the prevailing workload characteristics. Fourth, more work is needed to find additional useful predictors of time to next access. Fifth, research is needed to decide if Coda-style hoarding techniques that treat disk spin-downs like disconnections can reduce the frequency of disk spin-ups. Finally, we need more solutions to the file system design challenges that the use of flash memory and wireless networks in place of disk present.

Hardware designers can also do much to aid software techniques in reducing power. In order to minimize the impact of decisions to spin down the hard disk, the energy and time consumed by a disk when spinning up should be reduced. To make it easier for software to achieve good performance with flash memory, its design should emphasize fast writing and erasing, as well as the ability to erase at the same time that data is being read or written. Increasing the number of erasures possible in the lifetime of a segment would simplify software handling of flash memory. Finally, changes in disk design to minimize power, e.g. by spinning the disk more slowly, might have a significant effect.

4 Processor

4.1 Hardware features

Processors designed for low-power computers have many power-saving features. One power-saving feature is the ability to slow down the clock. Another is the ability to selectively shut off functional units, such as the floating-point unit; this ability is generally not externally controllable. Such a unit is usually turned off by stopping the clock propagated to it. Finally, there is the ability to shut down processor operation altogether so that it consumes little or no energy. When this last ability is used, the processor typically returns to full power when the next interrupt occurs.

In general, slowing down the processor clock without changing the voltage is not useful. Power consumption P is essentially proportional to the clock frequency f, the switching capacitance C, and the square of the voltage V ($P \propto CV^2 f$), but the time t it takes the processor to complete a task is inversely proportional to the clock frequency ($t \propto 1/f$). Since the energy E it takes the processor to complete a task is the product of its power consumption and the time it spends ($E = Pt$), this energy consumption is invariant with clock speed. Thus, reducing the clock speed lengthens processor response time without reducing the amount of energy the processor consumes during that time. In fact, slowing the clock speed can easily increase total energy consumption by extending the time other components need to remain powered. However, if the voltage can be decreased whenever the clock speed is reduced, then energy consumption, which is proportional to the square of the voltage ($E \propto CV^2$), would actually be reduced by slowing the clock.

Turning off a processor has little downside; no excess energy is expended turning the processor back on, the time until it comes back on is barely noticeable, and the state of the processor is unchanged from it turning off and on, unless it has a volatile cache [26]. On the other hand, there is a clear disadvantage to reducing the clock rate: tasks take longer. There may also be a slight delay while the processor changes clock speed.

Reducing the power consumption of the processor saves more than just the energy of the processor itself. When the processor is doing less work, or doing work less quickly, there is less activity for other components of the computer, such as memory and the bus. For example, when the processor on the Macintosh Duo 270c is off, not only is the 1.15 W of the processor saved, but also an additional 1.23 W from other components [45]. Thus, reducing the power consumption of the processor can have a greater effect on overall power savings than it might seem from merely examining the percentage of total power attributable to the processor.

4.2 Transition strategies for turning the processor off

When the side effects of turning the processor off and on are insignificant, the optimal off-line transition strategy is to turn it off whenever the processor will not be needed until the next interrupt occurs. With a well-designed operating system, this can be deduced from the current status of all processes. Thus, whenever any process is running or ready to run, the processor should not be turned off; when all processes are blocked, the processor should be turned off [46, 67, 68]. Examples of operating systems using this strategy are Windows [13, 56] and UNIX.

MacOS, however, uses a different strategy, since it does not make a clear distinction between blocked and unblocked processes. It uses an inactivity threshold, as is commonly used for hard disk power management. The processor is shut off when there have been no disk accesses in the last fifteen seconds and no sound chip accesses, changes to the cursor, displaying of the watch cursor, events posted, key presses, or mouse movements in the last two seconds. The savings achievable from this strategy vary greatly with workload [44, 45].

4.3 Load-change strategies when the CPU can turn off

Given a transition strategy that turns off the processor when it is not performing any tasks, the goal of a load-change strategy is simply to limit the energy a processor must consume to perform those tasks. There are three approaches such a load-change strategy can take: reducing the time those tasks take, using lower-power instructions to perform those tasks, and reducing the number of unnecessary tasks the processor is called upon to perform. Below we present several load-change strategies that use different subsets of these approaches.

One load-change technique, which uses the first two approaches, is to use more efficient operating system code [65]. Many commercial operating systems use excessively large amounts of code and excessively inefficient algorithms, but some, like Plan9 from Bell Labs, purportedly have dense code and efficient algorithms. Efficient algorithms lead to shorter task running times. Denser code leads to less memory system activity, which in turn leads to lower energy consumption per operation. Note that some of this energy savings comes not from the processor but from the memory system; this is an example of a technique to reduce the power consumption of one component having a favorable effect on the power consumption of another component. Realistically, if operating systems programmers cannot be persuaded to improve operating system performance for performance reasons alone, the advantages of reduced energy consumption are likely to have little additional weight.

Another load-change technique, which uses the same approaches, is to use energy-aware compilers, i.e. compilers that concern themselves with the energy efficiency of the code they generate [69]. Traditional compiler techniques, such as compiling to reduce pipeline stalls and using code transformations to improve cache hit rates, have application as load-change strategies in that they reduce the amount of time a processor takes to complete a task. Furthermore, giving the compiler information about the relative energy consumption of different instructions may allow the compiler to optimize total energy consumption instead of trying merely to optimize performance. There is little if any reason, however, to believe that such savings would be significant.

Another load-change technique uses the third approach, performing fewer unnecessary tasks [46]. Often, when an application is idle, it will refuse to perform any explicitly blocking action and instead "busy-wait" for a significant event to occur. When this is happening, the standard transition strategy will not turn off the processor even though it is not doing any useful work. One way to solve this problem is to force an application to block for a certain period whenever it satisfies certain conditions that indicate it is likely to be busy-waiting and not performing any useful activity. Simulations of such a strategy using traces of machines running on battery power showed that it would allow the processor to be off, on average, 66% of the time, compared to 47% when no measures were taken to forcibly block applications.

