Power Aware Disk Scheduling Algorithms for Real-Time Systems

by

ADAM ROTH

A THESIS

Presented to the Faculty of
The New Mexico Institute of Mining and Technology
In Partial Fulfillment of Requirements
For the Degree of Master of Science in Computer Science

Under the Supervision of Dr. Xiao Qin

Socorro, New Mexico

December, 2005

Abstract

Improving the power consumption attributes of embedded and real-time systems has become an important area of interest as processors and other system components have become increasingly powerful and demanding in their energy consumption. The focus of this thesis is on examining the characteristics of real-time hard-disk scheduling algorithms, as hard-disk power usage can amount to a substantial portion of the total system power consumption. Conventional disk scheduling algorithms typically either disregard power consumption completely, or at best apply a fairly naïve power management policy. Given the potential implications for improving the efficiency and longevity of real-world disk systems, we investigate the problem of scheduling a set of independent real-time disk requests such that the total power consumption is minimized, while the efficacy of the disk system is not compromised. We define a power consumption model that can reasonably approximate the performance characteristics of real-world disks. Next, we discuss two power-aware power management policies, I/O Burstiness for Energy Conservation (IBEC) and Speed-Aware Real-time Disk Scheduling for energy conservation (SARDS), which integrate differing power management policies into the disk scheduling algorithms for real-time I/O-intensive applications. Furthermore, to evaluate the performance of the proposed algorithms against existing solutions and ensure that the efficacy of the system is not compromised, we incorporate the earliest deadline first (EDF) and least laxity first (LLF) scheduling policies into SARDS and IBEC to implement power-aware real-time scheduling algorithms. Experimental results from real-world traces and synthetically generated workloads show that the dynamic algorithms have the potential to substantially reduce power consumption over existing scheduling algorithms and power management policies without compromising the overall performance of disk systems.

Keywords: SARDS; IBEC; power-aware scheduling; real-time; guarantee ratio; power consumption; hard-disk; power modeling; simulation

Corresponding author's email: aroth@nmt.edu

Table of Contents

Chapter

LIST OF FIGURES

1. Introduction

As progressively more powerful solutions for portable and embedded computing platforms continue to be introduced, system power consumption is becoming a growing area of concern as energy costs rise and as battery technology has not been able to keep pace with the rapid increase in the demands made by these systems. A large fraction of power consumed in modern embedded and portable systems is consumed by the I/O subsystem, and specifically, the hard-disk drive(s). The hard-disk drive (HDD) can be responsible for upwards of 30% of the total power consumption in these systems [6][16]. Furthermore, most systems in use today apply either no power management policy to govern HDD usage, or at best use a very naïve approach to managing HDD power consumption [16]. We believe that by approaching power management policies from a more dynamic context, it is possible to substantially reduce the amount of power consumed by servicing HDD requests in these systems, and that doing so provides the potential to significantly improve the energy efficiency of the system. For this research, we focus on real-time applications, although the algorithms investigated can be extended to operate on any generic computational platform with a modicum of effort.

1.1 Real Time Systems

A real-time system can be loosely defined as any computational system in which a unit of work must be completed before a certain time-deadline expires. Real-time systems come in two broad classes, hard real-time systems and soft real-time systems. A hard real-time system is one in which the penalty for a failed deadline is considered fatal to the system (for example, the flight control system on an aircraft might be viewed as a hard real-time system). A soft real-time system is one in which failed deadlines are undesirable, but not fatal to the functionality of the system (for example, a streaming audio/video player could be viewed as such a system, where a delay in processing might produce an annoying skip but not disrupt the playback process as a whole) [10].

For our purposes, a unit of work is considered to be a single disk I/O request, and the deadline is specified in relative terms as the number of milliseconds that the disk system has to complete the request, starting from the time of issuance of the request. Requests in a real-time system are processed and if necessary reordered by a scheduler. It is the scheduler's job to examine the stream of requests, and reorder them in such a way as to ensure that all deadlines are met, if possible. If it is not possible for the deadlines of all requests to be successfully met, the scheduler may also elect to drop one or many requests to ensure the deadlines of the remainder requests [10]. Alternately, the task of deciding which requests to allow and which to drop can be deferred to an admissions controller, which is a piece of logic that examines requests before they are actually dispatched to the disk, and makes a best guess as to whether or not a request's deadline can be serviced given what it knows about the current state of the disk system, and drops those requests it determines cannot be successfully serviced.

In our study, the scheduler is considered to exist as an interface between the hard-disk and the rest of the computational system, with the only component "closer" to the disk than the scheduler being the admissions controller. To apply a more concrete example, imagine an operating system receiving a number of I/O requests from various processes running under it. The operating system composes these requests into a single task-stream, and then sends the stream off to the scheduler for processing. If a real-time scheduling algorithm is capable of successfully scheduling any task stream that can be successfully scheduled, then that algorithm is considered to be optimal. Two commonly used real-time scheduling algorithms are Earliest Deadline First (EDF) and Least Laxity First (LLF) [10]. EDF schedules requests based on their absolute deadlines, placing requests with earlier deadlines ahead of those with later deadlines. LLF schedules requests based on their laxity, which is the amount of time that exists between when it is estimated a request would complete if it were serviced immediately and the expiry of its deadline. Those requests with the smallest amount of lax-time are placed ahead of requests with a larger amount. Both of these algorithms are known to be optimal [10].

1.2 Power Management

We introduce the concept of a power management policy, which figuratively sits on top of the scheduler and scheduling algorithm and is capable of manipulating the hard-disk power state(s) and performing other actions such as delaying the execution of requests in the scheduled queue of requests.

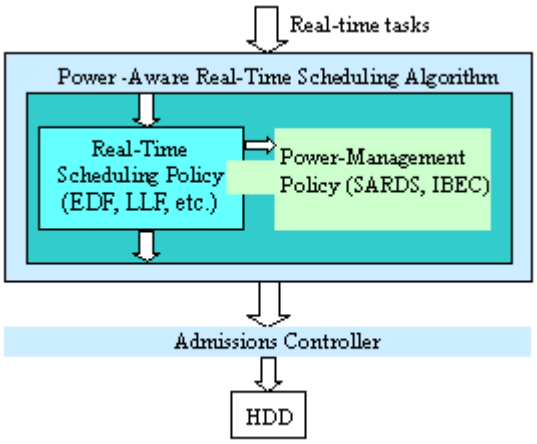

Fig 1. Abstract model of a power-aware scheduling algorithm [17]

Power management is accomplished in a variety of ways in both real-time and non-real-time systems. Most components now support a number of different power states that they can be made to operate in [12]. For example, a CPU may implement a feature such that when it is idle its clock-rate is reduced by a large factor which allows for the core voltage to be reduced, which reduces power consumption and thermal dissipation. In essence this is how the "SpeedStep" and "Cool'n'Quiet" features on Intel and AMD processors (respectively) work [5][9]. For the purposes of this research however, we are only concerned with hard-disks, which attempt to manage power in a similar though slightly less complex way.

As of this moment, most HDD's currently in deployment only explicitly support two operating power states, an active state and a sleep state ("implicit" states exist, for example a hard-disk may use more power when servicing a read or write request than it does when sitting idle in the active state, so technically this creates a third power

state, but it is not one that the power management policy can manually direct the disk to go into at will, and it is not one that exists for the purpose of regulating power consumption, it is simply an additional state whose existence is implied by the nature of the hard-disk and the way in which it operates). When in the active state the disk is fully operational and able to service requests. When in the sleep state, the motor which turns the disk platter is disabled, thus conserving energy but also rendering the hard-disk unable to service any requests until it is brought back to the active state [6][12][16]. Transitioning between these states is a process, which consumes a substantial amount of time and energy, especially when transitioning from the sleep state to the active state [12]. There are currently also hard-disks in the works and seeing limited deployment that support a larger number of power states and/or which allow the speed of the motor to be controlled in a fully dynamic way (enabling an essentially infinite number of power-states if viewed on a finely-grained scale) [7]. Most systems in active use today apply a fairly naive policy to control the transitions between these states, typically sleeping the hard-disk after it has been idle for longer than a certain threshold of time, and waking it as necessary [12][16]. Given the high cost associated with transitioning from one state to another, it is easy to see that this method might not be ideal.

1.3 IBEC and SARDS

By attempting to create dense clusters or bursts of disk requests (IBEC) and/or by dynamically controlling the HDD power-state in such a way that allows the workload to be serviced with the minimum amount of power being applied at any given time (SARDS), it should be possible to substantially reduce HDD power consumption when compared to traditional approaches. The specifics of these proposed algorithms will be discussed in depth in section 4 of this document. The objective of this thesis is to investigate dynamic power management policies under a variety of load patterns to determine their performance characteristics relative to existing solutions.

2. Related Work

A great deal of existing work has been done in the field of power modeling for various system components and also in the area of creating and evaluating scheduling algorithms both for real-time and non-real-time systems that can be applied to hard-disk requests or to any other form of schedulable task-stream [10]. There is also a rapidly growing breadth of work regarding the creation and evaluation of different power management policies for a wide array of application domains [6][7][8][12][15][16]. However, there as of yet has been comparatively little research done on the subject of comparing the power consumption characteristics of differing scheduling and power management policies when taken together in the context of a hard real-time system.

Some preliminary research has been done regarding techniques similar to those discussed in the IBEC policy which attempt to create bursty streams of disk requests in real-time and priority driven non-real-time systems [3][20]. Previous researchers have also examined the feasibility of dynamically scaling CPU voltages in hard real-time systems [14][18] and discussed a method for dynamically managing devices with multiple power states in hard real-time systems which is similar to SARDS but different in that it is not specific to hard-disks and also in that as opposed to a truly dynamic device, it requires a device with a finite set of distinct power states in order to function as specified [19].

There have also been multiple studies involving the use of a policy called DRPM, which is essentially similar to SARDS in that it attempts to reduce power consumption by manipulating HDD rotational speeds when paired with a multi-speed disk, but which is intended for use in non-real-time systems [4][11]. This prior research has shown promising results, indicating that the area is worth further exploration, as there does not yet exist a comprehensive body of research on this subject as applied to real-time systems relative to what exists in other domains.

2.1 Scheduling

For all the time and effort spent developing new and supposedly better scheduling algorithms, most consumer-level systems in use today still rely on the basic First-In, First-Out (FIFO, also referred to as "First-Come First-Served") scheduling algorithm to control the order in which the hard-disk services I/O requests. FIFO is a very simple (and arguably naive) algorithm that simply services requests in the order in which they are received, and its behavior is analogous to that of a simple queue.

With the very recent introduction of the SATA-IO (often erroneously referred to as SATA-II) specification this situation has been slowly changing. The SATA-IO specification allows for a feature called Native Command Queuing, which allows for the hard-disk controller to reorder the requests it receives in such a way as to make them more efficient to service.

The algorithm used to accomplish this is a variant of the Scan algorithm (also referred to as the Elevator algorithm). Scan works by ordering disk requests based upon the physical locations of the data on the hard-disk platter relative to the current position of the read/write head and the current direction of motion of the read/write head [1]. For example, let us assume that the disk has just serviced a request for data on cylinder number 18, and the next request resides on cylinder 27 (for the sake of brevity, we will identify requests by the cylinder number upon which the requested data resides). While servicing the request on cylinder 27, requests arrive for data on cylinders 2, 18, 19, 24, 40, and 63. The order the request will get serviced, once scheduled by the Scan algorithm, is 40, 63, 24, 19, 18, 2. This is because the current position of the read/write head is 27, and it is moving in an ascending direction with respect to the cylinder numbers. This means that initially only requests 40 and 63 are considered for processing, and they are serviced in ascending order. After request 63 is serviced, there are no remaining requests in the ascending direction, so the read/write head reverses direction, and services the remaining requests in descending order, starting at request 24 and working down to request 2. It is worth noting that enterprise-class solutions (and other systems in which I/O performance in crucial) have been using a very similar scheduling mechanism for years in conjunction with

SCSI-based hard-disks even though the technology has only recently made it into the mainstream consumer market [2].

There also exist a myriad of pre-emptive priority-based scheduling algorithms in this domain as well [10], although for the purposes of this research we are not concerned with these algorithms. It must be noted that none of the algorithms discussed so far are designed to be applied to real-time systems, although we discuss and evaluate them anyways as they provide an important and valuable comparison point to what is currently deployed in a large number of real-world systems. It is also important to note that the most commonly used scheduling algorithm in modern real-time systems is the EDF algorithm. Once again, this scheduling policy works by ordering requests in ascending order based on their absolute deadlines [10].

In order to give an example, it is necessary to first define a slightly more complex way of representing a disk request, so for now let a disk request be represented in the form of the 3-tuple {id, arrival_time, relative_deadline}, where 'id' is a unique identifier for the request (to simplify the discussion), 'arrival_time' is the time at which the request arrives, in absolute terms, and 'relative_deadline' is the amount of time the system has to process the request, measured in units of time from the request's arrival. Note that in a real system, additional fields would be necessary to specify the cylinder number, request size, etc., but for the purposes of our example this simplified version will suffice. Now, let us assume we have the requests {1, 100, 50}, {2, 105, 40}, {3, 120, 35}, and {4, 130, 10}. Under the EDF algorithm, the requests will be serviced in the order of 4, 2, 1, 3, as the absolute deadline of a request can be obtained by computing the sum of its arrival time and its relative deadline, and request 4 has the earliest absolute deadline, followed by request 2, 1, and then 3. EDF is a very commonly used algorithm thanks in part to its relatively simplicity and to the fact that it is an optimal algorithm.

