*Appears in Proceedings of SIGMETRICS 2002 Conference*

# Analysis of Methods for Scheduling Low Priority Disk Drive Tasks

Eitan Bachmat
Ben-Gurion University
Beer Sheva, Israel
ebachmat@cs.bgu.ac.il

Jiri Schindler
Carnegie Mellon University
Pittsburgh, PA, USA
jiri@andrew.cmu.edu

## ABSTRACT

This paper analyzes various algorithms for scheduling low priority disk drive tasks. The derived closed form solution is applicable to class of greedy algorithms that include a variety of background disk scanning applications. By paying close attention to many characteristics of modern disk drives, the analytical solutions achieve very high accuracy—the difference between the predicted response times and the measurements on two different disks is only 3% for all but one examined workload. This paper also proves a theorem which shows that background tasks implemented by greedy algorithms can be accomplished with very little seek penalty. Using greedy algorithm gives a 10% shorter response time for the foreground application requests and up to a 20% decrease in total background task run time compared to results from previously published techniques.

## 1. INTRODUCTION

Many storage systems perform tasks that fall into two broad categories— high priority foreground tasks and low priority background tasks. Foreground tasks are generated from a primary application of the system such as online transaction processing (OLTP). Background tasks result from upkeep and maintenance activity and do not contribute to the forward progress of the primary application. When requests from these two types of tasks are scheduled concurrently (either when the system is always busy or when the background task must maintain a certain rate of forward progress) the background requests have negative impact on the performance.

There are numerous examples of solutions that analytically express the impact of background tasks on foreground activity [4, 8, 9, 22]. All these solutions focus on tasks that fall into a class of algorithms that proceed sequentially from the beginning of the disk. We call these algorithms *ordered*. In this paper, we analyze a class of *greedy* algorithms that minimize the negative impact on response times of foreground requests and yield shorter completion times for the background activity. The analysis applies to disk scanning applications running as background tasks such as scrubbing, data backup, and rebuild, or to tasks that access only a portion of a disk such as disk reorganization and virus detection.

Reducing the completion time and minimizing the performance impact on foreground activity is particularly valuable for data rebuild in RAID configurations [4, 10]. The observed 19.2% faster rebuild time with our greedy algorithm shortens the window of opportunity for data loss due to another disk failure when the system is in degraded mode (i.e., rebuilding data after a disk failure). In addition, our greedy algorithm offers 10% shorter foreground request response times than the ordered algorithm, allowing the system to handle its load more gracefully when in degraded mode.

The analysis of methods for scheduling low priority disk drive tasks presented in this paper makes three contributions. First, it gives a closed form analytical solution to the class of greedy algorithms. The main theoretical result of this analysis is Theorem 1 which states that background tasks scheduled by greedy algorithms can be serviced with very little seek penalty. Second, it analytically and experimentally evaluates previously proposed algorithms for data rebuilds in RAID systems and applies them to other background tasks. And third, by paying close attention to many disk characteristics including zero-latency access, head switch times, and detailed seek profile and layout information, we observe a difference of at most 3% between the model predictions and the measurements on a real disk for all but one of the studied workloads.

The remainder of the paper is organized into three parts. Sections 2 and 3 identify various background tasks in storage systems and summarize prior work in background task analysis. Sections 4 and 5 describe two background task algorithms and analyze their performance using a variety of models. Section 6 gives both analytical and empirical results for the two algorithms and compares them to the results of prior work.

## 2. BACKGROUND TASKS

Storage systems experience many different low priority background tasks that occur simultaneously with high priority foreground activity. These background tasks are a result of preventive, maintenance, and repair activities of the system, or simply tasks of a secondary application whose performance is not as critical.

In RAID disk arrays [10], these activities include data integrity checking, write-back, prefetching, and data rebuild after a disk failure. Data integrity checking, or RAID scrubbing, is a periodic activity that verifies the integrity of the data stored on the disk to identify problematic areas where unrecoverable errors might later occur. Write-back is a process of moving data from a disk array's cache to the disk when the cache fills up or the number of dirty blocks in the cache reaches a certain threshold. Reads have priority over writes since a host is waiting for the data. Therefore, writes can be considered a background task.

Tasks originating from a secondary application include activities such as data reorganization (to optimize access times), reclamation of fragmented space (e.g., log-structured file system cleaning [15]), and a variety of scanning applications such as data backup, virus detection, or data mining on a live OLTP system [14]. Lumb et al. [7] describe in more detail various classes of background applications.