4.4 Transition strategies for dynamically changing CPU speed

As explained before, slowing the clock is useless if voltage is kept constant. Therefore, when we discuss strategies to take advantage of slowing the processor clock, we are assuming that slowing the clock is accompanied by reducing the voltage. The voltage can be reduced when the clock speed is reduced because under those conditions the longer gate settling times resulting from lower voltage become acceptable.

Previous calculations have shown that the lowest energy consumption comes at the lowest possible speed. However, performance is also reduced by reducing the clock speed, so any strategy to slow the clock must achieve all its energy savings at the expense of performance. Furthermore, reducing processor performance may cause an increase in the energy consumption of other components. Thus, it is important to make an appropriate trade-off between energy reduction and performance.

The ideal strategy is to determine the deadlines for all processing tasks, and at all times to run the processor just fast enough to meet those deadlines. Except in real-time systems, however, deadlines are not firm, and often can be estimated only with difficulty. Likewise, it is often difficult to estimate how much work a pending task entails. Some scheduling strategies have been presented that attempt to solve these problems [10, 71].

All of these strategies were designed to achieve two general goals. The first goal is to not delay the completion time of any task by more than several milliseconds. This ensures that interactive response times do not lengthen noticeably, and ensures that other components, which are usually power-managed by inactivity thresholds on the order of seconds or minutes, do not get turned off noticeably later. The second goal is to adjust CPU speed gradually. This is desirable because voltage scaling causes the minimum voltage V permissible at a clock speed f to be roughly proportional to f. Thus, the number of clock cycles executed in an interval I is proportional to $\int_I f(t)\ dt$, while the energy consumed during that interval is proportional to $\int_I f^3(t)\ dt$. Given these equations, it can be mathematically demonstrated that the most energy-efficient way to execute a certain number of cycles within a certain interval is to keep clock speed constant throughout the interval.

One strategy for adjusting CPU speed seeks to achieve these goals in the following way [71]. Time is divided into 10–50 ms intervals. At the beginning of each interval, processor utilization during the previous interval is determined. If utilization was high, CPU speed is slightly raised; if it was low, CPU speed is slightly lowered. If, however, the processor is falling significantly behind in its work, CPU speed is simply raised to the maximum allowable. Simulations show that if voltage can be varied between 3.3 V and 5 V, then application of this strategy reduces energy consumption by about 50% relative to what it is at a constant 5 V. Furthermore, if voltage can be varied between 2.2 V and 5 V, the strategy reduces energy consumption by 70% relative to 5 V. Results are not this good when voltage can be varied between 1 V and 5 V, seemingly because the availability of such a low voltage causes the strategy to generate extreme variation in the voltage level over time. And, as we have seen, it is important to keep the voltage as constant as possible.

Another strategy is similar in that it divides time into 10–50 ms intervals [10]. At the beginning of each interval, it predicts the number of CPU busy cycles that will occur during that interval, and sets the CPU speed just high enough to accomplish all this work. There are actually many variants of this strategy, each using a different prediction technique. In simulations, the most successful variants were one that always predicted the same amount of work would be introduced each interval, one that assumed the graph of CPU work introduced versus interval number would be volatile with narrow peaks, and one that made its prediction based on a weighted average of long-term and short-term CPU utilization but ignored any left-over work from the previous interval. The success of these variants suggests that it is important for such a strategy to balance performance and energy considerations by taking into account both short-term and long-term processor utilization in its predictions. Too much consideration for short-term utilization increases speed variance and thus energy consumption. Too much consideration for long-term utilization makes the processor fall behind during especially busy periods and thus decreases performance.

There is still much research to be done on interval-based transition strategies such as those described above. First, more strategies should be developed and evaluated. For example, per-process prediction might be more effective than aggregate prediction, and application hints could further aid prediction [10]. Second, the need to pre-set various parameters, such as the interval length for each strategy, should be eliminated. A strategy should stand on its own, performing any parameter adjustment internally based on the observed workload and environment. Third, strategies should be evaluated on a much broader set of workloads. Fourth, better simulation methods are needed that take into account process rescheduling resulting from changes in clock speed.

Furthermore, there are other, non-interval-based approaches to the design of clock speed transition strategies. Using intervals to ensure that no work gets delayed more than a fixed period of time is only one way to limit the performance impact. A better way is to determine appropriate deadlines for all tasks and attempt to delay no task past its deadline [10, 71]. In this way, lower speeds can be used when there are no urgent deadlines. To achieve this, a strategy needs to predict when tasks will begin, how many cycles they will need to complete, and what their deadlines will be. Furthermore, as we have learned from interval-based strategies, these predictions should be both short-term and long-term, so that the clock is adjusted precisely enough to complete tasks reasonably close to their deadlines but not so precisely that the clock speed changes unnecessarily. Research on this type of strategy must address the fact that deadlines and completion times are extremely difficult to predict. Furthermore, it must take into account the fact that if intervals are not used, a task could be extended long enough to significantly impact the energy consumption of some other component whose functionality the task requires.

4.5 Load-change strategies when CPU speed can change dynamically

Currently, no one has investigated load-change strategies to take advantage of slowing down the processor clock; such strategies would modify the load on the processor so that the clock could run at a lower or more constant speed. However, priority-based task scheduling, a strategy that many operating systems already use for effective multitasking, would also serve as such a load-change strategy. Priority-based scheduling assigns high priority to tasks with early completion deadlines, such as interactive tasks, and schedules tasks preferentially based on priority. This is good for energy consumption in the case that CPU speed can change dynamically, for the following reason. If time is squandered on tasks with far-off deadlines instead of being spent on tasks with earlier deadlines, then as the deadlines for these latter tasks approach, the CPU will have to run especially fast so they can meet their deadlines. And, as we have seen before, uneven CPU speed leads to greater CPU energy consumption.