The LLF algorithm (also referred to as Least Slack-Time First) operates in a similar way to EDF, except it orders requests based off of their slack time as opposed to their absolute deadlines. Note that this requires either prior knowledge of how long it should take to service a request, or a mechanism for estimating how long service should take, as laxity is computed as the request's absolute deadline minus the

current time minus the amount of time it will take to complete the request (so for example, if our current time is 90, and we have a request whose absolute deadline is 100 and whose time to completion is 3, then the request's laxity is 100 - 90 - 3, which gives 7) [10]. Because of this requirement, LLF is in general more complex to implement in real-world systems than EDF and can only be applied in a narrower range of situations. It should be noted that both the EDF and LLF algorithms are pre-emptive, meaning that if a new request arrives and its deadline is computed to be earlier than the currently executing request's is, then the currently executing request is suspended and the new request is allowed to execute immediately. This property is part of what ensures the optimality of these algorithms. As with the traditional scheduling algorithms, there exist priority driven algorithms and variations of these algorithms for scheduling tasks in real-time systems, but for our purposes we are not concerned with these algorithms [10].

2.2 Power Modeling

The power consumption characteristics of a hard-disk drive (or most any device) can be modeled using a simple finite state machine [11][12][13]. In our machine, states have values associated with them that indicate how much power is consumed per unit time while the device is in that state, and transitions between states have values associated with them that indicate how much power the device consumes during the transition, as well as how long transitioning between the two states takes in units of time.

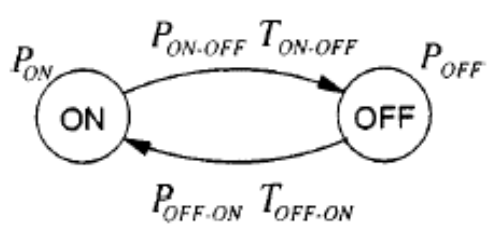

Fig 2. Example of a simple power-state model [11]

Note that this model allows for rules regarding power state usage to be encoded into the system (for example, if it is illegal to transition from the "sleep" state to the "active" state without first passing through the "warmup" state, this policy can be encoded into and enforced by the finite state machine), but it may not necessarily be ideal for modeling dynamically controlled systems or systems that have a very large number of power states. This is especially true if there exists a mathematical function or set of functions that can accurately approximate the power consumption characteristics of the system without needing to define and remember arbitrarily many states. Given this, when modeling a hard-disk that supports a more dynamic power management scheme than simply toggling between a sleep and an active state or a small number of well defined intermediary states, such as regulating the voltage applied to the hard-disk motor directly to attain the ability to spin the disk at virtually any speed between zero and the disks maximum supported speed, it is often more practical to instead use a simple mathematical model to simulate the hard-disk power consumption as opposed to a finite state machine [15].

A mathematical formula can approximately measure how fast the disk platter spins for any given input voltage, and similarly a mathematical formula can tell us how much energy is required to maintain the disk spinning at a given speed for a given amount of time, and also how much energy is expended when transitioning from speed x to speed y. Thus, we have two different methods for modeling hard disk power consumption within our real-time system. The finite state machine method is suitable when modeling disks similar to those that are currently in widespread use today, and that typically only support a fairly limited number of different power states. In contrast, the mathematical model is more appropriate for modeling disks that support a much finer-grained power management policy, even if such disks are not in real-world deployment as of this moment [7].

The desire to simulate such hypothetical disks is two-fold. First, it provides a valuable comparison point that shows how the algorithms we discuss will perform under a different paradigm, and secondly, it helps to cast some light onto how this potential future technology might compare in terms of performance to the technology that is currently most commonly used today. It should also be noticed that although

power management based on load-sensing is widely used on processors and has been discussed in terms of using it to manage hard-disk power management, real-world systems still typically apply a very naive power management policy, in which they will power down or otherwise reduce power being supplied to the hard-disk after it has been idle for a certain threshold of time [6]. Load-sensing has been discussed as a possible mechanism for improving this policy, although it still remains largely in the theoretical realm [12]. We will not delve too deeply into this concept, although it is worth noting that SARDS can be considered to be a load-sensing algorithm, as it relies on knowledge about the request-stream to make a best guess about what the minimal possible power-state it is possible to use while still guaranteeing the deadlines of all pending requests is.

3. Contributions

Our goal with this research is to evaluate the performance of two newly proposed power management and scheduling policies for real-time systems relative to existing solutions. Ultimately it is hoped that the information provided by this evaluation will show that it is possible to substantially reduce the amount of power consumed by the I/O subsystem in a real-time or non-real-time system without negatively impacting system performance when deadlines are important. We also hope to be able to propose policy guidelines regarding what sort of power management scheme makes sense in a given system and application domain. To quickly enumerate the specific contributions of this research:

1. We investigate two algorithms for power management and scheduling in real-time systems. We discuss the SARDS and IBEC power management and scheduling algorithms, and we evaluate their performance as compared to contemporary algorithms seeing real-world use today. These algorithms present improved ways of approaching hard-disk power management in real-

time systems, and can be extended for application in non-real-time systems as well.

2. Quantitative analysis of various scheduling algorithms and power management policies in the context of a real-time system. Through simulation and experimentation we record and compare a number of different performance metrics regarding several classical and new power management and scheduling algorithms under a variety of load patterns using both synthetically generated workloads as well as trace files gleaned from a number of real-world applications.

3. A simulation framework for modeling the power consumption characteristics of various types of hard-disk drives. In order to conduct experimentation it is necessary to apply the principles related to hard-disk power modeling to create a simulation environment that allows us to simulate the execution of a given workload on differing types of simulated hard-disks paired with a number of different power management and scheduling policies while recording and collecting power consumption related metrics.

4. A practical mechanism for reducing power consumption in real-world systems. We will show that it is possible to substantially reduce hard-disk power consumption over conventional methods under a number of different workloads by applying a more dynamic power management and scheduling policy such as SARDS or IBEC.

4. Research and Simulation

For the purposes of this research, we define a disk request as the 6-tuple {'id', 'arrival_time', 'cylinder', 'size', 'type', 'deadline'}, where 'id' represents the request's unique identifier (used internally to track individual disk requests), 'arrival_time' represents the absolute time at which the request arrives, 'cylinder' denotes the hard-disk cylinder number upon which our desired data resides, 'size' denotes the number of bytes of data that we are to read or write, 'type' denotes whether we are dealing with a read or a write request, and 'deadline' indicates the deadline that the system has to complete the request, in relative terms. Although modern hard-disk geometry allows for a much finer grained specification of the location of a desired piece of data than just the cylinder number, as we are primarily concerning ourselves with the power consumption characteristics of the system for this research, specification of just the desired cylinder provides enough granularity for us to gather the data that we are interested in.

4.1 Hard-Disk Power Model

We use two different mechanisms to model hard-disk power consumption. The first mechanism is a finite state machine representation built around real performance data gleaned from an IBM DTTA 350-640 hard-disk drive [12], and the second is based upon the mathematical relationship between a hard-disks speed of revolution and the power required to maintain it spinning at that speed [15].

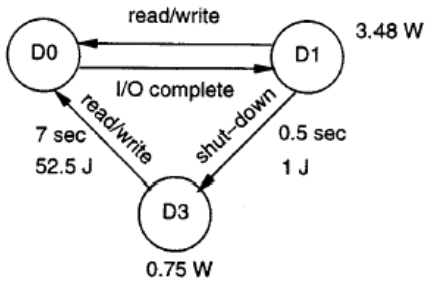

Fig 3. Power-state model of the IBM DTTA 350-640 HDD with real-world values [12]

DISK PARAMETERS: SUBSCRIPTS sd AND wu DENOTE SHUT DOWN AND WAKE UP, RESPECTIVELY

Model	P_{Off}	P_{On}	T_{sd}	E_{sd}	T_{wu}	E_{wu}
	Watt	Watt	sec	J	sec	J
IBM	0.75	3.48	0.51	1.08	6.97	52.5

Fig 4. A tabular representation of the above finite state machine [12]

The reason for using two separate models is because most of the algorithms we will be investigating are designed such that they are only aware of the "sleep" and "active" power states, and this characteristic is most easily matched with the finite state machine model. SARDS, on the other hand, requires the ability to dynamically manipulate the disk's power consumption on a much more finely grained scale than just toggling between a "sleep" state and an "active" state, and this necessitates the use of a mathematical model to approximate this capability. It should furthermore be noted that we want data obtained when simulating under one model to be directly comparable to data obtained when the simulation is run using the other model. In order to accomplish this, it is necessary to normalize the first power model such that the power consumed per unit time when the disk is in the sleep state is identical to the power consumed under the second model when the disk is in an equivalent state and also such that the power consumed per unit time under the first model when the disk is in the "active" state is identical to the power consumed under the second model when the disk is spun at its maximum supported speed (which is analogous to what happens in the "active" state). Values must be chosen carefully so as to preserve the relationship that exists between the power states in the original finite-state machine, and all other values in use in the machine must be scaled accordingly to ensure that the model remains accurate.

Note that as all we care about is relative variance in power consumption from one algorithm to the next it does not matter if the normalization process changes the absolute value of the result (which it will) so long as the relationships that exist in the original power state model between the power consumed in one state versus the power consumed in another versus the power used when transitioning from one state

to another are preserved. Thus, as a result of this normalization process, it becomes possible to compare data produced by the first power model directly against data produced under the second power model. For the purposes of our experiments, every algorithm except for SARDS will be tested using the finite state machine based power consumption model and SARDS will by necessity use the mathematically derived power consumption model.

4.2 IBEC Algorithm

The IBEC algorithm is based upon the idea that by delaying the execution of some requests it is possible to allow the disk to remain in a "sleep" state for a longer duration of time while processing larger contiguous blocks of requests when the disk is active, thus making better use of the disk when it is being fully powered. The consequences of this are twofold. First, we improve the efficiency of the disk by reducing the amount of time that it sits idle in the "active" state. Second, as a direct consequence of this policy we also reduce the sum total number of power state transitions that the disk will make when servicing a given request-stream.

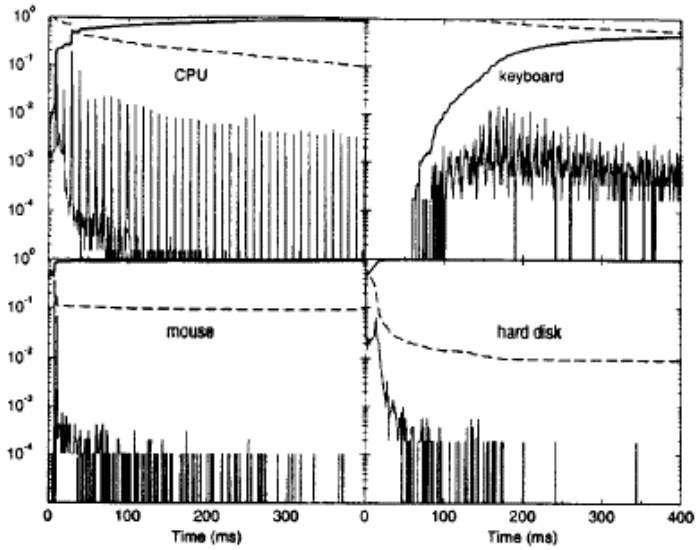

Fig 5. Typical request inter-arrival times for various devices, IBEC relies on being able to take sporadically arriving requests and grouping them so as the reduce the inter-arrival time [12]

This is important because transitioning from the "sleep" state to the "active" state is a process that is expensive both in terms of power consumed and the amount of time that it takes to complete this transition [12], so by reducing the number of these transitions it is possible to substantially improve the power consumption characteristics of the system. The general concept is that if a request arrives while the disk is in the sleep state, and if we determine that the request's deadline does not require immediately waking the disk, then we do not wake the disk, and instead cache the request for later process.

In fact, under the IBEC algorithm, if the disk is in the "sleep" state, we do not wake it until we determine that we must do so in order to guarantee the deadline of the waiting disk request(s). Note that it is probable that while we are waiting to wake the disk to service our waiting request other requests will continue to arrive, and that these requests are also queued until the disk is transitioned to the "active" state, at which point all waiting requests are serviced at once. This is the mechanism by which IBEC reduces the sum total number of necessary state transitions, and also how it attempts to create contiguous blocks of requests to service out of a potentially arbitrarily sporadic stream of requests. Also note that as more requests end up queued and consequently delayed, it is necessary to not just ensure that we guarantee the deadline of the first request in the cache, but also to ensure that by delaying the first request we are not causing the deadlines of subsequent requests to become unrealizable. Thus it may quite often be the case that we will need to transition to the "active" state some length of time before it is absolutely imperative to do so just to meet the deadline of the first request so that we can still hope to guarantee the deadlines of subsequent requests. This imposes a couple additional requirements on the algorithm. First, it becomes necessary to re-evaluate the threshold at which we must transition the disk to the "active" state whenever a new request is added to the cache. Second, there must be a mechanism for estimating the amount of time spent in serving a given request given what we currently know about the state of the system. The first criteria is fairly trivial, the second is a bit less so, however, the estimation does not need to be exact, so long as it is always greater than or equal to the worst-case service time of the given request.

It is obviously better to be as close to exact as possible, but as long as our estimation is never less than the actual value we can generally assume that we are not causing deadlines to be missed by delaying a series of disk requests, although it is possible to posit scenarios under which in is not possible to avoid missing a deadline when applying this power management policy. As we will see, this sort of situation actually occurs only extremely rarely in practice. A simple way to estimate the worst-case service time of a request while remaining reasonably accurate within the bounds of the system is to simply assume that the request we are estimating for will require seeking the read/write head entirely across the disk, from the first cylinder to the last, and that once it gets there we will have to wait for one full revolution of the hard-disk before the data we are interested in passes under it and reading/writing can commence. Assuming that the disk we are working with performs consistently in terms of its read and write speeds, which for the purposes of our simulation it does, we then have an estimate that is not only reasonably cheap to compute but which should also always exceed the actual service time of any real disk request without being so far off from the actual value as to render such estimation pointless.