When scheduling a background request we must determine the size of the request, its location, and timing with respect to the foreground requests. Larger request sizes cause longer service times with higher impact on the performance of the primary application. Thus, background requests should be generated such that they interfere with the foreground activity as little as possible.

First, setting the request size equal to the physical track size is a good compromise between the background task progress and the impact on foreground application [4, 8, 22]. It also achieves the highest disk efficiency [3, 18], especially when combined with the zero-latency access feature of modern disk drives, and simplifies our analysis.

Recall that zero-latency access allows sectors on a single track to be read out of logical block order. Thus, it takes at most one revolution to read an entire track because the disk head can start reading data as soon as it arrives at the destination track rather than waiting for the logical beginning of the track. In comparison, a disk without the zero-latency access feature will take, on average, one and a half revolutions to read a single track.

Second, in deciding when to schedule the next background request, one may schedule them at fixed time intervals or take into account the load of the system. In this paper, we consider the case of issuing the next background request as soon as the previous one finishes irrespective of the load.

A third choice concerns the priority given to the background task in the system's queue. A background request can have the same priority as foreground requests. This is known as the permanent customer model [2] (one that reenters the queue once it is serviced) and it has been analyzed extensively by others [4, 8, 22]. Another choice is to give the background requests lower priority and service them only when there is no other foreground request in the queue. This is known as the vacationing server model [20].

The fourth choice centers around the issue of what data to access next. In previous work on disk reconstruction in RAID systems, the choice was simply to read the next unreconstructed unit [8, 22]. This choice is inevitable in the context of RAID5, where a parity group is divided among $N$ disks with $N-1$ disks containing data and one disk containing the parity information. When reconstructing a failed disk, all $N-1$ disks must be accessed to reconstruct the $N$-th one. Since the disk heads of the $N-1$ independent disks are at different locations, each disk access will incur a seek regardless of the order in which the data is reconstructed.

In RAID1 (i.e., two mirrored disks) reconstruction the data is read from only one disk. Therefore, we can reconstruct data that are closest to the current head position. This minimizes (or completely eliminates) the seek penalty and thus lowers the access time for the background reconstruction request and lessens the impact on the foreground requests. Reading data opportunistically closest to the current head position benefits any background disk scanning application, not just RAID1 reconstruction.

# 3. RELATED WORK

Many researchers studied and analyzed data reconstruction algorithms for RAID configurations using queueing theory, but their analysis is only applicable to a specific disk configuration (i.e., RAID level).

Thomasian and Menon [22] provide extensive analysis of various reconstruction policies for RAID5 systems using the vacationing server model. All policies, which assume independent head movement for all disks in the same parity group, collect data from the same physical location on different disk drives. The seeks that bring the disk heads to the same position may vary greatly and the time spent seeking is essentially the maximum of several identically distributed random variables. They give the foreground requests higher priority and thus issue a background reconstruction request only when the RAID5 group of disks is idle.

Because of the need to bring all disk heads to the same position, their analysis uses the fork-join estimates [21] and considers ordered reconstruction, which proceeds sequentially, starting at the logical beginning of the disk, and maintains a pointer indicating the position of the last recovered block.

We present an analysis of the greedy reconstruction using the vacationing server model which is applicable to a large class of disk scanning background tasks including data reconstruction in RAID1. However, it does not apply to the RAID5 reconstruction because of the disk head independence assumption.

Merchant and Yu [8] devised a model for ordered data reconstruction for RAID1 using a variant of a permanent customer model. Their disk queue, serviced in FCFS order, always contains one background reconstruction task in addition to the foreground tasks, giving the foreground and background requests the same priority. We extend their analysis of the permanent customer model to greedy algorithms and compare it to the vacationing server model described by Thomasian and Menon [22] and applied to RAID1.

Lumb et al. experimentally studied the performance of background tasks issued to the disk drive [7]. Their freeblock scheduler issues only those background requests that can be serviced within a rotational latency between two foreground requests. Thus, the background requests have no impact on the performance of the foreground application. Unfortunately, the freeblock scheduler cannot serve as a general purpose background task scheduler since the request size is limited by the size of the opportunity window, resulting in long run times unacceptable for certain applications. Theorem 1 in Section 5.4 provides a partial analysis of such heavily constrained schedulers.