4.6 Load-change strategies when functional units can turn off

Typically, if functional units can be turned off, the chip internals turn them off automatically when unused. This makes software transition strategies unnecessary and impossible to implement, but makes load-change strategies tenable. One such load-change strategy is to use a compiler that clusters together several uses of a pipelined functional unit so that the unit is on for less time. Another is, when compiling, to preferentially generate instructions using functional units that do not get power-managed. However, no research has been done yet on such load-change strategies. It is unclear whether the power savings to be obtained from these strategies would be significant.

4.7 Summary

Reducing the power consumption of the processor can have a large effect on overall power consumption, so many software strategies for exploiting its power-saving hardware features have been developed. These hardware features include the ability to turn the processor off altogether, to turn off certain functional units, and to slow the clock while reducing the voltage.

The most common transition strategy for turning off the processor is to turn it off when all processes are blocked, although MacOS uses an inactivity threshold instead. There are also load-change strategies that are worthwhile when the processor can be turned off. Using energy-aware compilers as well as compact and efficient operating systems can reduce the number of instructions executed and may cause low-energy instructions to be executed instead of high-energy instructions. A heuristic for forcibly blocking busy-waiting processes can cause fewer unnecessary instructions to be executed. More research is needed on such load-change strategies, to verify and extend the evaluations performed so far and to suggest more and better strategies.

When clock speed can be dynamically adjusted along with voltage, a transition strategy is needed to run the processor quickly enough to meet processing deadlines but slowly enough to save energy. Interval-based strategies divide time into fixed-size intervals, and attempt to complete tasks by the end of the same interval in which they would complete if the processor ran at full speed. More work is necessary to further evaluate existing interval-based transition strategies and to develop new ones. Researchers should also look for non-interval-based transition strategies that consider differences in task deadlines.

No load-change strategies have been suggested for slowing the processor clock, but the main component of such strategies likely will be task scheduling. Also, no software strategies to take advantage of the ability to shut off functional units have been developed. Such strategies might reduce the energy consumption of such power-managed functional units by clustering their uses or by preferentially using other functional units.

There are several things that hardware designers can do to increase the usefulness of software strategies for processor energy reduction. Perhaps the most important is designing the motherboard so that reduction in the energy consumption of the processor yields a consequent large reduction in the energy consumption of other components. In other words, motherboard components should have states with low power consumption and negligible transition side effects that are automatically entered when the processor is not presenting them with work. Voltage scaling is not widely available, and needs to be made so. Once this happens, the software strategies that anticipate this technology can be put to good use. Finally, if hardware designers can keep the time and energy required to make transitions between clock speeds low, the savings from clock speed transition strategies will be even greater.

5 Wireless communication devices

5.1 Hardware features

Wireless communication devices are appearing with increasing frequency on portable computers. These devices can be used for participating in a local or wide area wireless network, or for interacting with disconnected peripherals like a printer or mouse. They typically have five operating modes; in order of decreasing power consumption, these are transmit, receive, idle, sleep, and off [29]. In transmit mode, the device is transmitting data; in receive mode, the device is receiving data; in idle mode, it is doing neither, but the transceiver is still powered and ready to receive or transmit; in sleep mode, the transceiver circuitry is powered down, except sometimes for a small amount of circuitry listening for incoming transmissions. Transitions between idle and sleep mode typically take some time; for instance, the HSDL-1001 infrared transceiver takes about 10 μs to enter sleep mode and about 40 μs to wake from it. Furthermore, some devices provide the ability to dynamically modify their transmission power, to trade off signal strength for energy savings.

Wireless device power consumption depends strongly on the distance to the receiver. For instance, the wide-area ARDIS system, in which each base station covers a large area, requires transmit power of about 40 W, but the wide-area Metricom system, which uses many base stations each serving a small area, requires mobile unit transmit power of only about 1 W [15]. Local-area networks also tend to provide short transmission distances, allowing low power dissipation. For instance, the WaveLAN PCMCIA card, meant for use in such networks, consumes only about 1.875 W in transmit mode [47]. Even smaller distances, such as within an office or home, yield even smaller power requirements. For instance, the HSDL-1001 infrared transceiver consumes only 55 mW in transmit mode [31], and the CT-2 specification used for cordless telephones requires less than 10 mW for transmission [15].

5.2 Transition strategies for entering sleep mode

Transition strategy issues for wireless communication devices entering sleep mode are quite similar to those for hard disks, so the solutions presented for hard disks are generally applicable to them. However, some features of wireless communication suggest two changes to the standard inactivity threshold methods used with hard disks. First, because a wireless communication device does not have the large mechanical component a hard disk has, the time and energy required to put it in and out of sleep mode are much smaller. Further, the user is unlikely to know when the unit is off or on unless some sort of monitor is installed, so users should be unaware of start-up times unless they are unduly long. These factors suggest a more aggressive energy management strategy such as using a much shorter inactivity threshold. Second, it may be necessary to have wireless devices periodically exit sleep mode for a short period of time to make contact with a server, so that the server does not decide the unit is off or out of range and delete state information related to the connection [48].

Estimates of the amount of time wireless devices in local-area networks can be kept in sleep mode are relatively high. For instance, even though the WaveLAN PCMCIA card consumes 1.875 W in transmit mode, 1.625 W in receive mode, and 0.18 W in sleep mode, the manufacturer estimates its average power consumption to be about 0.325 W [47].

5.3 Load-change strategies when sleep mode is used

One way to increase the amount of time a wireless device can spend sleeping is simply to reduce network usage altogether. There are many strategies for doing this, some examples of which are as follows. One strategy is to compress TCP/IP headers; this can reduce their size by an order of magnitude, thus reducing the wireless communication activity of a mobile client [16]. Another strategy is to reduce the packet transmission rate while the channel quality appears to be degraded, so that less transmission time is wasted sending packets that will be lost [77]. Yet another strategy is to have servers [54] or proxies [25] use information about client machine characteristics and data semantics to provide mobile clients with versions of that data with reduced fidelity and smaller size; this reduces the amount of energy mobile clients must expend to receive the data. For example, a data server might convert a color picture to a black-and-white version before sending it to a mobile client. A fourth strategy is to design applications that avoid unnecessary communication, especially in the expensive transmit direction. Note that these strategies also function as load-change strategies for increasing the use of the idle mode instead of the higher-power modes, and that the last strategy can function as a load-change strategy for increasing the use of receive mode instead of transmit mode.