It should also be noted that by itself, IBEC is a power management policy, not a scheduling algorithm. It is meant to be implemented on top of or in conjunction with a separate scheduling algorithm, which should be a real-time scheduling algorithm such as EDF or LLF. As requests arrive, they are first scheduled according to whichever algorithm is being used, and then processed by the IBEC power management policy. A high-level algorithmic description of IBEC follows (note that this description assumes that any arriving requests have already been properly scheduled by the associated scheduling algorithm):

```
If there are no requests active or pending
        Sleep the disk
Else
        If the disk is active
                Dispatch any requests that have arrived
        Else
                While deadlines can be guaranteed
```

8. Cache any requests that have arrived
9. Wake the disk
10. Dispatch any waiting requests

So when viewed from a high-level perspective, the IBEC algorithm is fairly simple. Note however that certain steps, such as verifying that deadlines can still be guaranteed as new requests are added to the cache, require a bit of thought when actually being implemented. Also note that it may also be desirable to allow the disk to idle for some threshold of time before sleeping it as opposed to immediately sleeping the disk as soon as there are no requests active or pending. This is due to the relatively large amount of power consumed in transitioning from the "sleep" state back to the "active" state, which creates a situation in which it could conceivably be more efficient to allow the disk to sit idle for a brief period of time in hopes that more requests will arrive. The exact threshold at which it is no longer beneficial to allow the drive to sit idle is something that is a function of the disk's power consumption characteristics as well as the relative distribution of requests in the request-stream, and as such this is something which can be computed if we know details about our request-stream in advance or if we have some way of making a best-guess based on observation of the request-stream at runtime. This however is beyond the scope of both this research and the IBEC algorithm in its current incarnation.

4.3 SARDS Algorithm

The SARDS algorithm is designed to operate in conjunction with hard-disks that support either fully dynamic power management in the form of voltage regulation that allows the hard-disk to be spun at virtually any speed between zero and its maximum possible speed, or that at the very least support some number of intermediate power states. Such disks are called multi-speed disks, and although a fairly recent development it is entirely possible that these kinds of disks will become more commonplace as time goes by. Most disks in current deployment however are not multi-speed disks [15].

The basic concept behind the SARDS algorithm is that it attempts to run the drive at the slowest possible speed that can be used while still guaranteeing the deadline of a request. Given that the amount of power needed to spin a hard-disk at a given speed grows as the square of that speed, it is possible to save a substantial amount of power by reducing the speed at which the disk spins [11][15]. This also has the added benefit of reducing the impact and severity of state transitions, because as there are now a potentially arbitrary number of intermediate states the "distance" traveled in a single transition will on average be less than that of a full transition from the "sleep" state to the "active" state that is required in conventional hard-disks, which implies less of a penalty in terms of power consumption and the time required to complete the transition. While running the disk at a reduced speed can be effective at saving power, it also degrades the performance of the hard-disk in terms of the amount of time required to service a given request [15]. To put it bluntly, the slower the drive is allowed to spin, the longer the amount of time required to complete any given request becomes.

Obviously this has the potential to become a problem when working with a real-time system that imposes deadlines on request processing time, as it is necessary to ensure that a request's deadline can be guaranteed given the current speed of the hard-disk, or that if the deadline of the request cannot be met at the current speed, the speed is increased if there exists an attainable speed that will allow the deadline of the request to be met. As in the case of IBEC, this can be handled by estimating the worst-case service time of a given request, although unlike in IBEC, a more complex method of estimation must be used. Because our hard-disk now supports an arbitrary number of differing power states, and because all things remaining equal different states will offer different service times for the same request, it is necessary to not only be able to estimate the worst-case service time of a request given the current power state, but it is also necessary to be able to estimate the worst-case service time of the same request in any of the other possible power states. This is not exceptionally difficult however, as the only variable that changes from one power state to the next is the rotational speed of the disk, so all we need is the ability to recompute those parts of the estimated service time that depend on the rotational speed. Therefore, we

can again start with the assumption that our worst-case request will require seeking the read/write head completely from one end of the disk to the other, followed by waiting for a full revolution of the disk platter to complete before reading/writing of data can begin. This part does not change.

Now, the read/write speeds as well as the amount of time necessary for one full rotation of the disk platter vary linearly with the speed of the hard-disk rotation, and since we already know what these values are when the disk is spinning at full speed (recall that IBEC requires this computation for its estimation process), one can simply compute the ratio of the current speed to the maximum speed and then multiply the full-speed variants of these values by this ratio to determine what they would be for any given rotational speed of the disk. Given this, we have a computationally cheap and reasonably accurate mechanism for estimating the worst-case service time of a request for any given power state/rotational speed that the disk supports, which is exactly what the SARDS algorithm requires in order to function properly. Again, the more accurate that this estimation can be made the better, but the most important thing is that it never underestimate the service-time of a request, and given a consistently performing hard-disk drive, this method of estimation should never produce a result that is less than what the actual value ends up being. Similarly to IBEC, SARDS is best viewed as a power management policy, and is meant to be applied on top of or in conjunction with a separate scheduling algorithm designed to function in a real-time environment such as EDF or LLF. Requests are first scheduled by this algorithm before being processed by SARDS. A high-level description of the algorithm follows (again note that we assume that any arriving requests have already been properly scheduled):

```
For each pending request R[i]
        Compute the minimum speed necessary to service R[i]
    If this speed is greater than the maximum possible speed
        Drop R[i]
    Else
        Transition to the minimum speed to service R[i]
        Dispatch R[i]
```

It is worth mentioning that SARDS essentially contains a built-in admissions controller (recall for our earlier discussion that an admissions controller is a device which ensures that only requests whose deadlines can be reasonably guaranteed are dispatched to the disk), in that it will drop any request that it determines cannot be serviced. IBEC does not do this, nor do the other real-time scheduling algorithms that we discuss [10]. Also worthy of mention is the fact that the central task of computing the minimum rotational speed needed to service the request successfully which is fundamental to the SARDS algorithm is not necessarily a trivial undertaking, especially if working with a truly dynamic multi-speed disk, as there may be an arbitrary number of different power states that require searching. It makes sense in this case to partition the search space in a fairly coarsely grained way, as from a practical standpoint there is little difference between spinning the disk at 3600 RPM and spinning it at 3601 RPM, so it doesn't make sense to bother considering them both as possible options. We can still offer good quality of service by only considering a limited subset of the possible speeds, and selecting whichever element of this subset comes closest to the optimal value without being less than the optimal value (because then we would fail the deadline), and this simplifies our search substantially.

Note that there are some fairly conspicuous issues which can be pointed out in the SARDS algorithm as specified, such as the lack of any policy for sleeping the hard-disk completely, or even more importantly, the lack of a step to evaluate the queue of requests as a whole to ensure that reducing the speed of the disk to service one request will not create a situation under which the deadlines of subsequent requests becomes unrealizable. The purpose of this research is to simulate and evaluate the performance of the SARDS algorithm as specified however, so this is the algorithm used in experimentation. A discussion of these and other issues, as well as proposed refinements to the algorithm, follows in sections 5 through 7.

4.4 Simulation Goals, Configuration, and Parameters

Through simulation we want to show how the performance if IBEC and SARDS compare to existing and well-established algorithms in terms of power consumption performance as well as in terms of the ability to successfully schedule a stream of disk requests without failing any more deadlines than is absolutely necessary. In other words, we want to demonstrate that SARDS and IBEC preserve the property of optimality when used in conjunction with an optimal real-time scheduling algorithm.

To accomplish this we will use a number of synthetically generated workloads as well as traces generated from instances of several real-world applications, namely cholesky, db2, dmine, lu, pgrep, and titan. In the cases of the synthetically generated workloads, we will generally choose settings which result in the creation of a workload that is not entirely schedulable, which is to say it will not be possible to successfully service all disk requests without either dropping some or allowing deadlines to fail on others. This is to ensure that under heavy load conditions, SARDS and IBEC perform at least as well as competing algorithms. As a number of non-real-time scheduling algorithms will also be tested to provide another set of comparison points, an admissions controller will be used that will attempt to estimate the service time of each request being dispatched to the disk, and drop any requests whose deadlines cannot be guaranteed. Note that this happens after the requests have been scheduled by whichever scheduling algorithm is being simulated. This is to ensure a degree of "fairness", by preventing the non-deadline-aware algorithms from creating a condition under which by placing a number of unserviceable requests at the start of the queue and dispatching them to the disk they therefore render it impossible for the deadlines of any subsequent requests to be realized, which would cause their performance to appear artificially poor.

In addition to various scheduling algorithms, we will also run tests involving differing power management policies. Specifically, SARDS and IBEC will be compared to the two most widely applied naive power management schemes currently in use, a scheme which transitions the disk to the "sleep" state whenever it becomes idle and wakes it immediately whenever there is work to process (henceforth, use of this policy shall be denoted by prefixing the character "P" to the name of a scheduling algorithm, and is meant to be interpreted as "naive power

management on top of scheduling algorithm X", so for example, PSCAN means that this power management technique is being used in conjunction with the SCAN scheduling algorithm), and a scheme which transitions the disk to the "sleep" state whenever it has been idle for greater than a certain threshold of time, and wakes it immediately whenever there is work to process (henceforth, use of this policy shall be denoted by prefixing the characters "DP" to the name of a scheduling algorithm, and is meant to be interpreted as "naive delayed power management on top of scheduling algorithm X", so for example, DPSCAN means that this power management technique is being used in conjunction with the SCAN scheduling algorithm). The specific scheduling algorithms being evaluated are FIFO, SCAN, EDF, and LLF. Because SARDS and IBEC require an underlying scheduling algorithm in order to function, and because the underlying algorithm is required to be a real-time algorithm, we will test these algorithms both in conjunction with the EDF and LLF scheduling policies. When EDF is being used, the character "E" will be suffixed to the algorithm name (so SARDSE should be interpreted as "SARDS with EDF scheduling"), and when LLF is being used, the character "L" will be suffixed in its stead (so IBECL should be interpreted as "IBEC with LLF scheduling").

To enumerate the combinations of power management policies and scheduling algorithms that we will be testing, we are evaluating FIFO, SCAN, EDF, LLF, PFIFO, PSCAN, PEDF, PLLF, DPFIFO, DPSCAN, DPEDF, DPLLF, IBECE, IBECL, SARDSE, and SARDSL. In our synthetically generated testcases, we will vary a number of parameters, one at a time, so that we may see what effect, if any, they have on the performance of the scheduling and power management policies, and more importantly, what effect, if any, they have on SARDS and IBEC, which are the algorithms that we are most interested in.

The different variables we will test are request arrival rate (varied from 0.05 to 1), average request deadline (varied from 5.5 ms to 25,000 ms), request distribution (statistically normal, sparse, and densely clustered workloads will be tested), and average request size (in terms of how much data is to be read from or written to the disk, varied from 320 bytes to 41,175,040 bytes). To understand what it means for arrival rate to vary from 0.05 to 1 (the rest of the variables can be understood in a

straightforward way), it is necessary to briefly discuss the program used to generate the synthetic workloads. This program is configured with a number of parameters that describe properties of the workload that we want to generate, such as what the minimum and maximum allowed deadlines or request sizes should be (it will then generate requests with deadlines that fall randomly between the minimum and maximum bounds), or whether it should generate a statistically normal load pattern versus a very sparse or densely clustered one, and when run it will generate a file containing synthetic disk requests that fit the profile of the specified workload. When generating a statistically normal workload, the arrival rate specifies the probability of a request occurring at any given timestep (the actual size of the timestep used is another configurable parameter of the generator program). Thus, if the arrival rate value is set to 1, this means that at every timestep, there is a 100% chance of a new request arriving, and if the arrival rate is 0.05, it means that at every timestep there is a 5% chance of a new request arriving, and so on, and that is how the arrival rate parameters should be interpreted in this context. Note that the following table specifies our range of experimental parameters:

Experimental Parameter	Values
Timestep	1*
min_deadline	20*, 1, 10, 50, 100, 250, 1000, 2000, 5000, 10000, 15000
max_deadline	100*, 10, 50, 100, 250, 1000, 2000, 5000, 15000, 20000, 35000
arrival_rate	0.5*, 0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95, 1.0
distribution	Normal*, Clustered, Sparse
min_request_size	1024*, 128, 512, 1024, 4096, 16384, 32168, 128672, 514688, 1029376, 2058752, 8235008, 16470016
max_request_size	32167*, 512, 1024, 4096, 16384, 32168, 128672, 514688, 1029376, 2058752, 8235008, 16470016, 65880064
read_percentage	0.66*
min_cluster_size	5*, 25, 100, 500
max_cluster_size	7*, 50, 500, 1000
sparse_idle_threshold	400*, 50, 500, 1500, 4000
num_cylinders	64*
num_entries	20000*

Fig 6. Table of experimental parameters, entries denoted with * indicate default values

Unless otherwise specified, our unit of time is the millisecond, and our unit of data-size is the byte. For each variable that we test, we only modify one of the above settings (or one pair of min/max settings) while allowing the remainder of the settings to remain constant. Note that when a range of variation is specified for a parameter that has a minimum and a maximum attribute, the specified range of variation describes the range by which the average of these two values varies. Also note that 'num_cylinders' and 'num_entries' pertain to the disk geometry that we are generating requests for and to the total number of requests that we are generating for the current workload, respectively. Also note that the 'min_cluster_size', 'max_cluster_size', and 'sparse_idle_threshold' parameters are only used when a distribution mode other than statistically normal is used, and also that an exhaustive list of all settings used to generate each and every synthetic workload used for simulation is available in the supplemental Excel spreadsheet document included as part of this thesis.

For the purposes of this research, the metrics that we are interested in are guarantee ratio, which indicates how well a given algorithm performed at meeting request deadlines (a guarantee ratio of 1 indicates that all deadlines were met successfully), and normalized power consumption, which indicates how much power an algorithm consumes relative to the other algorithms (a normalized power consumption value of 1 indicates that an algorithm used as much as or more power than any other algorithm, and values closer to 0 indicate increasing levels of power efficiency relative to the least efficient algorithm(s)).