# 4. ALGORITHMS

We consider two classes of algorithms for issuing background tasks. The first class, which we call *ordered*, is derived from other models for data reconstruction in RAID1 and RAID5 disk arrays [8, 22]. The second class, which we call *greedy*, is a new alternative to the ordered class that offers better performance.

For the remainder of the paper, we will consider the task of reconstructing a disk after a failure in a mirrored system (RAID1) and thus refer to the algorithms as ordered and greedy reconstruction. However, our analysis applies without loss of generality to any scanning background task issuing requests to a single disk.

We choose a single track as a unit of reconstruction. While this may complicate the implementation as disks have different number of sectors per track for different zones, the time to read a track is always the same and thus useful in our analysis. We will address the issue of disk zones in Section 6.

## 4.1 Ordered algorithm

The ordered reconstruction algorithm reads data sequentially from a primary disk and keeps a pointer to the next unreconstructed track. A single background request is thus a read of a single track marked by the pointer:

1. Point to the first track on the disk.
2. When it is time to service a reconstruction request, read the unreconstructed track marked by the pointer.
3. Advance the pointer to the next track.
4. If the entire disk has not yet been read, return to 2.

## 4.2 Greedy algorithm

The greedy reconstruction algorithm chooses the unreconstructed track near the current head position and works as follows:

1. Maintain the head position by observing the location of the last foreground request.
2. When it is time to service a reconstruction request, read the unreconstructed track closest to the current head position.
3. Mark the track as reconstructed.
4. If the entire disk has not yet been read, return to 1.

Unlike the ordered reconstruction algorithm, which requires only a single pointer, this algorithm must keep a table of the tracks reconstructed so far. However, using a single bit for each reconstructed track on a modern 72 GB disk requires only about 60 KB of memory.

# 5. ANALYSIS

We analyze the ordered and greedy algorithms using two different models—the vacationing server and the permanent customer models. The ordered algorithm analysis is based on previous work. The greedy algorithm analysis is our original work and is based on models well known in the queueing theory community. We apply these models to the particulars of disk drive characteristics.

Our analysis of the ordered reconstruction with vacationing server model is derived from the work of Thomasian and Menon [22]. We make minor adjustments to their analysis and apply it in the broader context of background tasks. We do not include a presentation of the permanent customer model of ordered reconstruction as we can directly apply the results of Merchant and Yu [8].

In our discussion we assume that the access pattern for the foreground requests is random and uniform over the entire disk, the request inter-arrival times are exponentially distributed with parameter $\lambda$, and that the request size is fixed and small (on the order of several KB).

## 5.1 Definitions

A foreground read request consists of a seek, whose duration we denote by $K$, some rotational latency $L$, and data transfer. We assume that the request size is fixed and small and hence the media transfer time is a small constant which we denote by $X$. Then, the mean time to service a request is $E[S] = E[K]+E[L]+E[X]$. Assuming that the seek time $K$ can be computed by equation $F(d) = A + Bd$ as a function of cylinder distance $d$, where $0 \leq d \leq 1$, the expected service time becomes

$$E[S] = A + B/3 + R/2 + X \tag{1}$$

given the random uniform request distribution, the average seek distance $d$ is $1/3$ with an average rotational latency of half a revolution $R$ ($E[L] = R/2$).

The distribution of seek time from a certain radial location $r$ (which we assume by symmetry to satisfy $r \leq 1/2$ ) to a random location is given by the PDF $J_r$, whose density $J'_r(t)$ is given by the formula[1]

$$J'_r(t) = \begin{cases} \frac{2}{B} & \text{if } A \leq t \leq A + rB \\ \frac{1}{B} & \text{if } A + rB \leq t \leq A + (1-r)B \end{cases} \tag{2}$$

## 5.2 Vacationing server model

In the vacationing server model we consider an M/G/1 server (exponential inter-arrival, general service time PDF, single server) with a FCFS queue. Foreground requests arrive to the server with arrival rate $\lambda$ and the request service time is $S$. When the queue contains no foreground requests (i.e., the queue is empty), the server takes a vacation. In our case a vacation is the service of a single background request.

The length of the first vacation is given by a PDF $V_1$. During that vacation no service is provided to the incoming foreground requests. If there is no request in the queue upon the completion of the first vacation, the server takes another vacation with length given by the PDF $V_2$. More generally, if the request queue is empty upon the completion of the $i$-th vacation, the server takes a vacation with PDF $V_{i+1}$, whose Laplace transform $V^*$ is defined as

$$V_i^*(t) = \int_0^\infty e^{-tx} V_i'(x)dx. \tag{3}$$

A busy period of the server starts with the arrival of a foreground request to an empty queue and ends when all foreground requests in the queue have been serviced and the queue is empty again. Therefore, a vacation occurs only at the end of a busy period. A busy cycle is thus the time between the starts of two consecutive busy periods, that is, it is a sum of all service times of the foreground requests in the queue and all vacations taken by the server after the queue is emptied.