Another way to increase the amount of time a wireless device can sleep is to cluster its accesses. This is because it is better to perform many accesses at once, and then sleep for a long time, than to perform accesses every once in a while, going back and forth into sleep mode. One strategy for this is to buffer requests until the wireless device has to be turned on anyway to maintain contact with the server. This strategy is probably best implemented by applications, since they best know how long it is reasonable to delay accesses; however, the applications would need information about when the wireless device is activated for server contact. Another strategy, which does not require applications to know when this server contact is made, is simply for applications to buffer their wireless accesses until a reasonable number of them can be made at once. For instance, a mail program can buffer outgoing mail until a certain amount of mail has built up or until the next time it sends a request for incoming mail.

5.4 Transition strategies for changing transmission power

When transmission power can be reduced dynamically, a transition strategy to decide when to change power is needed. Usually, such a strategy takes two things into account in deciding what power level to use: the quality of service currently required, i.e. the rate of successful transmission currently needed, and the interference level. The greater the quality of service needed, the more power is required. The higher the interference, the more power is needed to overcome that interference; however, above a certain level of interference it becomes pointless to use any transmission power at all since transmissions are unlikely to succeed. Note that reducing transmission power does not always reduce energy consumption, since if transmission power is so low that errors are extremely frequent, the large number of retransmissions and/or the increased number of error correction bits necessary might cause total energy consumption to increase.

One transition strategy is to choose a power level and interference threshold based on the current quality of service requirements, then to run the transmitter at either that power level or zero, depending on whether the interference level is below or above that threshold. Given a model of interference that provides the probability distribution of interference level and the probability of successful transmission as a function of interference level, numerical methods can be used to find the power level and interference threshold that achieve the quality of service desired while minimizing energy consumption [62].

Another transition strategy is to choose a ratio of signal power to interference level and an interference threshold based on the current quality of service requirements, then to run the transmitter at sufficient power to provide that ratio whenever interference is below that threshold. As before, a model of interference can provide values for the ratio and threshold that minimize energy consumption while maintaining quality of service [61].

The most complex, yet most effective, strategy of this sort is to allow power level to vary as an arbitrary function of interference level and quality of service. If the model of interference is straightforward, numerical techniques can still be used to determine the function to use to minimize energy consumption and maintain quality of service. Analytical results suggest that this kind of strategy can save as much as 25% more transmission energy than the previous one described [61].

A transition strategy that changes transmission power is unique among the strategies we have considered in this paper in that it can have an effect on the power consumption of other machines. This is because transmissions of one machine are interference to others. Thus, it may be appropriate to consider a global, distributed strategy that minimizes energy consumption across all portable machines in range of each other. Some evidence suggests that if each machine attempts to minimize its own energy consumption, good global behavior results. One reason for this is that machines attempting to reduce their energy use will produce as little interference as possible. Another reason is that if machines tend to reduce their power levels when interference makes transmission unworthwhile, then machines will tend somewhat to take turns in transmitting and thus waste fewer resources competing for the same time slots [61].

5.5 Load-change strategies when transmission power can change

When transmission power can be reduced dynamically, load-change strategies can be used to minimize the transmission power needed. One such strategy is to explicitly arbitrate for time division of the transmission medium, so that interference levels are low when transmission is done. Another strategy is to use error-correcting codes in transmitted packets. This reduces the frequency of retransmission when errors occur, thus allowing lower transmission power to be used.

5.6 Summary

Wireless devices are increasingly being used in portable computers, so software strategies to take advantage of their hardware power-saving features are becoming increasingly important. Transition strategies for using sleep mode are like those for hard disks, except that they generally use shorter inactivity thresholds. Load-change strategies increase the use of sleep mode by reducing bandwidth usage or clustering wireless accesses. Such load-change strategies can also increase the use of idle mode instead of active modes and the use of receive mode instead of transmit mode. When transmission power can be reduced dynamically, transition strategies can be used to vary power in a way that minimizes energy consumption subject to quality of service constraints. Load-change strategies can then be used to reduce the amount of transmission power needed over time.

6 Display and Backlight

6.1 Hardware features

The display and backlight have few energy-saving features. This is unfortunate, since they consume a great deal of power in their maximum-power states; for instance, on the Duo 280c, the display consumes a maximum of 0.75 W and the backlight consumes a maximum of 3.40 W [45]. The backlight can have its power reduced by reducing the brightness level or by turning it off. The display power consumption can be reduced by switching from color to monochrome, by reducing the update frequency, or by turning it off. Reducing the update frequency reduces the range of colors or shades of gray for each pixel, since such shading is done by electrically selecting each pixel for a particular fraction of its duty cycle. Generally, the only disadvantage of these low-power modes is reduced readability. However, in the case of switches among update frequencies and switches between color and monochrome, the transitions can also cause annoying flashes.

6.2 Transition strategies

Essentially the only transition strategy currently used to take advantage of these low-power features is to turn off or down the backlight and display after a certain period of time has passed with no user input. The reasoning behind this strategy is that if the user has not performed any input recently, then it is likely he or she is no longer looking at the screen, and thus the reduced readability of a low-power mode is acceptable for the immediate future. To lessen the effect of a wrong guess about such inactivity on the part of the user, some machines do not shut the backlight off immediately but rather make it progressively dimmer. In this way, if the user is actually still looking at the screen, he or she gets a chance to indicate his or her presence before the entire screen becomes unreadable. One study found that thanks to the use of low-power states, the backlights on some machines consumed only 32–67% of maximum possible energy while running on battery power [45].