We also record a number of other metrics, such as average request service time, average request waiting time, and so on, but we do not discuss these metrics in depth as they are not directly related to the research at hand. So for each algorithm, for each variable, for each testcase related to that variable (and also including the real-world traces), we want to run the simulation and record our performance metrics. A high-level description of the simulation algorithm used follows:

1. While requests remain in the workload queue
2. Move all requests that have arrived at the current timestep to the request queue
3. Pass the request queue to the scheduler/power management

policy for processing

4. Dispatch any scheduled requests returned by the scheduler/power management policy to disk
5. As service completes for each request, compute the necessary metrics and place the request in the completed queue
6. Advance the simulation timestep to the next increment
7. Perform any necessary post-processing and write the completed queue to file for analysis

Note that this is a very high-level description, and that at each step there may be additional tasks being carried out to support the simulation. For example, in step 4 when requests are dispatched, they are not sent directly to disk but are passed through the admissions controller, which processes the scheduled requests to ensure that their deadlines can be guaranteed, and then passes them off to the disk (or drops any requests whose deadlines cannot be realized). Also note that complete source code for the simulator and all other related modules is available for review in the source archive that is included as part of this thesis.

The timestep parameter used for all simulations was 150 ms. This is an important value, as it influences how many requests are dispatched to the scheduler at any given time, and this can have a significant impact on the performance of scheduling algorithms that rely on having multiple requests to work with at once, such as SCAN. For example, if the timestep were set to 1, then unless multiple requests arrived at the exact same millisecond, the scheduler would never receive more than one request at a time, and SCAN scheduling would essentially become indistinguishable from FIFO scheduling. In all cases, simulation was carried out on a system matching the following general specifications:

- MSI K8N Neo4 Platinum Mainboard (MS-7125) w/ BIOS version 1.9
- AMD Athlon64 X2 3800+ CPU @ 2,420 MHz
- 1 GB PC4000 RAM @ 261 MHz, dual channel
- 2x WD Raptor 37 GB HDD's in RAID-0
- Windows XP Professional w/ SP2
- Eclipse 3.1
- Java 1.5.0
- Java VM started with default settings

5. Equations

Here we discuss some of the key equations used in the simulation process. Note that this is an overview of the most critical and least intuitive equations, and not an exhaustive list. We model power consumption of the multi-speed disk based on the relationship:

$$P = V^2 / R,$$

where 'P' is the power consumption, V is a function of the square of the disk's rotational speed (and corresponds to voltage being applied to the motor), and R is the resistance of the motor.

We compute the amount of power needed to service a given request as:

$$P_{request} = P_{lead\text{-}in} + P_{service},$$

where $P_{lead\text{-}in}$ is the amount of power consumed by the disk while idling or sleeping between the completion of the previous request and the arrival of our current request, plus the penalties of any state transitions that have occurred during this time, and $P_{service}$ is the amount of power consumed by the disk as it services the request (our power model specifies how much power the disk consumes per unit time in a given state, as well as what penalties are assessed for state transitions). Our total simulation power consumption is the summation of these values across all requests in the simulation.

We compute the time spent servicing the request as:

$$T_{request} = T_{seek} + T_{service},$$

where T_{seek} is the amount of time spent seeking to the location of the desired data, and

$T_{service}$ is the amount of time spent actually reading from or writing to the disk. We compute T_{seek} as:

$$T_{seek} = T_{rot} + (0.6 * \text{sqrt}(N)),$$

where T_{rot} is the rotational delay involved in this request (the amount of time we must wait for the disk platter to spin such that the data we are interested in passes under the read/write head) and computed to be one-third of the time for the disk platter to complete one full revolution at its current speed, and N is the number of cylinders that the read/write head must move to service the request. We compute $T_{service}$ as:

$$T_{service} = (S / D) * 1000,$$

where S indicates the size of our request in bytes (in terms of amount of data being read/written), and D indicates the disk's read speed or write speed (as appropriate to the type of our current request) in terms of bytes per second. We multiply by 1000 to convert to milliseconds.

We compute the guarantee ratio metric as:

$$G = (R_{tot} – (R_{drop} + R_{fail})) / R_{tot},$$

where R_{tot} is the total number of requests, R_{drop} is the total number of dropped requests, and R_{fail} is the total number of requests with failed deadlines and G is the guarantee ratio.

We compute the normalized power consumption metric of a simulation by taking the maximum absolute power consumption value across all algorithms used in the given simulation and then dividing all power consumption values by this number. This scales the power consumption into a range between 0 and 1, inclusive, where 1 indicates the most power-intensive algorithm(s), and smaller numbers indicate increasing levels of efficiency.

6. Results

Our experimental simulations show some very interesting trends, and expose areas in which the SARDS algorithm in particular can be improved. These general trends and characteristics will be discussed later, for now we focus on the results of the individual experiments.

6.1 Arrival Rate

Let us start by discussing the experiment in which the request arrival rate was varied. Figs. 7 and 8 plot the guarantee ratios and power factors of all the tested algorithms.

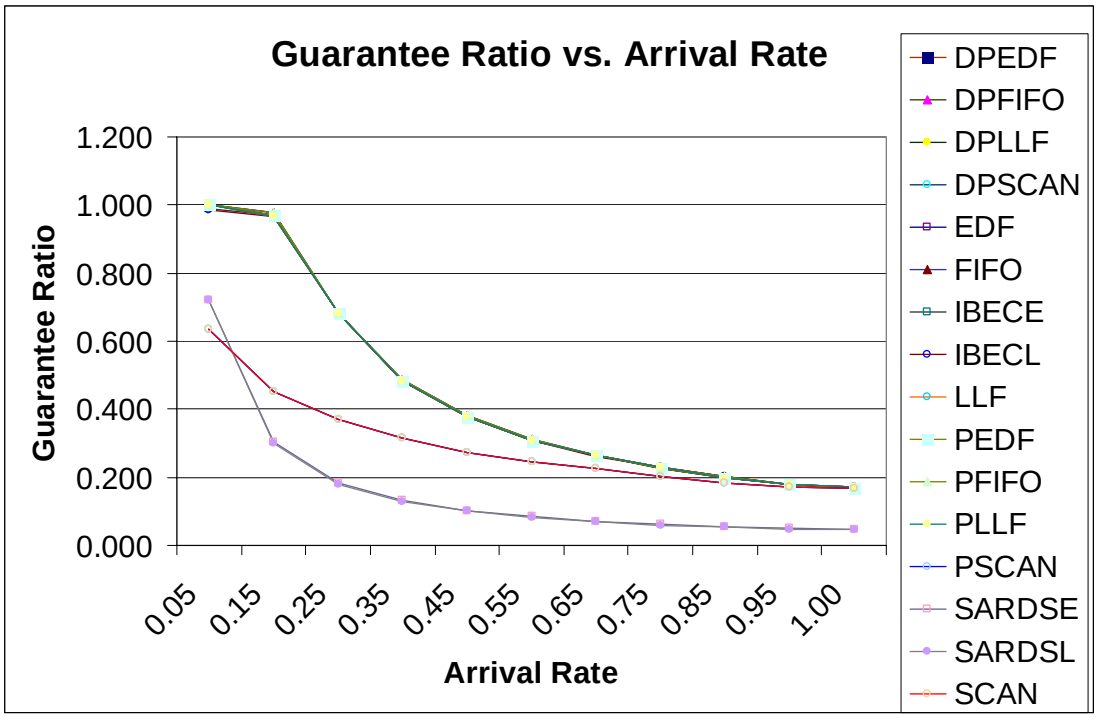

Fig 7. Guarantee ratio for different arrival rates, chart and table

	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95	1.00
DPEDF	1.00	0.97	0.68	0.48	0.38	0.31	0.26	0.23	0.20	0.18	0.17
DPFIFO	1.00	0.98	0.68	0.49	0.38	0.31	0.27	0.23	0.20	0.18	0.17
DPLLF	1.00	0.97	0.68	0.48	0.38	0.31	0.26	0.23	0.20	0.18	0.17
DPSCAN	0.64	0.45	0.37	0.31	0.27	0.25	0.22	0.20	0.18	0.17	0.17
EDF	1.00	0.97	0.68	0.48	0.38	0.31	0.26	0.23	0.20	0.18	0.17
FIFO	1.00	0.98	0.68	0.49	0.38	0.31	0.27	0.23	0.20	0.18	0.17
IBECE	0.99	0.97	0.68	0.48	0.38	0.31	0.27	0.23	0.20	0.18	0.17
IBECL	0.99	0.97	0.68	0.48	0.38	0.31	0.26	0.23	0.20	0.18	0.17
LLF	1.00	0.97	0.68	0.48	0.38	0.31	0.26	0.23	0.20	0.18	0.17
PEDF	1.00	0.97	0.68	0.48	0.38	0.31	0.26	0.23	0.20	0.18	0.17
PFIFO	1.00	0.98	0.68	0.49	0.38	0.31	0.27	0.23	0.20	0.18	0.17
PLLF	1.00	0.97	0.68	0.48	0.38	0.31	0.26	0.23	0.20	0.18	0.17
PSCAN	0.64	0.45	0.37	0.31	0.27	0.25	0.22	0.20	0.18	0.17	0.17
SARDSE	0.72	0.30	0.18	0.13	0.10	0.09	0.07	0.06	0.05	0.05	0.05
SARDSL	0.72	0.30	0.18	0.13	0.10	0.08	0.07	0.06	0.05	0.05	0.05
SCAN	0.64	0.45	0.37	0.31	0.27	0.25	0.22	0.20	0.18	0.17	0.17

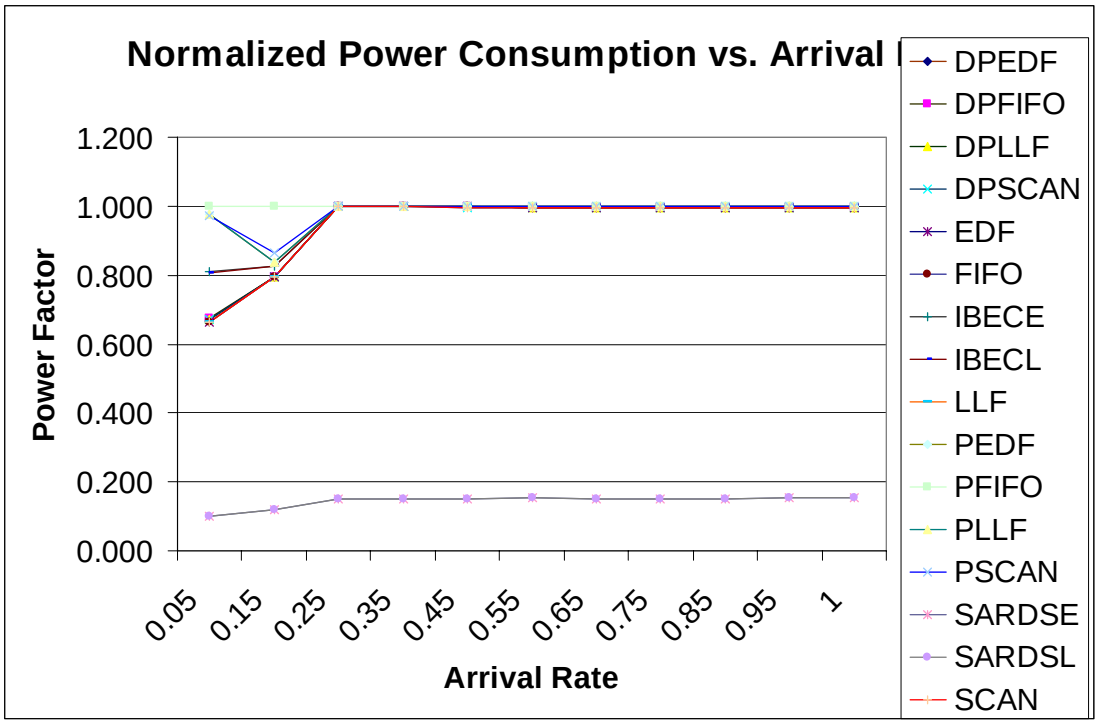

Fig 8. Power consumption for different arrival rates, chart and table

	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95	1
DPEDF	0.67	0.80	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99
DPFIFO	0.68	0.80	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99
DPLLF	0.67	0.80	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99
DPSCAN	0.67	0.80	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99
EDF	0.66	0.80	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99
FIFO	0.66	0.80	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99
IBECE	0.81	0.83	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
IBECL	0.81	0.83	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
LLF	0.66	0.80	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99
PEDF	0.97	0.84	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PFIFO	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PLLF	0.97	0.84	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PSCAN	0.97	0.87	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SARDSE	0.10	0.12	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
SARDSL	0.10	0.12	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
SCAN	0.66	0.80	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99

Here we see that in terms of guarantee ratio, all of the real-time algorithms performed essentially identically as denoted by the fact that their graphs all coincide. We also see that algorithms built around SCAN scheduling performed poorly relative to most of the other algorithms. This makes sense because SCAN is not a real-time algorithm, and it does not take deadlines into account when it reorders requests.

A review of the other metrics collected indicates that SCAN does a very good job of reducing the total amount of time spent in serving a set of disk requests, but at the same time SCAN causes a substantial increase in the average waiting time experienced by the average disk request, which can be extremely detrimental in a real-time system.

Curiously, we also see that algorithms based around FIFO scheduling perform roughly on par with real-time scheduling algorithms, even though as with SCAN, FIFO is not a real-time scheduling algorithm. We can understand why this occurs by considering the nature of our request stream. The statistical trend exhibited by our synthetically generated request-streams is that requests arriving at a later point in time will have later deadlines than those requests that arrived earlier (so if request 'y' arrives after request 'x', it is likely that the deadline of 'y', in absolute terms, will fall after the deadline of 'x'). Note that the synthetic workloads do contain situations in which a later request will have an earlier deadline than a prior request, but the general trend is that requests that arrive later have later deadlines. What this essentially means is that a FIFO scheduling policy will roughly approximate an EDF scheduling policy given a long enough timeline and a large enough number of requests, and this explains the seemingly out of place performance characteristics exhibited by FIFO-based scheduling and power management policies.