### 5.2.1 *Ordered algorithm*

Ordered reconstruction is modeled by a two vacation model where all the vacations starting from the second are equally distributed, that is, $V_2 = V_3 = \ldots = V_i$ for all $i > 1$. The first vacation in the ordered reconstruction includes a seek to the location that was last reconstructed and reading the blocks of one reconstruction unit. Therefore, $V_1 = J_r(t) + R$ when the reconstruction unit is exactly one track of a zero-latency disk. For disks that do not implement zero-latency reads $V_1 = J_r(t) + 3R/2$.

The calculations in this paper assume zero-latency disks and thus use the former expression for $V_1$. Also note that adding the deterministic random variable $R$ to $J_r$ has the same effect as changing the term $A$ to $A + R$ in the seek time function $F(d)$. Thus, in all

[1] For a general seek time function $t = F(d)$ we have $J'_r(t) = 2/(F'(F^{-1}(t)))$ if $F(0) \leq t \leq F(r)$ and $J'_r(t) = 1/(F'(F^{-1}(t)))$ for $F(r) \leq t \leq F(1-r)$, where $F^{-1}$ refers to the inverse function which exists when $F$ is strictly increasing.

computations regarding $J_r$, replacing $A$ by $A + R$ will give the correct result for $J_r + R$.

Any vacation for $i > 1$ consists of head repositioning to the next track, which incurs a head switch or possibly a one-cylinder seek, and reading of the data from the track. Thus, $V_i = H + R$, where $H$ is the head switch time or one-cylinder seek time which are the same in today's high-end SCSI disks [13, 19].

The first request in a busy cycle has an exceptional service time $S_i$ as it has to wait for the $i$-th vacation to finish. Since the disk head position at the end of the vacation is approximately the same as it was at the start of the vacation and the foreground requests are randomly and uniformly distributed across the entire disk, we use the seek distribution $J_r$ to determine the seek portion of the first foreground request in a busy cycle. We approximate the moments of the waiting time of the first request by the moments of the residual vacation time as

$$\delta_{i,j} = \frac{\gamma_{i+1,j}}{(i+1)\gamma_{i,j}} \tag{4}$$

where $\gamma_{i,j}$ is the $i$-th moment of the $j$-th vacation and $\delta_{i,j}$ is the $i$-th residual moment of the $j$-th vacation as defined by [6, 20].

The probability of no request arrival during the first vacation period is

$$\int_0^\infty \left(1 - \left(1 - e^{-\lambda t}\right)\right) dJ_r(t) = \int_0^\infty e^{-\lambda t} dJ_r(t) = J_r^*(\lambda)$$

Hence, the probability that the first request in a busy cycle arrives during the first vacation is $1-(J_r+R)^*(\lambda)$. The mean service time for the first foreground request that arrived during the $i$-th vacation (including the waiting time until the end of the vacation) is

$$E[S_i] = \delta_{1,i} + E[J_r] + R/2 + X \tag{5}$$

where $\delta_{1,i}$ is the expected residual duration of the $i$-th vacation, $E[J_r]$ is the expected seek time from the location $r$ of the head after the vacation, and $R/2$ and $X$ the expected rotational latency and transfer time, respectively, of the request. A simple calculation yields

$$\begin{aligned} E[J_r] &= \int_A^{A+rB} \frac{2t}{B} dt + \int_{A+rB}^{A+(1-r)B} \frac{t}{B} dt \\ &= A + (\frac{1}{2} - r + r^2)B \end{aligned} \tag{6}$$

and

$$E[J_r^2] = \frac{(A+rB)^3 + (A+(1-r)B)^3 - 2A^3}{3B} \tag{7}$$

Now, we can express the first residual moment for the first vacation

$$\delta_{1,1} = \frac{\gamma_{2,1}}{2\gamma_{1,1}} = \frac{E[(J_r+R)^2]}{2E[J_r+R]} = \frac{(A+R+rB)^3 + (A+R+(1-r)B)^3 - 2(A+R)^3}{6B\left(A+R+(\frac{1}{2}-r+r^2)B\right)}$$

and the first residual moment for the second vacation

$$\delta_{1,2} = \frac{V_2^2}{2V_2} = \frac{(H+R)^2}{2(H+R)} = \frac{H+R}{2} \tag{8}$$