A possible modification of this standard strategy is to automatically readjust the inactivity threshold to make it a better predictor of user inactivity. For example, if the user hits a key just as the backlight begins to dim, such a strategy might increase the inactivity threshold on the assumption that its current value is too short.

Another possible transition strategy is to switch the display to monochrome when color is not being displayed, or to switch it to a lower update frequency when the items displayed do not require a high update frequency. The operating system might even switch to a lower-power display mode when those parts of the screen making use of the current display mode are not visually important to the user. For example, if the only color used on the screen were in a window belonging to an application not recently used, the operating system might switch the display to monochrome. The use of such features would be most acceptable if such transitions could be made unobtrusive, e.g. without a flash, and perhaps even with progressive fading.

Since less backlight brightness is needed when the ambient light is brighter, another possible transition strategy is to dim the backlight when ambient light increases [66]. Such a strategy would require a sensor to determine the ambient light level.

6.3 Load-change strategies

Currently, there are no formal load-change strategies for reducing the energy consumption of the display or backlight. However, it has been suggested that using a light virtual desktop pattern rather than a dark one can reduce the load on the backlight. This happens because lighter colors make the screen seem brighter and thus encourage users to use lower default backlight levels [41]. Furthermore, since most LCD's are "normally white," i.e. their pixels are white when unselected and dark when selected [72], the display of light colors consumes marginally less power than the display of dark colors. A similar strategy would be for the operating system or an application to decrease the resolution of a screen image by only illuminating a certain fraction of its pixels.

6.4 Summary

The display and backlight have several power-saving modes: dimming the backlight, turning off the backlight, changing the display from color to monochrome, reducing the update frequency of the display, and turning off the display. Only one transition strategy is currently used to take advantage of these, namely turning off the display and backlight after a fixed period of user inactivity. Only one load-change strategy is currently used to increase the use of these low-power modes, namely using a light virtual desktop pattern to allow use of lower backlight levels.

There are several things designers of display and backlight hardware can do to make software power management of these components more effective. Switching to low-power modes could be made unobtrusive. If an ambient light sensor were available, the operating system could automatically reduce the backlight level when ambient light brightened. Finally, as for all other components mentioned in this paper, designers of display and backlight hardware should seek to include as many low-power modes as possible that provide reduced but reasonable levels of functionality.

7 Other Components

7.1 Main memory

Main memory is generally implemented using DRAM with three modes: active, standby, and off. In active mode, the chip is reading or writing; in standby, it is neither reading nor writing but is maintaining data by periodically refreshing it. As an example of the power consumption in these states, 8 MB of EDO memory from Micron consumes about 1.65 mW in standby mode and 580 mW in active mode [52]. The only transition strategy currently used to reduce memory energy makes use of the off state: when it is determined that the entire system will be idle for a significant period of time, all of main memory is saved to disk and the memory system is turned off. The memory contents are restored when the system is no longer idle. When memory is saved in this manner, the machine state is said to be *suspended*; restoring memory is called *resuming*. Load-change strategies for saving memory power have been discussed before in the context of load-change strategies for saving processor power: they included using energy-aware compilers and using compact and efficient operating system code. Such strategies reduce the load on the memory system by making application and system code more compact and efficient, thus permitting greater use of the standby state. They may also convince the user to purchase a machine with less main memory, thus reducing the energy consumption of the memory system.

In future machines, memory may be divided into banks, with each bank able to turn on and off independently. Such capability broadens the ability of the operating system to manage memory energy. At times when the memory working set could fit in a small amount of memory, unused memory could be turned off. If the contents of a bank of memory were not expected to be used for a long time, they could even be saved to disk. Note that the expected period of idle time would have to be large to make up for the significant energy and time consumed in saving and restoring such memory. A related approach would be to compress, using standard data compression methods, the contents of memory, and turn off the unused banks; memory contents could be uncompressed either as needed or when activity resumed.

7.2 Modem

A modem can be transmitting, receiving, idle, or off. Some modems provide another state with power consumption between idle and off, called wake-on-ring; in this state, the modem consumes just enough power to detect an incoming call. MacOS uses no power saving strategies for the modem, relying on the user or an application to turn the modem on and off explicitly [45]. Presumably, this is because the operating system has no idea when data will arrive at the modem, and needs to make sure the modem is enabled whenever such data arrives so that it is not lost. A better strategy would be to have the operating system, or even the modem itself, switch the modem to the off or wake-on-ring state whenever there is no active connection to the modem.

7.3 Sound system

The sound system is another miscellaneous consumer of energy that can be active, idle, or off. MacOS uses an inactivity timer to decide when to turn the sound card off [45]. Another possibility is to turn the sound card off whenever a sound request from an application that triggered it turning on is completed; this has the disadvantage of increasing sound emission latency when one sound closely follows another.

8 Overall Strategies

It is possible to energy manage the entire computer as if it were a single component. When the computer is unneeded now and probably for some time, the operating system may put the entire system in a low-power state. Just how low-power a state depends on how long the system is expected to be idle since, in general, the lower the power of the state, the greater the time to return the system to full functionality.

Transition strategies for entering low-power system states generally use inactivity thresholds. If the user and all processes have been idle for some set period of time, the next lower system state is entered. APM 1.1 [34], a standard for energy management in computers, allows this decision-making process to be enhanced in at least two ways. First, the user may be allowed to make an explicit request to switch to a lower-power system state. Second, certain applications may be consulted before making a transition to a lower-power system state, so they can reject the request or make internal preparations for entering such a state.

Several low-power system states can be devised, and some examples of these are defined in APM 1.1. In the APM standby state, most devices are in a low-power state but can return to their full-power states quickly. For example, the disk is spun down and the backlight is off. In the APM suspend state, all devices are in very low-power states and take a relatively long time to return to functionality. For instance, the contents of memory are saved to disk and main memory is turned off. In the APM off state, the entire machine is off; in particular, all memory contents are lost and a long boot period is needed to return to functionality.