Of potentially greater import is that we see that algorithms using IBEC power management perform identically to other real-time scheduling algorithms, implying that the IBEC power management policy does indeed not destroy the property of optimality enjoyed by the underlying scheduling algorithm(s). We also notice here a trend that will persist through virtually every experiment, namely that SARDS-based algorithms appear to offer very poor performance, failing more deadlines or dropping

more requests than even the non-real-time SCAN algorithm. If we consider what is happening internally, we can see why this occurs when using the SARDS algorithm as originally specified. Essentially what occurs is that when a request arrives, SARDS evaluates the minimum disk speed necessary to service that request, and then sets the disk to spin at that speed. It does not evaluate subsequent requests when it does this to ensure that their deadlines will still be realizable after the current request is serviced at the reduced speed. So, what occurs, especially in workloads in which requests arrive more frequently, is that SARDS will set a disk speed that allows the first request to be serviced, but this speed will be slow enough that it causes some number of subsequent requests to become impossible to service. These unserviceable requests will be dropped, until we reach a request with a serviceable deadline, at which point the entire process repeats anew. This obviously results in a very large number of dropped requests, especially for more densely populated workloads, and it explains why SARDS-based solutions offer such poor performance, especially when we consider workloads in which it is not possible to service every request even if the disk operates at 100% capacity the entire time, as the SARDS approach of slowing down the disk just serves to exacerbate this condition.

When we look at power consumption, we don't see too much worth noting, as the workload is such that it requires that the disk be active essentially full time, thus making it nearly impossible to conserve any power. We do see something interesting in the first two datapoints however, in that it is actually the non-power aware algorithms that show the least total power consumption. This is an excellent example of how the overhead of transitioning from the sleep state to the active state can be quite high, as the reason this interesting anomaly occurs is that the nature of the work-stream is such that it just doesn't allow the disk to be left in the "sleep" state long enough to make up for the overhead of transitioning back to the "active" state.

It is also worth noting that at these two datapoints, the IBEC based algorithms show better performance than the naive power management algorithms by almost 20%, which indicates that it is at least improving the power management situation somewhat. SARDS based solutions show the lowest total power consumption (at the cost of many more failed deadlines), which is another fairly persistent trend,

indicating that the algorithms is keeping the drive in a lower power-state essentially the entire time, which is consistent with our observation as to why the guarantee ratio for SARDS is so low in this case.

6.2 Deadlines

Next, let us consider the experiment which varied the average request deadline, which generated the following charts:

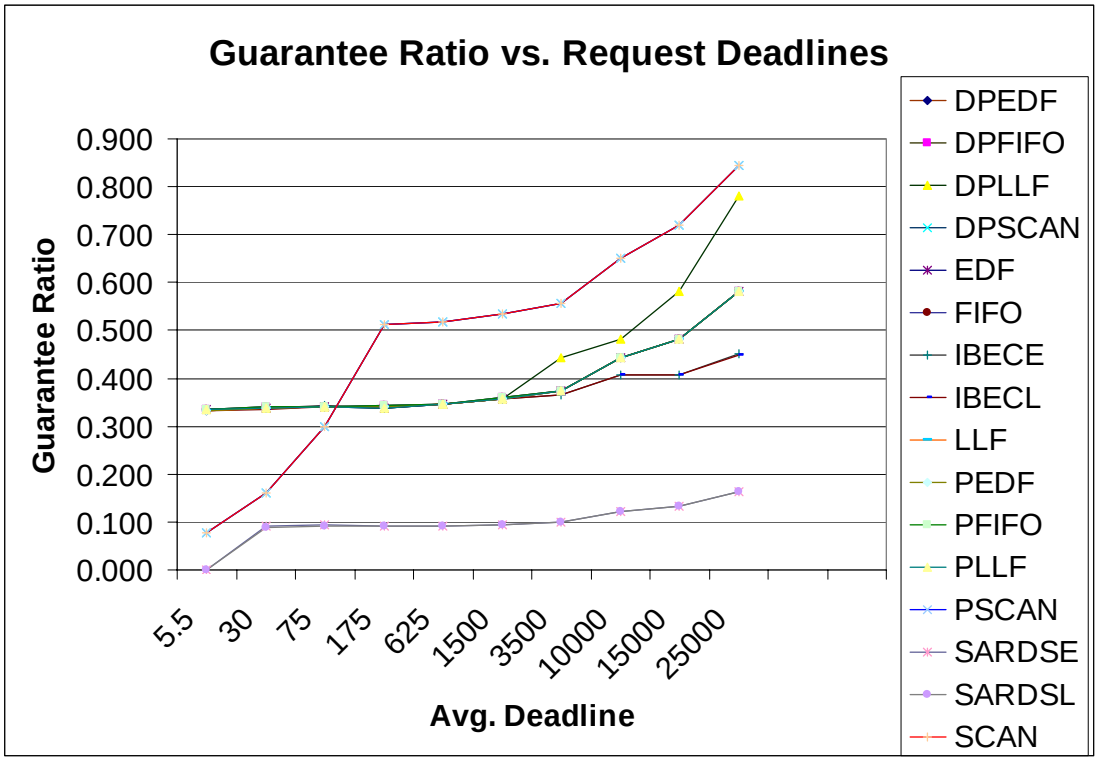

Fig 9. Guarantee ratio for different average request deadlines, chart and table

	5.5	30	75	175	625	1500	3500	10000	15000	25000
DPEDF	0.33	0.34	0.34	0.34	0.35	0.36	0.37	0.44	0.48	0.58
DPFIFO	0.34	0.34	0.34	0.34	0.35	0.36	0.37	0.44	0.48	0.58
DPLLF	0.33	0.34	0.34	0.34	0.35	0.36	0.44	0.48	0.58	0.78
DPSCAN	0.08	0.16	0.30	0.51	0.52	0.54	0.56	0.65	0.72	0.84
EDF	0.33	0.34	0.34	0.34	0.35	0.36	0.37	0.44	0.48	0.58
FIFO	0.34	0.34	0.34	0.34	0.35	0.36	0.37	0.44	0.48	0.58
IBECE	0.33	0.34	0.34	0.34	0.35	0.36	0.37	0.41	0.41	0.45
IBECL	0.33	0.34	0.34	0.34	0.35	0.36	0.37	0.41	0.41	0.45
LLF	0.33	0.34	0.34	0.34	0.35	0.36	0.37	0.44	0.48	0.58
PEDF	0.33	0.34	0.34	0.34	0.35	0.36	0.37	0.44	0.48	0.58
PFIFO	0.34	0.34	0.34	0.34	0.35	0.36	0.37	0.44	0.48	0.58
PLLF	0.33	0.34	0.34	0.34	0.35	0.36	0.37	0.44	0.48	0.58
PSCAN	0.08	0.16	0.30	0.51	0.52	0.54	0.56	0.65	0.72	0.84
SARDSE	0.00	0.09	0.09	0.09	0.09	0.09	0.10	0.12	0.13	0.16
SARDSL	0.00	0.09	0.09	0.09	0.09	0.09	0.10	0.12	0.13	0.16
SCAN	0.08	0.16	0.30	0.51	0.52	0.54	0.56	0.65	0.72	0.84

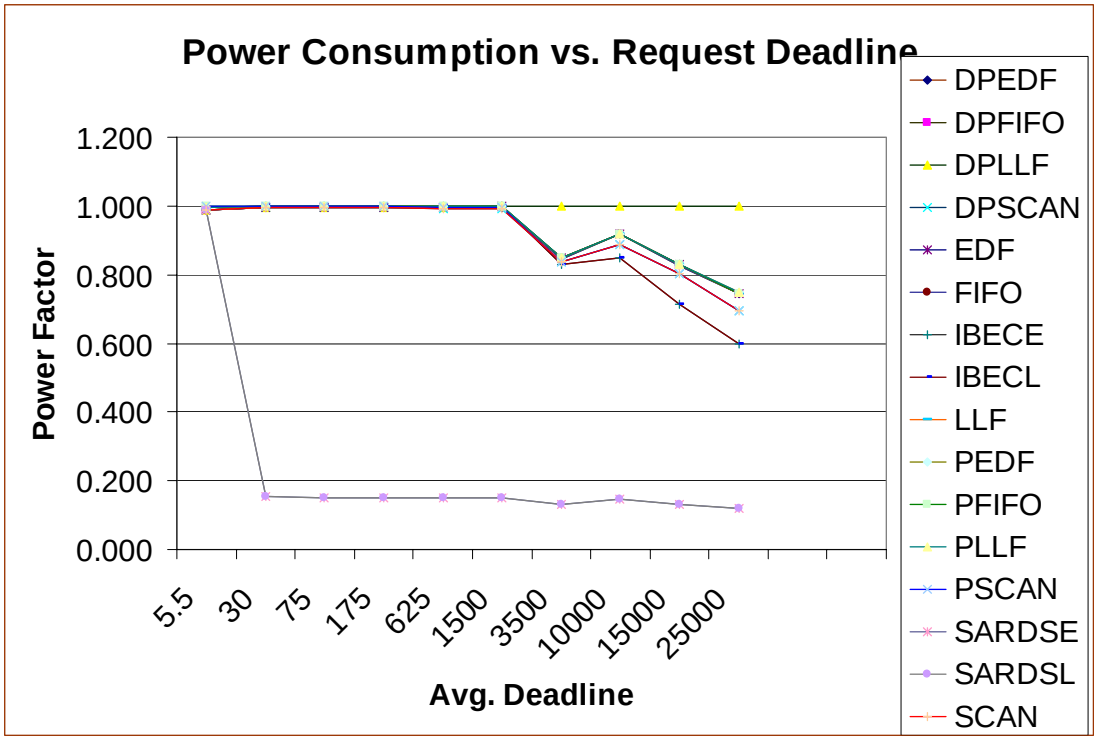

Fig 10. Power consumption for different average request deadlines, chart and table

	5.5	30	75	175	625	1500	3500	10000	15000	25000
DPEDF	0.99	1.00	1.00	1.00	1.00	1.00	0.85	0.92	0.83	0.75
DPFIFO	0.99	1.00	1.00	1.00	1.00	1.00	0.85	0.92	0.83	0.74
DPLLF	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DPSCAN	0.99	1.00	1.00	1.00	0.99	0.99	0.84	0.89	0.80	0.69
EDF	0.99	1.00	1.00	1.00	1.00	1.00	0.85	0.92	0.83	0.75
FIFO	0.99	1.00	1.00	1.00	1.00	1.00	0.85	0.92	0.83	0.74
IBECE	0.99	1.00	1.00	1.00	1.00	1.00	0.83	0.85	0.72	0.60
IBECL	0.99	1.00	1.00	1.00	1.00	1.00	0.83	0.85	0.72	0.60
LLF	0.99	1.00	1.00	1.00	1.00	1.00	0.85	0.92	0.83	0.75
PEDF	0.99	1.00	1.00	1.00	1.00	1.00	0.85	0.92	0.83	0.75
PFIFO	1.00	1.00	1.00	1.00	1.00	1.00	0.85	0.92	0.83	0.75
PLLF	0.99	1.00	1.00	1.00	1.00	1.00	0.85	0.92	0.83	0.75
PSCAN	1.00	1.00	1.00	1.00	1.00	1.00	0.84	0.89	0.80	0.69
SARDSE	0.99	0.15	0.15	0.15	0.15	0.15	0.13	0.14	0.13	0.12
SARDSL	0.99	0.15	0.15	0.15	0.15	0.15	0.13	0.14	0.13	0.12
SCAN	0.99	1.00	1.00	1.00	0.99	0.99	0.84	0.89	0.80	0.69

Here we see a few interesting things. The first is that in terms of guarantee ratio all the real-time algorithms start out performing virtually identically and that the IBEC based solutions match their performance for the first several testcases. Curiously however, IBEC performance drops off abruptly after a certain point. This is actually an artifact of our specific implementation of the IBEC algorithm, in which in order to keep simulation times manageable and to more closely approximate a real system in which there is not an infinite amount of time to spend re-evaluating the schedulability of a queue, we limited the maximum number of requests that IBEC would scan to ensure that it was not causing any deadlines to fail by delaying the execution of a request to 1,000. This means that if the queue of waiting requests grows larger than 1,000, our implementation of the IBEC algorithm will no longer guarantee that the schedulability of requests after the 1,000th will be preserved. Given our workload parameters, we know that on average, a new request should be arriving every 2 ms on average (because our default timestep is 1 ms, and our arrival rate is 0.5).

We also know that the average request deadline roughly approximates the maximum amount of time that IBEC will cache incoming requests in its queue as it attempts to allow the disk to sleep for as long as possible before waking it. Given this, we see that there should not be more than 1,000 entries in the waiting queue until the average deadline exceeds roughly 2,000 ms. Our data supports this observation, as for all data points with an average request deadline of less than 2,000 ms we see that IBEC based solutions perform exactly as well as other real-time algorithms,

indicating that the property of optimality is being properly preserved, while for all data points with an average request deadline of greater than 2,000 ms we see that the guarantee ratio for IBEC begins to fall off rapidly, indicating that the queue has grown to the length where our implementation of the IBEC algorithm no longer guarantees that optimality will be preserved.

SCAN also turns in an interesting performance for this set of test cases, as it initially starts out with a guarantee ratio that is much worse than the other algorithms, but it then comes back to offer better performance than any other algorithm. This is because as the deadlines becomes more and more lax, the added waiting time that requests experience under the SCAN algorithm is ultimately offset by the added performance that SCAN offers, allowing more requests to be serviced in the same amount of time, without keeping them waiting so long as to render their deadlines unserviceable.

As before, SARDS offers absolutely abysmal performance as the result of the exact same reasons discussed above. When we look at power consumption for this set of experiments, we see that initially every algorithm performs the same, as we again have a situation in which the hard-disk must be active essentially the entire time. As deadlines become more lax however, it does become possible to conserve some energy, and we see that IBEC based policies come in slightly ahead of the other algorithms, with the benefit increasing as deadlines continue to become more lax. SCAN again also performs well in this area, again thanks to its ability to reduce the total amount of time necessary to service a given set of requests, and thus in turn reducing the total amount of time that the disk must be active for, although this time SCAN is outperformed by IBEC in terms of power consumption, and it only just barely comes out ahead of the naive power management policies. SARDS continues its trend of showing very low power consumption, indicating that it is at least getting some benefit out of all the deadlines that it is allowing to fail.