From Equation 2.2.33 in [20], the mean duration of a busy period after the server took an $i$-th vacation is

$$D_i = \frac{E[S_i]}{1-\rho} \tag{9}$$

where $\rho = \lambda E[S] = \lambda(A + B/3 + R/2 + X)$. During such a busy period, the first request is serviced with mean service time $E[S_i]$ and all other requests are serviced with mean service time $E[S]$. Hence the average number of requests which are serviced during a busy cycle is

$$N_i = 1 + \frac{\frac{E[S_i]}{1-\rho} - E[S_i]}{E[S]} = 1 + \frac{\lambda E[S_i]}{1-\rho} \tag{10}$$

The mean response time for requests in a busy period is by Equation 2.2.40a in [20]

$$\begin{aligned} E[T_i] &= \frac{\lambda E[S^2]}{2(1-\rho)} + \frac{\lambda(E[S_i^2] - E[S^2])}{2(1+\lambda E[S_i] - \rho)} + \\ &\quad + \frac{E[S_i]}{1+\lambda E[S_i] - \rho} \end{aligned} \tag{11}$$

The first term comes from the Pollaczek-Khinchine formula, the second is a correction to the formula coming from the first exceptional request and the third term is the mean service time.

Before we can calculate $E[T_i]$ explicitly, we need to express the terms $E[S^2]$ and $E[S_i^2]$. Recalling that the service time $S$ is the sum of seek time $K$, latency $L$, and transfer time $X$, the second moment of service time $E[S^2] = E[(K+L+X)^2]$. Since all three components are independent of each other, we only need to find the densities and second moments of $K$, $L$, and $X$. Thus[2],

$$\begin{aligned} K'(t) &= \frac{2(A+B-t)}{B^2} \text{ for } A \le t \le A+B \\ L' &= \frac{1}{R} \text{ for } 0 \le t \le R \end{aligned}$$

and

$$\begin{aligned} E[K^2] &= \frac{2}{B^2}\int_A^{A+B} (A+B-t)t^2 dt = A^2 + \frac{2AB}{3} + \frac{B^2}{6} \\ E[L^2] &= \frac{1}{R}\int_0^R t^2 dt = \frac{R^2}{3} \end{aligned}$$

Since the transfer time $X$ is by our assumption deterministic, its density and second moment are 0 and $X^2$, respectively. Therefore,

$$\begin{aligned} E[S^2] &= E[(K+L+X)^2] \\ &= A^2 + \frac{2AB}{3} + \frac{B^2}{6} + \frac{R^2}{3} + R\left(A + \frac{B}{3}\right) + \\ &\quad + 2X\left(A + \frac{B}{3}\right) + RX + X^2 \end{aligned} \tag{12}$$

Finally, we express the second moment of the exceptional service time, $E[S_i^2]$, to get

$$\begin{aligned} E[S_i^2] &= E[(\Delta_i + J_r + L + X)^2] \\ &= \Delta_i^2 + 2\Delta_i J_r + J_r^2 + \Delta_i R + J_r R + \\ &\quad + \frac{R^2}{3} + 2\Delta_i X + 2J_r X + RX + X^2 \end{aligned} \tag{13}$$

where $\Delta_i = \delta_{1,i}$ and $\Delta_i^2 = \delta_{2,i}$. All the terms have been computed except for $\delta_{2,i}$. For $i = 2$, $\Delta_i^2 = \delta_{2,2} = (H+R)/3$, and for $i = 1$,

[2]For a general seek function $t = F(d)$ we have $K'(t) = 2(1 - F^{-1}(t))/(F'(F^{-1}(t)))$.

we can compute $\delta_{2,1}$ from Equation 4. Thus,

$$\delta_{2,1} = \frac{\gamma_{3,1}}{3\gamma_{2,1}} = \frac{E[(J_r + R)^3]}{3E[(J_r + R)^2]} = \frac{(A+R+rB)^4 + (A+R+(1-r)B)^4 - 2(A+R)^4}{4((A+R+rB)^3 + (A+R+(1-r)B)^3 - 2(A+R)^3)}$$