Note that the sequence with which low power states are entered is significant. For example, if the memory is copied to the disk before the disk is spun down, then the machine, or at least the memory, can be shut down without spinning up the disk to establish a checkpoint.

There are both a disadvantage and an advantage to energy managing the system instead of separately managing each component. The disadvantage is that it requires all components' energy management be synchronized. Thus, if one component is still active, some other inactive component may not get turned off. Also, if it takes microseconds to determine the idleness of one component but seconds to determine the idleness of another, the first component will not be energy-managed as efficiently as possible. The advantage of treating the system as a single component is simplicity. It is simpler for the operating system to make a single prediction about the viability of entering a system state than to make separate predictions for each component state. It is simpler for an application to give hints to the operating system about when state transitions are reasonable, and to accept and reject requests by the operating system to make such transitions. Also, it is simpler for the user, if he or she is called upon to make energy management decisions, to understand and handle a few system state transitions than to understand and handle an array of individual component transitions. For these reasons, an operating system will typically do both component-level and system-level energy management. For example, APM 1.1 has a system state called enabled, in which individual component energy management is performed. After extended periods of idleness, when most components can be managed uniformly, different low-power system states can be entered.

9 Conclusions

Computer hardware components often have low-power modes. These hardware modes raise software issues of three types: transition, load-change, and adaptation. Several solutions to these issues have been implemented in real portable computers, others have been suggested by researchers, and many others have not yet been developed. Generally, each solution targets the energy consumption of one component.

The disk system has been the focus of many software solutions. Currently, the main hardware power-saving feature is the ability to turn the motor off by entering sleep mode. The main existing software solutions consist of entering sleep mode after a fixed period of inactivity and caching disk requests to reduce the frequency with which the disk must be spun up. Other technologies may improve the energy consumption of the storage system further, but present new challenges in file system management. These technologies include flash memory, which can function either as a secondary storage cache or a secondary storage unit, and wireless communication, which can make remote disks appear local to a portable computer.

New low-power modes for the CPU have also presented software challenges. Currently, the main hardware energy-saving feature is the ability to turn the CPU off, and the main existing software solution is to turn the CPU off when all processes are blocked. Other software strategies include using energy-aware compilers, using compact and efficient operating system code, and forcing processes to block when they appear to be busy-waiting. An energy-saving feature that may soon appear in portable computers is the ability to reduce the processor voltage by simultaneously reducing the clock speed. Proposed solutions that take advantage of this feature have generally been interval-based, attempting to complete all processor work by the end of each interval. However, there may be effective non-interval-based solutions as well.

The display unit, including the backlight, typically consumes more power than any other component, so energy management is especially important for it. Power-saving modes available include dimming the backlight, turning the display and/or backlight off, switching from color to monochrome, and reducing the update frequency. Current system strategies only take advantage of the former two abilities, dimming the backlight and eventually turning off the display unit after a fixed period of inactivity. Other software strategies can be envisioned, especially if future hardware makes transitions to other low-power modes less obtrusive.

Other components for which software power management is possible include main memory, the modem, the sound system, and the wireless communication system. It is also possible to power manage the entire system as if it were a single component, bringing all components simultaneously to a low-power state when general inactivity is detected. Such system-level power management is simple to implement and allows simple communication with applications and users about power management; however, it should not completely supplant individual component power management because it requires synchronization of all components' power management.

To conclude, there are a few things we believe developers of future solutions to computer energy reduction should keep in mind.

1. A hardware feature is rarely a complete solution to an energy consumption problem, since software modification is generally needed to make best use of it.

2. Energy consumption can be reduced not only by reducing the power consumption of components, but also by introducing lower-power, lower-functionality modes for those components and permitting external control over transitions between those modes.

3. Standard operating system elements may need to be redesigned when dealing with low-power components that have different performance characteristics than the components they replace.

4. On a portable computer, the main goal of a component energy management strategy is to increase the amount of work the entire system can perform on one battery charge; thus, evaluation of such a strategy requires knowledge of how much energy *each* component consumes.

5. Evaluation of a power management strategy should take into account not only how much energy it saves, but also whether the trade-off it makes between energy savings and performance is desirable for users.

6. Seemingly independent energy management strategies can interact.

7. Finally, thanks to continual technological advances, there are many challenges ahead on both the hardware and software sides of energy management.

References

[1] Adelberg, B., Garcia-Molina, H., and Kao, B. Emulating soft real-time scheduling using traditional operating system schedulers. *Proceedings of the 1994 Real-Time Systems Symposium*, 292–298, December 1994.

[2] Apple Computer, Inc. *Inside Macintosh, Volume VI*, Addison Wesley Publishing Company, Reading, MA, 1991.

[3] Baker, M., Asami, S., Deprit, E., Ousterhout, J., and Seltzer, M. Non-volatile memory for fast, reliable file systems. *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS V)*, 10–22, October 1992.

[4] Barringer, B., Burd, T., Burghardt, F., Burstein, A., Chandrakasan, A., Doering, R., Narayanaswamy, S., Pering, T., Richards, B., Truman, T., Rabaey, J., and Broderson, R. InfoPad: a system design for portable multimedia access. *Proceedings of the Calgary Wireless 1994 Conference*, July 1994.

[5] Brewer, E., Burd, T., Burghardt, F., Burstein, A., Doering, R., Lutz, K., Narayansaramy, S., Pering, T., Richards, B., Truman, T., Katz, R., Rabaey, J., and Broderson, R. Design of wireless portable systems. *Proceedings of the IEEE International Computer Society Conference (COMPCON 95)*, 169–176, March 1995.

[6] Bertsch, J., Bernstein, K., Heller, L, Nowak, E., and White, F. Experimental 2.0 V Power/Performance Optimization of a 3.6 V-Design CMOS Microprocessor — PowerPC 601. *1994 Symposium on VLSI Technology Digest of Technical Papers*, 83–84, 1994.

[7] Cáceres, R., Douglis, F., Li, K., and Marsh, B. Operating system implications of solid-state mobile computers. *Technical Report MITL-TR-56-93*, May 1993.