6.3 Distribution Pattern

Now let us examine the impact that different request distribution patterns have on our algorithms:

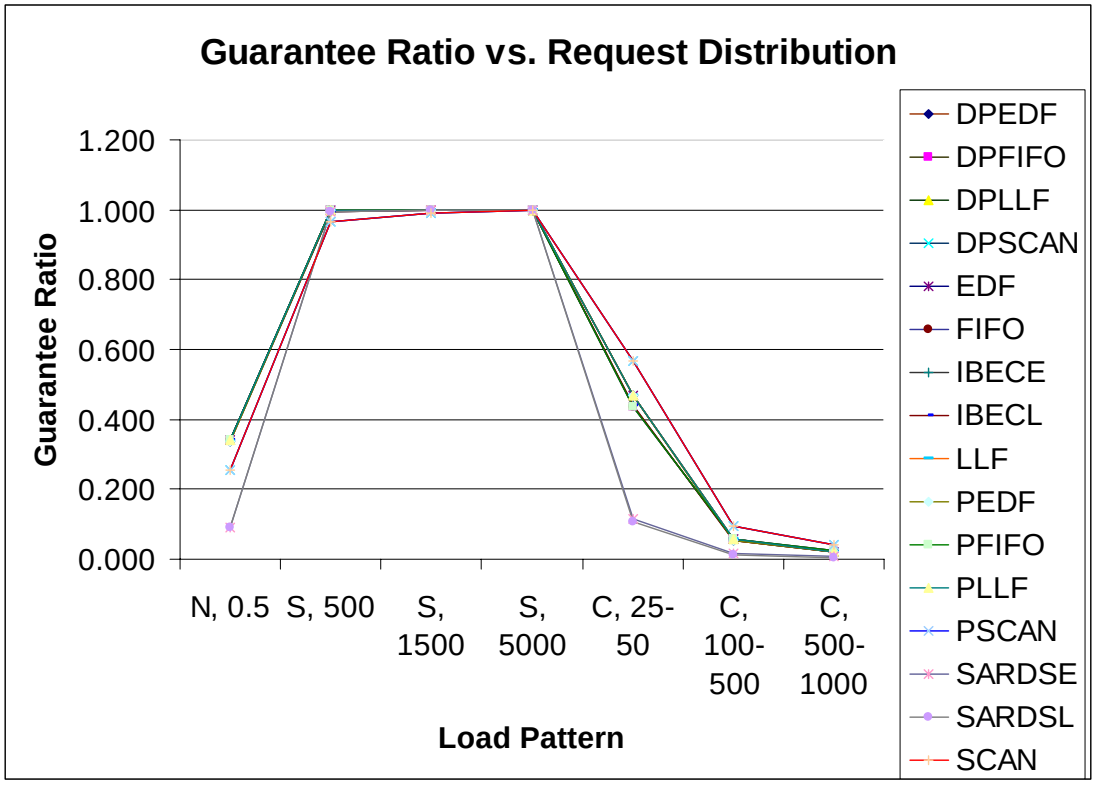

Fig 11. Guarantee ratio for different workload distributions, chart and table

	N, 0.5	S, 500	S, 1500	S, 5000	C, 25-50	C, 100-500	C, 500-1000
DPEDF	0.34	1.00	1.00	1.00	0.47	0.06	0.02
DPFIFO	0.34	1.00	1.00	1.00	0.43	0.06	0.02
DPLLF	0.34	1.00	1.00	1.00	0.47	0.06	0.02
DPSCAN	0.26	0.97	0.99	1.00	0.57	0.09	0.04
EDF	0.34	1.00	1.00	1.00	0.47	0.06	0.02
FIFO	0.34	1.00	1.00	1.00	0.43	0.06	0.02
IBECE	0.34	0.99	1.00	1.00	0.44	0.05	0.02
IBECL	0.34	0.99	1.00	1.00	0.44	0.05	0.02
LLF	0.34	1.00	1.00	1.00	0.47	0.06	0.02
PEDF	0.34	1.00	1.00	1.00	0.47	0.06	0.02
PFIFO	0.34	1.00	1.00	1.00	0.43	0.06	0.02
PLLF	0.34	1.00	1.00	1.00	0.47	0.06	0.02
PSCAN	0.26	0.97	0.99	1.00	0.57	0.09	0.04
SARDSE	0.09	1.00	1.00	1.00	0.12	0.02	0.01
SARDSL	0.09	1.00	1.00	1.00	0.11	0.01	0.01
SCAN	0.26	0.97	0.99	1.00	0.57	0.09	0.04

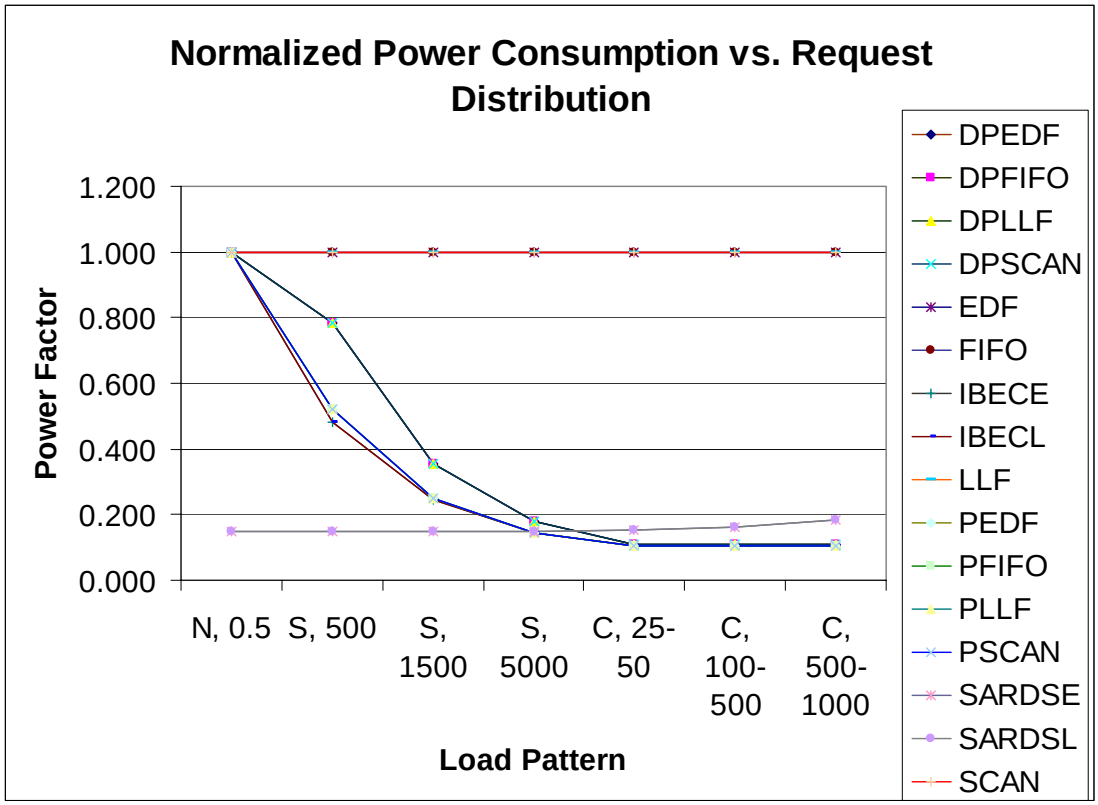

Fig 12. Power consumption for different workload distributions, chart and table

	N, 0.5	S, 500	S, 1500	S, 5000	C, 25-50	C, 100-500	C, 500-1000
DPEDF	1.00	0.78	0.36	0.18	0.11	0.11	0.11
DPFIFO	1.00	0.78	0.36	0.18	0.11	0.11	0.11
DPLLF	1.00	0.78	0.36	0.18	0.11	0.11	0.11
DPSCAN	1.00	0.78	0.36	0.18	0.11	0.11	0.11
EDF	1.00	1.00	1.00	1.00	1.00	1.00	1.00
FIFO	1.00	1.00	1.00	1.00	1.00	1.00	1.00
IBECE	1.00	0.48	0.24	0.15	0.11	0.11	0.11
IBECL	1.00	0.48	0.24	0.15	0.11	0.11	0.11
LLF	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PEDF	1.00	0.52	0.25	0.15	0.11	0.11	0.11
PFIFO	1.00	0.52	0.25	0.15	0.11	0.11	0.11
PLLF	1.00	0.52	0.25	0.15	0.11	0.11	0.11
PSCAN	1.00	0.52	0.25	0.15	0.11	0.11	0.11
SARDSE	0.15	0.15	0.15	0.15	0.15	0.16	0.18
SARDSL	0.15	0.15	0.15	0.15	0.15	0.16	0.18
SCAN	1.00	1.00	1.00	1.00	1.00	1.00	1.00

First, to explain the notation used along the X-axis of these two charts, the letter denotes the distribution pattern being used ("N" means normal, "S" means sparse, "C" means clustered), while the numbers denote additional parameters that were specified that are specific to the distribution model being used. For example, "N, 0.5" indicates normal distribution with an arrival rate of 0.5, "S, 500" indicates sparse distribution with a sparse idle threshold of 500 ms, and "C, 25-50" indicates a clustered distribution mode with between 25 and 50 requests occurring per cluster. Note that as all other experiments use a normal distribution pattern, we only include the single data point under normal distribution for this comparison. Here we see something interesting, in that the sparse load patters present the first fully schedulable workloads that we will discuss. In these cases, we see that all of the real-time algorithms deliver a 100% guarantee ratio, as do our IBEC based solutions. The sparse distribution also allows the SARDS based solutions to perform adequately for the first time that we have seen so far, as SARDS also yields a 100% guarantee ratio in these tests. The reason for this is that the average request deadline is on average much shorter than the sparse idle threshold, meaning that even though SARDS will aggressively slow down the disk when servicing each request, as long as it is still meeting the deadline of the current request, it will still finish servicing the request before the next request arrives, and it will thus no longer invalidate the deadlines of future requests when it slows the disk to service the current request.

We also see that in the case of a statistically normal distribution, the real-time algorithms and IBEC offer the best performance, and all perform identically, they are followed in performance by SCAN, and SARDS brings up the rear yet again in this test. When we use a clustered distribution pattern, we see an interesting occurrence in that although the performance of the real-time algorithms and IBEC remain the same relative to one another, SCAN now performs better than any of the real-time algorithms. This is again due to the ability of the SCAN algorithm to service a greater number of requests in the same amount of time as it takes the real-time algorithms to service a smaller number of requests, and also the fact that requests making up a given cluster will have deadlines that all fall at approximately the same time (but not exactly the same time), meaning that in this instance, the most

successful algorithm will be the one that can service the largest number of requests before the cluster's approximate deadline expires, and SCAN is the algorithm that is most capable of doing this.

When we look at the power consumption data, we see that for the normal distribution all algorithms essentially perform the same, with the exception of SARDS. Under the sparse distribution however, we see that SARDS actually offers the best performance here, and in this instance without failing any deadlines unnecessarily, indicating that if SARDS were modified to ensure that it will properly preserve optimality, it may become a very competitive power management policy. IBEC based algorithms performed second best, coming in just slightly ahead of our naive power management techniques, and algorithms with no power management whatsoever performed substantially worse than both IBEC and naive power management techniques. When we look at the power consumption data for our clustered workloads, we see that IBEC and the naive power management techniques perform comparably, and that they both perform better than SARDS. This is due to the fact that SARDS does not include a policy for sleeping the disk completely, unlike both the naive power management schemes and IBEC, which means that SARDS instead simply runs the disk in a minimal power state no matter how long it is left to idle between clusters of requests, and this is what allows the other methods to surpass its performance. We also note that under the clustered load pattern, algorithms with no power management policy whatsoever demonstrated the worst power consumption.

6.4 Request Size

Finally, let us take a look at how request size affects our performance, before we delve into our real-world traces:

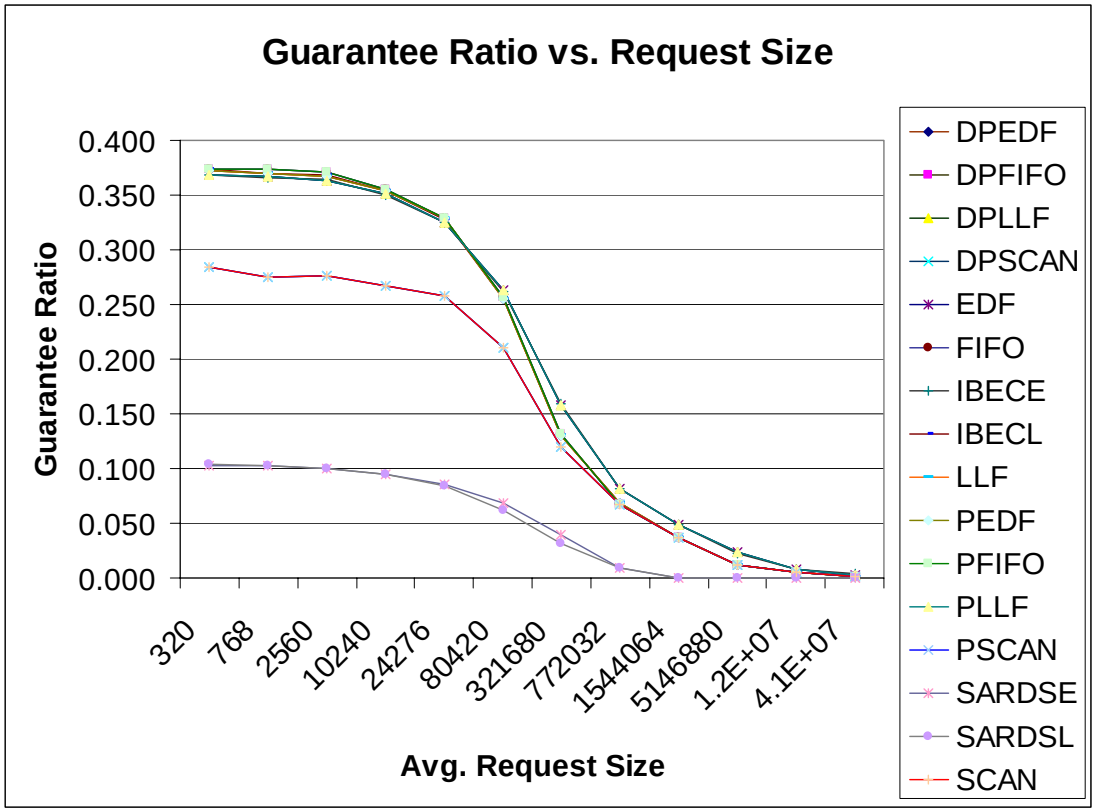

Fig 13. Guarantee ratio for different average request sizes, chart and table

	320	768	2560	10240	24276	80420	321680	772032	1544064	5146880	12352512	41175040
DPEDF	0.37	0.37	0.37	0.35	0.33	0.26	0.13	0.07	0.04	0.01	0.01	0.00
DPFIFO	0.37	0.37	0.37	0.36	0.33	0.26	0.13	0.07	0.04	0.01	0.01	0.00
DPLLF	0.37	0.37	0.36	0.35	0.33	0.26	0.16	0.08	0.05	0.02	0.01	0.00
DPSCAN	0.28	0.28	0.28	0.27	0.26	0.21	0.12	0.07	0.04	0.01	0.01	0.00
EDF	0.37	0.37	0.36	0.35	0.33	0.26	0.16	0.08	0.05	0.02	0.01	0.00
FIFO	0.37	0.37	0.37	0.36	0.33	0.26	0.13	0.07	0.04	0.01	0.01	0.00
IBECE	0.37	0.37	0.36	0.35	0.32	0.26	0.16	0.08	0.05	0.02	0.01	0.00
IBECL	0.37	0.37	0.37	0.35	0.33	0.26	0.13	0.07	0.04	0.01	0.01	0.00
LLF	0.37	0.37	0.37	0.35	0.33	0.26	0.13	0.07	0.04	0.01	0.01	0.00
PEDF	0.37	0.37	0.37	0.35	0.33	0.26	0.13	0.07	0.04	0.01	0.01	0.00
PFIFO	0.37	0.37	0.37	0.36	0.33	0.26	0.13	0.07	0.04	0.01	0.01	0.00
PLLF	0.37	0.37	0.36	0.35	0.33	0.26	0.16	0.08	0.05	0.02	0.01	0.00
PSCAN	0.28	0.28	0.28	0.27	0.26	0.21	0.12	0.07	0.04	0.01	0.01	0.00
SARDSE	0.10	0.10	0.10	0.09	0.09	0.07	0.04	0.01	0.00	0.00	0.00	0.00
SARDSL	0.10	0.10	0.10	0.09	0.08	0.06	0.03	0.01	0.00	0.00	0.00	0.00
SCAN	0.28	0.28	0.28	0.27	0.26	0.21	0.12	0.07	0.04	0.01	0.01	0.00

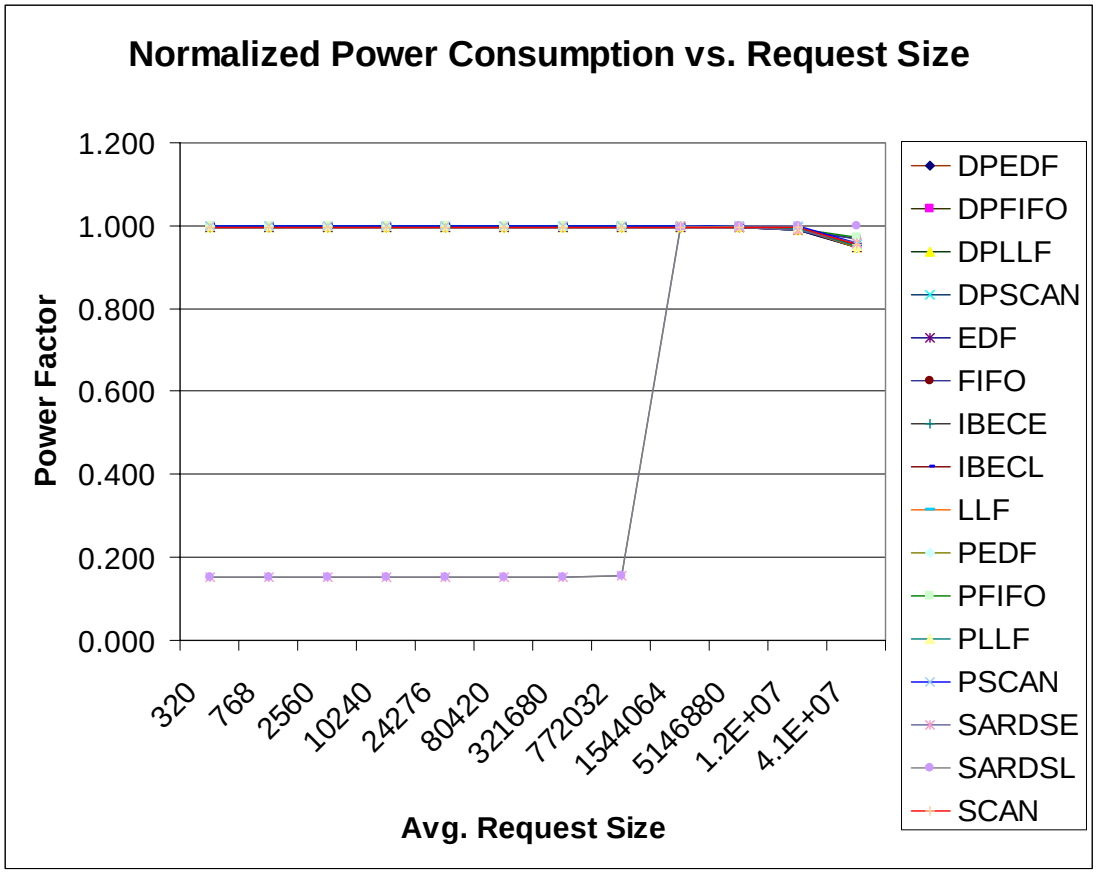

Fig 14. Power consumption for different average request sizes, chart and table

	320	768	2560	10240	24276	80420	321680	772032	1544064	5146880	12352512	41175040
DPEDF	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.95
DPFIFO	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.97
DPLLF	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.95
DPSCAN	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96
EDF	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.95
FIFO	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.97
IBECE	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.95
IBECL	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.95
LLF	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.95
PEDF	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.95
PFIFO	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.97
PLLF	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.95
PSCAN	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96
SARDSE	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	1.00	1.00	0.99	0.96
SARDSL	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	1.00	1.00	1.00	1.00
SCAN	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96

Here we again see that in terms of guarantee ratio, the real-time algorithms and the IBEC based solutions perform in an identical fashion. SARDS returns to its pattern of performing worse than any other algorithm, and SCAN initially performs worse than the real-time algorithms, and then on par with them as the average request size becomes larger. This is because the request sizes are allowed to grow to the

point where they are so large that the scheduling algorithm used essentially becomes irrelevant, as performance is simply bounded by how many requests can be completed before all deadlines have expired, which is the same regardless of which algorithm is used thanks to the admissions controller.

In terms of power consumption, things are essentially flat across the board, with the exception of our SARDS based algorithms, which offer very low power consumption across most test cases but which are not worth considering in this instance due to the relatively poor guarantee ratio. The remaining algorithms perform essentially the same in terms of power consumption for this test because the nature of the data is such that the disk is required to be active essentially the entire time. It is also worth noting that once the request size becomes so large that it is not possible to meet a deadline unless the disk is allowed to operate at full speed, SARDS exhibits the exact same power consumption characteristics as the other algorithms, as it can no longer get away with running the disk at a slower speed.

6.5 Traces

Now, let us consider our real-world traces:

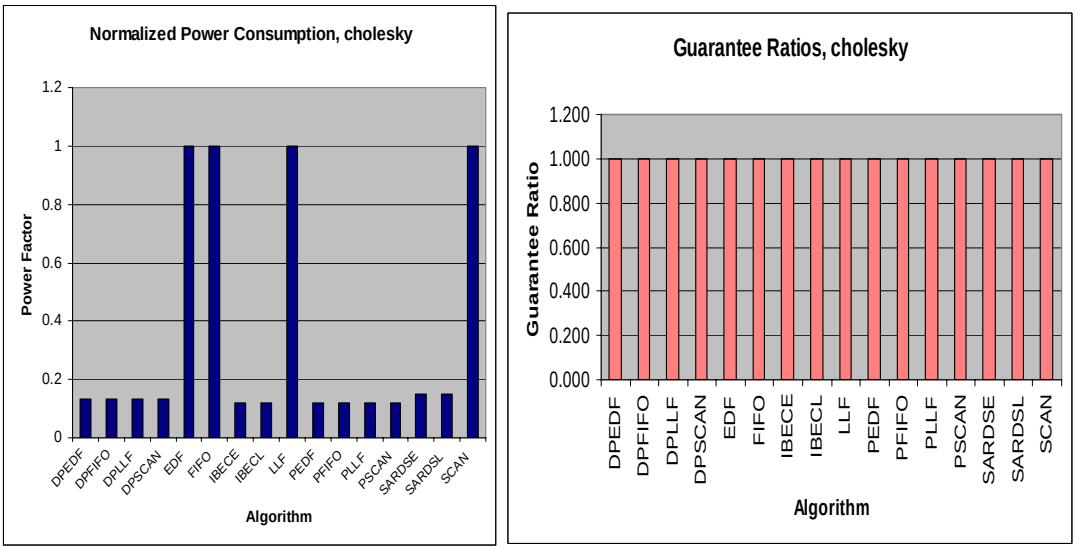

Fig 15. Power consumption and guarantee ratios for cholesky, a factorization application

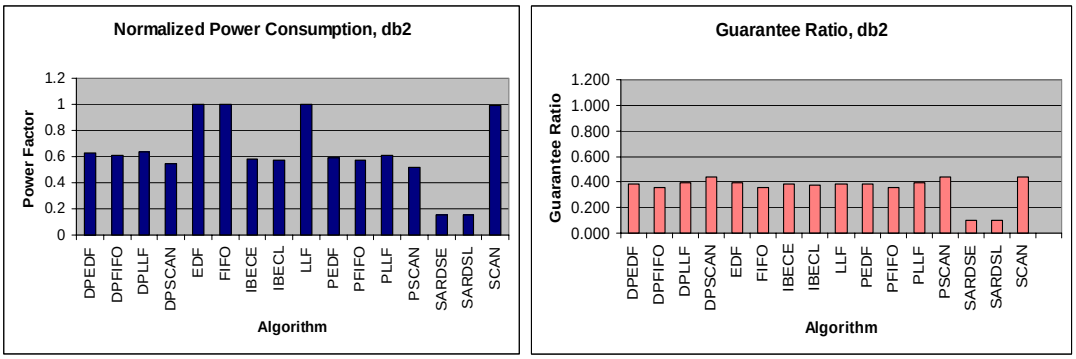

Fig 16. Power consumption and guarantee ratios for db2, a RDBMS application

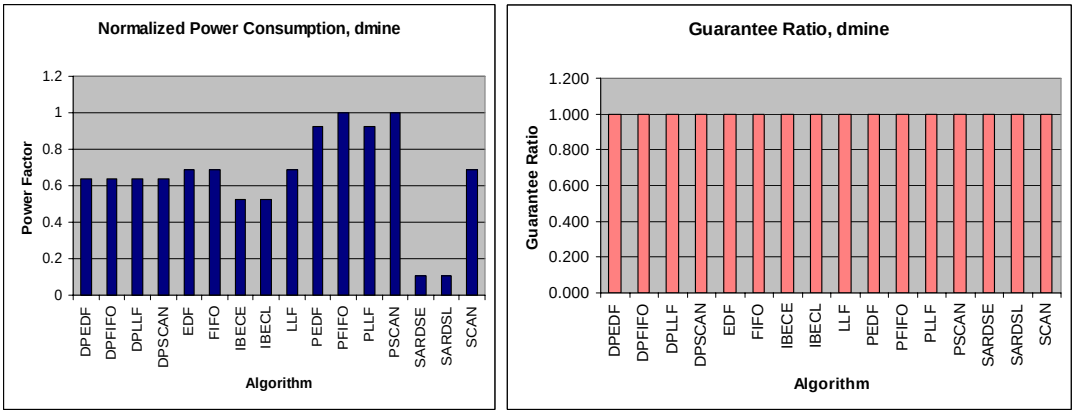

Fig 17. Power consumption and guarantee ratios for dmine, a data-mining application

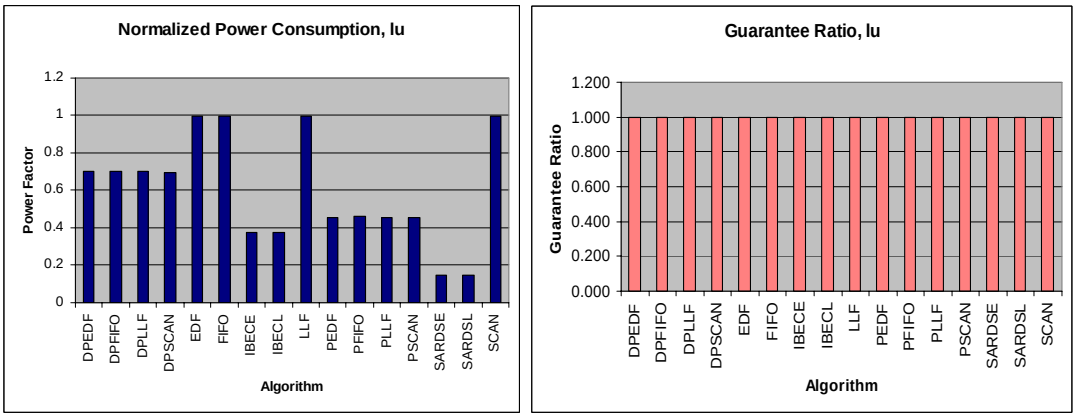

Fig 18. Power consumption and guarantee ratios for lu, another factorization application