The mean time between the start of two consecutive busy periods, $Z$, is given by

$$Z = \frac{1}{\lambda} + \frac{E[S_i]}{1-\rho} \tag{14}$$

The average number of vacations, $M$, taken between these two busy periods is $M = \sum_{i=1}^{\infty} i p_i$, where $p_i$ is the probability that $i$ vacations were taken. For ordered reconstruction with two types of vacations

$$\begin{aligned} p_1 &= 1 - V_1^*(\lambda) \\ p_i &= (1 - V_2^*(\lambda))V_1^*(\lambda)V_2^*(\lambda)^{i-2} \qquad \text{for } i > 1 \end{aligned}$$

and therefore

$$M = 1 + \frac{V_1^*(\lambda)}{1 - V_2^*(\lambda)} \tag{15}$$

If $V_1 = V_2 = V$ (i.e., when all vacations taken after a busy period are the same), $M = 1/(1 - V^*(\lambda))$.

It is easy to see that the average reconstruction time of a single track at radius $r$ is the average time between the starts of two busy periods, $Z_r$, divided by the average number of tracks reconstructed during a single busy cycle, $M_r$.

There are two types of busy periods that depend on the type of vacation taken when the first request of the busy period arrived. We denote their durations as $D_1$ and $D_2$ and compute them by Equation 9. Furthermore, since the arrival process is exponential, the average time between the end of a busy period and the start of a new one is $1/\lambda$. Thus,

$$\begin{aligned} Z_r &= \frac{1}{\lambda} + p_1 D_1 + (1 - p_1)D_2 \\ &= \frac{1}{\lambda} + (1 - (J_r + R)^*(\lambda))D_1 + (J_r + R)^*(\lambda)D_2 \end{aligned}$$

To compute the total reconstruction time, $T_{total}$, we sum over all radii and multiply by the number of tracks being reconstructed to get

$$T_{total} = N_{tracks} \int_0^1 \frac{Z_r}{M_r} dr \tag{16}$$

The vacationing server formulas allow us to compute this time explicitly, where

$$\begin{aligned} V_1^*(\lambda) &= \frac{e^{-\lambda(A+R)}}{\lambda B}\left(2 - e^{-\lambda r B} - e^{-\lambda(1-r)B}\right) \\ V_2^*(\lambda) &= e^{-\lambda(H+R)} \end{aligned}$$

To calculate the average response time of the foreground request, $E[T]$, when the process is at radius $r$, we look at the average number of requests serviced during a busy cycle, $N_i$, which was expressed by Equation 10. To express $E[T]$ more conveniently we define $Q_i = (1-\rho)N_i = 1 - \rho + \lambda E(S_i)$ and have

$$E[T] = \frac{(1 - V_1^*(\lambda))Q_1 E[T_1] + V_1^*(\lambda)Q_2 E[T_2]}{(1 - V_1^*(\lambda))Q_1 + V_1^*(\lambda)Q_2} \tag{17}$$

Having defined previously all the terms in this equation, we can compute explicitly the mean response time of the foreground request.

### 5.2.2 *Greedy algorithm*

The analysis of the greedy reconstruction algorithm is simpler if we make two assumptions:

(i) we can find an unreconstructed track very close to the current head position

(ii) after reconstructing the $i$-th track, we do not have to seek far to reconstruct another track.

These two assumptions, which are closely related (the second is an iteration of the first), allow us to apply a single vacation model with $V_i = H + R$ and to compute $Z$ and $M$.

To compute the average response time of the foreground request, we use the expected wait time $E[W] = \lambda E[S^2]/(2(1-\rho)) + \delta_{1,1}$ from equation 2.2.14a of [20] and get

$$E[T] = E[S] + E[W] = E[S] + \frac{\lambda E[S^2]}{2(1-\rho)} + \frac{H+R}{2} \tag{18}$$

where $E[S]$ is expressed in Equation 1 and $E[S^2]$ in Equation 12.

Finally, the time needed to reconstruct a single track, $E[T_{track}]$, is the same as the average response time of a background request. Thus, using the equation from [20] for calculating the average duration of a vacation, we get

$$E[T_{track}] = \frac{E[V]}{1-\rho} = \frac{H+R}{1-\rho} \tag{19}$$

## 5.3 Permanent customer model

The queue of the permanent customer M/G/1 service center always contains a single background request (i.e., "customer"). As soon as a background request is serviced, a new one is inserted into the FCFS queue. Both foreground and background requests have the same priority.