[8] Caruthers, F. Hue and cry for color stirs flat-panel makers. *Computer Design, OEM Integration section*, 11–16, July 1994.

[9] Caruthers, F. Next-generation flat panels. *Computer Design, OEM Integration section*, 16–18, July 1994.

[10] Chan, E., Govil, K., and Wasserman, H. Comparing algorithms for dynamic speed-setting of a low-power CPU. *Proceedings of the First ACM International Conference on Mobile Computing and Networking (MOBICOM 95)*, 13–25, November 1995.

[11] Chandrakasan, A. P., Sheng, S., and Broderson, R. W. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, April 1992.

[12] Chetto, H. and Chetto, M. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, October 1989.

[13] Conger, J. *Windows API Bible*, Waite Group Press, Corte Madera, CA, 1992.

[14] Copeland, G., Keller, T., Krishnamurthy, R., and Smith, M. The case for Safe RAM. *Proceedings of the Fifteenth International Conference on Very Large Databases*, Amsterdam, 327–335, August 1989.

[15] Cox, D. C. Wireless personal communications: what is it? *IEEE Personal Communications*, 20–35, April 1995.

[16] Degermark, M., Engan, M., Nordgren, B., and Pink, S. Low-loss TCP/IP header compression for wireless networks. *Proceedings of the Second ACM International Conference on Mobile Computing and Networking (MOBICOM 96)*, 1–14, November 1996.

[17] Douglis, F., Kaashoek, F., Marsh, B., Cáceres, R., Li, K., and Tauber, J. Storage alternatives for mobile computers. *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation*, 25–37, November 1994.

[18] Douglis, F., Krishnan, P., and Bershad, B. Adaptive disk spin-down policies for mobile computers. *Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing*, 121–137, April 1995.

[19] Douglis, F., Krishnan, P., and Marsh, B. Thwarting the power-hungry disk. *Proceedings of the 1994 Winter USENIX Conference*, 293–306, January 1994.

[20] Duchamp, D. Issues in wireless mobile computing. *Proceedings of the IEEE Workshop on Workstation Operating Systems*, 2–10, April 1992.

[21] Ellis, S. C. Power management in notebook PC's. *Proceedings of the Silicon Valley Personal Computing Conference*, 749–754, 1991.

[22] Fazzio, D. P., Moser, M. A., Polson, C. J., and Scheffel, J. N. Head actuator dynamics of an IBM $5\ 1/4$-inch disk drive. *IBM Journal of Research and Development*, 37(4):479–490, July 1993.

[23] Feibus, M. and McCarron, D. Conquering notebook power. *OEM Magazine*, 60–69, April 1994.

[24] Forman, G. H. and Zahorjan, J. The challenges of mobile computing. *Computer*, 38–47, April 1994.

[25] Fox, A., Gribble, S. D., Brewer, E. A., and Amir, E. Adapting to network and client variability via on-demand dynamic distillation. *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, Cambridge, MA, 160–170, October 1996.

[26] Gary, S., Dietz, C., Eno, J., Gerosa, G., Park, S., and Sanchez, H. The PowerPC^{TM} 603 microprocessor: a low-power design for portable applications. *Proceedings of the IEEE International Computer Society Conference (COMPCON 94)*, 307–315, February 1994.

[27] Greenawalt, P. M. Modeling power management for hard disks. *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, 62–66, February 1994.

[28] Hansche, L. ROM BIOS: the best place for portable PC power-management features. *Proceedings of the Silicon Valley Personal Computing Conference*, 755–760, 1991.

[29] Harris, E. P., Depp, S. W., Pence, W. E., Kirkpatrick, S., Sri-Jayantha, M., and Troutman, R. R. Technology directions for portable computers. *Proceedings of the IEEE*, 83(4):636–657, April 1995.

[30] Helmbold, D. P., Long, D. D. E., and Sherrod, B. Dynamic disk spin-down technique for mobile computing. *Proceedings of the Second ACM International Conference on Mobile Computing and Networking (MOBICOM 96)*, 130–142, November 1996.

[31] Hewlett Packard. *Infrared IRDA Compliant Transceiver HSDL-1001 Technical Data*, on the World Wide Web at `ftp://ftp.hp.com/pub/accesshp/HP-COMP/ir/hsdl1001.pdf`, November 1996.

[32] Hospodor, A. D. and Hoagland, A. S. The changing nature of disk controllers. *Proceedings of the IEEE*, 81(4):586–594, April 1993.

[33] Imielinski, T., Viswanathan, S., and Badrinath, B. R. Energy efficient indexing on air. *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, 25–36, May 1994.

[34] Intel Corporation and Microsoft Corporation. *Advanced Power Management (APM) BIOS interface specification revision 1.1*, September 1993.

[35] Jain, R. and Werth, J. Airdisks and AirRAID: modeling and scheduling periodic wireless data broadcast. *Computer Architecture News*, 23(4):23–28, September 1995.

[36] Karlin, A. R., Manasse, M. S., McGeoch, L. A., and Owicki, S. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, June 1994.

[37] Kawaguchi, A., Nishioka, S., and Motoda, H. A flash-memory based file system. *Proceedings of the 1995 USENIX Technical Conference*, 155–164, January 1995.

[38] Klostermeyer, W. F. and Srinivas, K. Reducing disk power consumption in a portable computer. *Operating Systems Review*, 29(2):27–32, April 1995.

[39] Krishnan, P., Long, P. M., and Vitter, J. S. Adaptive disk spindown via optimal rent-to-buy in probabilistic environments. *Duke University Technical Note CS-1995-08*, Duke University, 1995.

[40] Kuenning, G. H., Popek, G. J., and Reiher, P. L. An analysis of trace data for predictive file caching. *Proceedings of the 1994 Summer USENIX Conference*, 291–303, June 1994.

[41] Kuty, D. Personal communication.