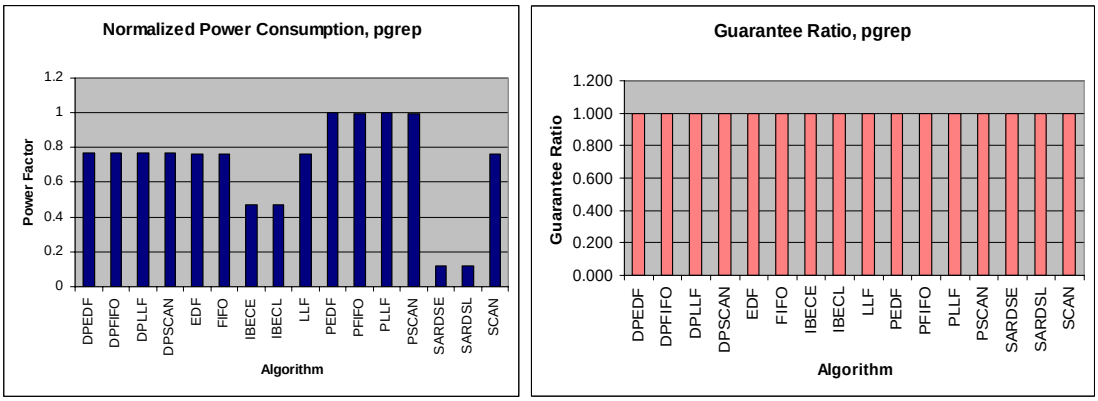

Fig 19. Power consumption and guarantee ratios for pgrep, an application for searching active processes

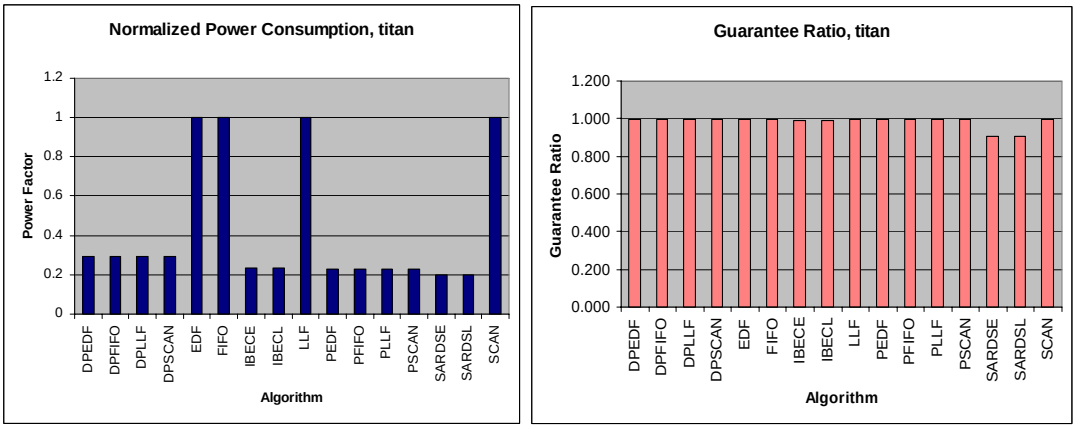

Fig 20. Power consumption and guarantee ratios for titan, a parallel computing application

Here we see some very promising results. For all 6 applications considered, IBEC is able to offer guarantee ratios that are on par with those provided by other real-time algorithms while at the same time consistently producing reduced power consumption over naive power management as well as over no power management. In some cases the improvement is a fairly modest one, although in others it is more substantial. The most important thing to note is that the power savings are consistent across every test conducted. We can therefore see that IBEC does indeed appear to be a valid mechanism of improving the power consumption characteristics of a system without compromising performance.

It is also interesting to note that when the workload is completely schedulable, algorithms based on the SARDS policy also deliver high guarantee ratios and reduced power consumption in most cases (namely, when working with pgrep, lu, dmine, and cholesky).

In each of these four cases, with the exception of cholesky, we see that SARDS consistently reduces power consumption by a substantial margin when compared with the next best algorithm. The fact that this is not the case when working with cholesky is likely due to the fact that as previously mentioned, SARDS does not have a policy that allows for it to sleep the disk during periods of extended idle activity, and is thus forced to run it at a minimal power level when idle, and as shown above, in certain situations this makes it possible for other power management policies to outperform SARDS. These results are promising not only because they show that IBEC can be reliably used to consistently conserve at least a modicum of energy, but also because they indicate that the SARDS algorithm as specified may be a very effective way to conserve energy if SARDS is deployed in a system which is always guaranteed to produce a schedulable stream of disk requests, and also because they indicate that if the SARDS algorithm were improved to ensure that it always preserves optimality and so that it has the ability to sleep the disk when it is idle it could potentially be a very effective and robust power management policy.

Although our synthetic tests consisting of workloads deliberately chosen such that they are not schedulable indicate that SARDS as specified has some serious issues that need addressing, our tests using traces from real-world applications indicate that the algorithm is by no means hopeless, and that there are situations where it can potentially perform exceptionally well.

7. Future Work

Our results indicate that although the SARDS algorithm as specified at the start of this research has some fairly serious issues, it still holds a lot of promise in terms of reducing the overall power consumption of the I/O subsystem in real-time systems. As such, the future work will be focused on refining and improving the SARDS algorithm to improve its power management characteristics as well as its ability to preserve the schedulability of the request queue that it is operating on. Our research indicates that in order for SARDS to be a viable algorithm in the real-world context, it is necessary first to:

1. Add a policy to the SARDS algorithm that allows it to examine requests in the queue to ensure that by slowing disk performance to service the current request it is not rendering the deadlines of subsequent requests unrealizable. Note that this is similar to the policy that already exists in IBEC by which it will scan its queue of waiting requests to verify that by delaying the waking of the hard-disk it is not causing future deadlines to fail, but substantially more complex. Instead of being able to simply maintain a sliding threshold of when it is necessary to wake the disk to ensure the completion of all deadlines in the queue like IBEC can, SARDS must be capable of dynamically determining what the optimal power state is to allow the servicing of all requests in the queue without expending more energy than is absolutely essential from a field of arbitrarily many potential power states, and it must be capable of performing this evaluation in real-time. Such a policy would make the actual use of SARDS in real systems a feasible possibility, and brings with it the potential to dramatically reduce the total amount of power consumed, as our results indicate.

2. Add a policy to the SARDS algorithm that specifies criteria under which it is permitted to sleep the hard-disk drive. This is not an essential addition in terms of getting SARDS to the point where it functions well enough to be

truly useful, but it is a relatively easy optimization that will improve the algorithm's overall performance and eliminate the current behavior of simply letting the disk idle in a minimal power state when there are no current requests to process which causes it to perform worse than other algorithms under certain circumstances. Our results indicate that even a naive power management method such as sleeping the hard-disk whenever it is idle for longer than some threshold of time is a reasonable policy for managing power consumption under most types of workloads (although it is important to not forget that this policy does introduce the possibility of actually degrading performance under certain conditions, and that it is sometimes outperformed by other naive algorithms as well), so it should be possible to improve the characteristics of SARDS by using it in conjunction with such a policy controlling when to sleep and wake the disk. This would yield an improved algorithm that could reduce power consumption when the system was active by running the disk in the minimal power-state necessary to guarantee that deadlines are met, while at the same time also reducing power consumption during periods of idle activity by allowing the algorithm to sleep the disk until there are more jobs to service. This has the added benefit of that it causes the SARDS algorithm to become compatible with non multi-speed disks, in that if it is used with such a disk, it will still be able to perform at least as well as the naive power management policy would.

If these improvements are implemented, it is believed on the basis of our current results that the improved SARDS algorithm would perform consistently better than either naive power management policies or IBEC under nearly any workload scenario, and that the margin of improvement could be a substantial one. Additional simulation and research should be conducted in such a case however, to ensure that the modified algorithm both lives up to its expected performance and also to prove that it will properly preserve the property of optimality possessed by the underlying scheduling algorithm used with it.

8. Conclusions

We have shown through extensive simulation experiments that it is possible to improve the power consumption characteristics of the disk subsystem in a real-time system consistently without compromising the system's ability to guarantee deadlines through application of the IBEC power management policy when used in conjunction with a real-time scheduling algorithm. We have also shown that although the SARDS algorithm as specified is inadequate in most situations, the underlying concept does offer a great deal of promise in terms of being able to reduce power consumption if the algorithm is revised to ensure that it preserves the schedulability of the requests that it is working with. It is feasible to apply SARDS to a real-time system with soft deadlines, or in any system in which a given stream of requests will always be schedulable, as we observe that in such instances SARDS does indeed achieve a guarantee ratio that is on par with other real-time algorithms, at a substantially reduced level of power consumption.

We propose future work in the form of refinements that can be made to SARDS both to address the issue regarding schedulability and to improve the performance characteristics of the algorithm in general by adding to it a policy that allows SARDS to sleep the disk completely when there are no current jobs available to process. By examining how these algorithms perform on traces from real-world applications, we observe that both SARDS and IBEC have the potential to reduce the power consumption of the system when compared to those techniques most commonly in use today.

It is important to not lose sight of the fact that to realize the added energy savings offered by SARDS, a multi-speed disk must be used. If such a disk is not available, then IBEC would be the algorithm which offers the best power consumption characteristics. Given our experimental results, we believe that IBEC is the power management algorithm of choice under nearly any workload conditions, as in essentially all cases we observed that IBEC preserves or comes very close to preserving optimality despite the fact that it is possible to envision cases where it will

not always do so. Further, IBEC offers consistently better power consumption characteristics relative to naive power management policies under a wide-range of workloads. If a multi-speed disk is available, and an improved version of the SARDS algorithm implemented, then SARDS would likely replace IBEC as the official recommendation, although additional research regarding the modified SARDS algorithm would be necessary in order to support this recommendation.

9. Acknowledgements

I would like to thank my advisor, Professor Xiao Qin, for his assistance with this research, for helping proofread my thesis, for his encouragement along the way, for prodding me to make sure I didn't fall too far behind or forget to do anything important, and for providing the initial specification of the IBEC and SARDS algorithms. I would also like to thank Dr. Subhasish Mazumdar and Dr. Lorie Liebrock for reviewing the original versions of my thesis. The discussions in addition to their valuable comments have tremendously helped in improving the quality of the thesis. I would also like to thank Dan Greening for being understanding of the fact that college can be very time-consuming at times, and for being flexible enough to continue to work with me regardless. I also thank Ron Lussier, and all of Bigtribe Corporation, for providing me with the software development, implementation, and testing skills necessary to implement, test, and debug a fairly complicated simulation platform on an extremely limited timeframe. Were it not for the knowledge that I gained working with them, this research would not have been feasible except for on a much longer time-frame. Finally, I thank Microsoft, if only because they get picked on a lot here, and to their credit Windows did not crash a single time during the simulation runs, or while this document was being written, so they must be doing something right.

10. References

[1] A. Huffman and J.Clark, "Native Command Queueing," *Intel Corporation and Seagate Technology*, July 2003.

[2] A. Huffman, "Comparing Serial ATA Native Command Queuing (NCQ) and ATA Tagged Command Queuing (TCQ)," *Intel Corporation*, October 2003.

[3] A. Papathanasiou and M. Scott, "Increasing Disk Burstiness for Energy Efficiency," *University of Rochester*, 2002, Technical Report 792.

[4] A. Papathanasiou and M. Scott, "Power-efficient Server-class Performance from Arrays of Laptop Disks," *University of Rochester*, 2003, Technical Report 837.

[5] Advanced Micro Devices, "Cool 'n' Quiet™ Technology Installation Guide for AMD Athlon™ 64 Processor Based Systems," *AMD*, June 2004.

[6] D. Helmbold, D.Long, T. Sconyers and B. Sherrod, "Adaptive disk spin-down for mobile computers," *Mobile Networks and Applications 5*, 2000, pp. 285-297.

[7] E. Carrera, E. Pinheiro and R. Bianchini, "Conserving Disk Energy in Network Servers," *Proceedings of the 17th annual international conference on Supercomputing*, 2003, Session: Power, pp. 86-97.

[8] I. Hong and M. Potkonjak, "Power Optimization in Disk-Based Real-Time Application Specific Systems," *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, 1996, pp 634 - 637.

[9] Intel Corporation, "Enhanced Intel Speedstep® Technology," *Intel Technology Journal*, May 2003, Volume 7, Issue 2, Section 10.

[10] J. Liu, "Real-Time Systems," *Prentice Hall*, 2000.

[11] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthyy and R. Wang, "Modeling Hard-Disk Power Consumption, " *Proceedings of the Second USENIX Conference on File and Storage Technologies*, 2003.

[12] L. Benini, A. Bogliolo and G. D. Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, June 2000, Vol. 8, No. 3.

[13] P. Greenawalt, “Modeling Power Management for Hard Disks,” *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, September 1993, pp. 62 – 66.

[14] P. Pillai and K. Shin, “RealTime Dynamic Voltage Scaling for LowPower Embedded Operating Systems,” *ACM SOSP*, 2001, pp. 86-102.

[15] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir and H. Franke, “DRPM: Dynamic Speed Control for Power Management in Server Class Disks,” *Proceedings of the International Symposium on Computer Architecture*, 2003.

[16] T. Bisson and S. Brandt, "Adaptive Disk Spin-Down Algorithms in Practice," *3rd USENIX Conference on File and Storage Technologies*, 2004.

[17] T. Xie and X. Qin, “Scheduling Security-Critical Real-Time Applications on Clusters,” *Proceedings of the 34th International Conference on Parallel Processing*, June 2005, pp. 5-12.

[18] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadron, “Power-aware QoS Management in Web Servers,” *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, 2003.

[19] V. Swaminathan and K. Chakrabarty, “Energy-Conscious, Deterministic I/O Device Scheduling in Hard Real-Time Systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, July 2003, Vol. 22, No. 7.

[20] Y. Lu, L. Benini, and G. Micheli, “Low-Power Task Scheduling for Multiple Devices,” *International Workshop on Hardware/Software Codesign*, 2000, pp.39-43.