### 5.3.1 *Ordered algorithm*

The analysis of the ordered reconstruction using a permanent customer model has been done previously by Merchant and Yu [8].

### 5.3.2 *Greedy algorithm*

To analyze the greedy algorithm for the permanent customer model we can apply the gated service vacationing server model [20]. Using the equation for the mean response time in the gated service vacationing server (equation 2.5.24a in [20]), we get

$$E[T] = E[S] + \frac{\lambda E(S^2)}{2(1-\rho)} + \delta_{1,1} + \frac{\rho(H+R)}{1-\rho} \tag{20}$$

Compared to Equation 18, which calculates the response time in the greedy vacationing server model, this expression includes an additional term. Thus, the response time of the foreground request using the vacationing server model will always be better. This is not surprising because the vacationing server model gives the foreground tasks higher priority, while the permanent customer model treats the foreground and background requests equally.

As before, the average time needed to reconstruct a single track in the permanent customer model, $E[T_{track}]$, is equal to that of the vacationing server model. Hence, there is no benefit in making

Suppose there are 16 locations to be reconstructed ($N = 16$), $X_{12} = 10$, and that the locations $0, 2, 3, 5, 6, 8, 9, 10, 11, 14, 15$ are already marked. We first check for $k = 0$ if the interval $I_{10,0} = \{10\}$ is fully marked. It is and so are the intervals $I_{10,1} = \{10, 11\}$ and $I_{10,2} = \{8, 9, 10, 11\}$ for $k = 1$ and $k = 2$. We then consider $I_{10,3} = \{8, 9, 10, 11, 12, 13, 14, 15\}$ and find out that in that dyadic interval 12 and 13 are unmarked. We then randomly choose one of them, say 13, as $Y_{12}$.

**Figure 1: An example of a dyadic algorithm.**

the priority of the background and foreground requests the same, especially since the mean response time, $E[T]$, of the foreground request is affected more than in the vacationing server model.

These results seems at first to contradict the "no free lunch theorem" [6] which states that, in a non-preemptive priority system where the arrival and service times of requests are independent of the priorities, a weighted average of the response times for the priority classes is independent of the priorities. However, since the arrival times in the permanent customer model are not independent of the priority assignment, the assumptions of the theorem do not hold and thus the theorem does not apply. Other cases, in which the dependence of service time on priorities leads to similar anomalies, have been observed by other researchers [1, 20].

## 5.4 Assumptions for the greedy algorithm

We now discuss the assumptions in Section 5.2.2 that helped us with the analysis of the greedy reconstruction algorithm. Specifically, we prove a theorem which states that, until the very end of the reconstruction process, all background request seeks are at most logarithmic (in the number of tracks).

Consider a disk with $N$ tracks and assume that $(j-1)$ tracks, whose locations are $T_1, ..., T_{j-1}$, have already been reconstructed. Let $X_j$ be the location of the disk head before the reconstruction of the $j$-th track. If the $j$-th track is the first one to be reconstructed after a busy period, then $X_j$ is a uniformly distributed random location on the disk. Otherwise, the $j$-th track is reconstructed immediately after the $(j-1)$-th track in the same vacation period and $X_j = T_{j-1}$. We define $S_j = |T_j - X_j|$ to be the seek distance to the $j$-th background request's location.

A dyadic interval of order $k$ is a set of all integers whose binary representation is $yz$ with a fixed bit pattern $y$ and a variable bit pattern $z$ of length $k$. Stated otherwise, a dyadic interval of order $k$ consists of all integers in the range $y2^k, y2^k+1, \ldots, y2^k+(2^k-1)$. For a given $k$ and a given integer $X$, we denote by $I_{X,k}$ the dyadic interval of order $k$ containing $X$. Note that dyadic intervals of the same order are either disjoint or equal.

Using our notation, we now define a variant of the greedy algorithm, called *dyadic*, that is easier to analyze than the greedy algorithm in Section 4.2. Given $X_j$, search for the minimal $k$ for which a dyadic interval $I_{X_j,k}$ is not fully marked. To obtain $T_j$, randomly choose an unmarked integer from the interval $I_{X_j,k}$ as shown in the example listed in Figure 1. Note that any observations about the dyadic algorithm apply to the original greedy algorithm, because the track chosen by this algorithm may be even closer to the current head position than the one chosen by the dyadic algorithm.