[42] Li, K., Kumpf, R., Horton, P., and Anderson, T. A quantitative analysis of disk drive power management in portable computers. *Proceedings of the 1994 Winter USENIX Conference*, 279–291, January 1994.

[43] Lin, H.-D., Yan, R.-H., and Yu, D. Improving CMOS speed at low supply voltages. *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 618–621, October 1994.

[44] Lorch, J. Modeling the effect of different processor cycling techniques on power consumption. *Performance Evaluation Group Technical Note #179*, ATG Integrated Systems, November 1995.

[45] Lorch, J. A complete picture of the energy consumption of a portable computer. *Masters Thesis*, Computer Science, University of California at Berkeley, December 1995.

[46] Lorch, J. and Smith, A. J. Reducing processor power consumption by improving processor time management in a single-user operating system. *Proceedings of the Second ACM International Conference on Mobile Computing and Networking (MOBICOM 96)*, 143–154, November 1996.

[47] Lucent Technologies. *WaveLAN/PCMCIA Card User's Guide*, on the World Wide Web at `ftp://ftp.wavelan.com/pub/pdf-file/pcmcia/pcman3x.pdf`, October 1996.

[48] Lucent Technologies. *WaveLAN/PCMCIA Card*, on the World Wide Web at `ftp://ftp.wavelan.com/pub/pdf-file/pcmcia/fs-pcm.pdf`, November 1996.

[49] MacDonald, J. Power management for 386DXL-based notebook computers. *Proceedings of the Silicon Valley Personal Computing Conference*, 735–748, 1991.

[50] Marsh, B., Douglis, F., and Krishnan, P. Flash memory file caching for mobile computers. *Proceedings of the 27th Hawaii Conference on Systems Sciences*, 451–460, January 1993.

[51] Maxtor Corporation. *251350, 251010 and 250840 Specifications*, on the World Wide Web at `http://www.maxtor.com/products.html`, February 1997.

[52] Micron Technology, Inc. *4 MEG x 16 Preliminary Datasheet MT4LC4M16R6*, on the World Wide Web at `http://www.micron.com/mti/marketing/pdfs/datasheets/129.pdf`, March 1997.

[53] Motorola. *PowerPC 603^{TM} RISC Microprocessor Technical Summary*, 1993.

[54] Noble, B. D., Price, M., and Satyanarayanan, M. A programming interface for application-aware adaptation in mobile computing. *Computing Systems*, 8(4):345–363, 1995.

[55] Parrish, T. and Kelsic, G. Dynamic head loading technology increases drive ruggedness. *Computer Technology Review*, 12(16):51–55, February 1993.

[56] Pietrek, M. *Windows Internals*, Addison Wesley, Reading, MA, 1993.

[57] Powers, R. A. Batteries for low power electronics. *Proceedings of the IEEE*, 83(4):687–693, April 1995.

[58] Ritchie, D. M. and Thompson, K. The UNIX time-sharing system. *Communications of the ACM*, 17(7):365–275, July 1974.

[59] Road Warrior International. *815 MB Slimline Hard Drive Functional Specifications*, on the World Wide Web at `http://warrior.com/arihd/815.html`, May 1996.

[60] Rosenblum, M. and Ousterhout, J. K. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, February 1992.

[61] Rulnick, J. M. and Bambos, N. Mobile power management for maximum battery life in wireless communication networks. *Proceedings of the IEEE Conference on Computer Communications (INFOCOM 96)*, 443–450, March 1996.

[62] Rulnick, J. M. and Bambos, N. Performance evaluation of power-managed mobile communication devices. *Proceedings of the International Conference on Communications (ICC/SUPERCOMM 96)*, 1477–1481, June 1996.

[63] Saltzer, J. H., Reed, D. P., and Clark, D. D. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[64] Satyanarayanan, M., Kistler, J. J., Mummert, L. B., Ebling, M. R., Kumar, P., and Lu, Q. Experience with disconnected operation in a mobile computing environment. *Proceedings of the USENIX Mobile and Location-Independent Computing Symposium*, 11–28, August 1993.

[65] Snyder, J. H., McKie, J. B., and Locanthi, B. N. Low-power software for low-power people. *Proceedings of the 1994 IEEE Symposium on Low Power Electronics*, 32–35, October 1994.

[66] Sohn, P. Personal communication.

[67] Srivastava, M. B., Chandrakasan, A. P., and Broderson, R. W. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(1):42–55, March 1996.

[68] Suzuki, N. and Uno, S. Information processing system having power saving control of the processor clock. *United States Patent #5,189,647*, February 1993.

[69] Tiwari, V., Malik, S., and Wolfe, A. Compilation techniques for low energy: an overview. *Proceedings of the 1994 IEEE Symposium on Low Power Electronics*, 38–39, October 1994.

[70] Toshiba America Information Systems. *MK2720 Functional Specifications*, on the World Wide Web at `http://www.toshiba.com/taisdpd/mk2720a.htm`, January 1997.

[71] Weiser, M., Welch, B., Demers, A., and Shenker, S. Scheduling for reduced CPU energy. *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation*, 13–23, November 1994.

[72] Werner, K. Flat panels fill the color bill for laptops. *Circuits and Devices*, 21–29, July 1994.

[73] Western Digital Corporation. *WD Portfolio Specifications*, on the World Wide Web at `http://www.wdc.com/acrobat/portfoli.pdf`, February 1997.

[74] Wood, C. and Hodges, P. DASD trends: cost, performance, and form factor. *Proceedings of the IEEE*, 81(4):573–585, April 1993.

[75] Wu, M. and Zwaenepoel, W. eNVy: a non-volatile, main memory storage system. *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI)*, 86–97, October 1994.

[76] Zivkov, B. T. and Smith, A. J. Disk caching in large database and timeshared systems. *Proceedings of the Fifth International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 97)*, 184–195, January 1997.

[77] Zorzi, M. and Rao, R. R. Error control and energy consumption in communications for nomadic computing. *IEEE Transactions on Computers*, 46(3):279–289, March 1997.