An important property of the dyadic algorithm is that $T_i$ is chosen from the interval $I_{X_i,k}$ only if $I_{X_i,k-1}$ is fully marked (i.e., all locations closer to $X_i$ have already been reconstructed). This property is important for proving the following theorem that validates our assumptions of the greedy algorithm.

THEOREM 1. *Consider a greedy reconstruction process implemented by the dyadic algorithm using the vacationing server model. Then there exist constants $s_1(c)$ and $s_2(c)$ dependent on $c$ with $0 < c < 1$ such that, with high probability, it is possible to reconstruct $cN$ tracks with average seek $s_1(c)$ and without ever seeking more than $s_2(c) \ln N$ tracks.*

We first prove this theorem for the case when the server only takes a single vacation between busy periods. We then show how to adjust the argument to the more general case where more than one track is reconstructed after a single busy period, thus validating both assumptions about the greedy algorithm.

PROOF. For a given $k$, let $j(k) = cN$ be the first time index for which $S_j > 2^{k-1}$. This implies that $T_j$ is not in $I_{X_j,k-1}$ and hence, by the main property of the dyadic algorithm, $I_{X_j,k-1}$ is fully marked at time $j(k)$. Let $\Im$ be the first interval of order $k-1$ which was fully marked.

We show that with high probability no interval of size greater or equal to $\left(-1/\ln ce^{1-c}\right)\ln N$ is fully marked. Hence this interval size bounds the maximal seek distance up to the reconstruction of $cN$ tracks. To show this, we express the probability of a given interval of order $k-1$ being fully marked and determine when this probability is $o(1/N)$. Since there are at most $N/2^{k-1}$ intervals of order $k-1$, this will ensure that with probability $1 - o(1)$ none of them are marked.

Let $\Im = \{y_{k-1}2^{k-1}, y_{k-1}2^{k-1} + 1, \ldots, (y_{k-1}2^{k-1} + (2^{k-1} - 1))\}$, $t_m$ be the time in which $Y_{t_m} = z_{k-1}2^{k-1} + m$ was marked, and let $t_m \le j(k)$ with $m \in \{0, \ldots, 2^{k-1} - 1\}$. We claim that $X_{t_m} \in \Im$ for all $m$. We call intervals with this property *self-marking*.

Assume that for some $m$, $X_{t_m} \notin \Im$. Then $I_{X_{t_m},k-1}$ is a dyadic interval of order $k-1$ that differs from $\Im$ and hence is disjoint from it. We conclude that $Y_{t_m}$ is not in $I_{X_{t_m},k-1}$. However, the main property of the dyadic algorithm implies that $I_{X_{t_m},k-1}$ was already fully marked at time $t_m$, contradicting the definition $\Im$ as the first fully marked interval of order $k-1$. Conversely, if there are $2^{k-1}$ time indexes all less than or equal to $j(k)$ for which $X_{t_i} \in \Im$, then by time $j(k)$, $\Im$ is fully marked.

Let $l = 2^{k-1}$ be the probability that exactly $l$ points $X_{t_0}, \ldots, X_{t_{l-1}}$ fall in the interval $\Im$, as given by a $\mathsf{Binomial}(cN, l/N)$ distribution, which we denote $B$ and define as

$$
\begin{aligned}
P(B = l) &= \frac{(cN)!}{l!(cN-l)!}\left(\frac{l}{N}\right)^l\left(1 - \frac{l}{N}\right)^{cN-l} \\
&\le \frac{(cl)^l}{l!}e^{-lc}\left(1 - \frac{l}{N}\right)^{-l}
\end{aligned}
$$

For the purposes of the proof, let $l$ be of the form $s \ln N$ with $s$ being a constant not dependent on $N$. Then, the last factor in the expression for $P(B = l)$ is approximately equal to one. Thus we can ignore it and our estimate for $P(B = l)$ becomes a $\mathsf{Poisson}(cl)$ that we denote by $C$.

For our claim that $\Im$ will be fully marked after $cN$ steps, we are interested in the probability $P(C = l)$. Since we do not want any of the dyadic intervals of order $k-1$ to be fully marked, we want this probability to be $o(1/N)$.

Using Stirling's approximation, we get

$$P(C = l) = \frac{(cl)^l}{l!}e^{-cl} \approx \left(\frac{c}{e^{c-1}}\right)^l (2\pi l)^{-1/2}$$

Letting $l = s\ln N$, $q_c = ce^{1-c}$, and ignoring the non-dominant term $(2\pi l)^{-1/2}$, we get $P(C = s\ln N) \approx q_c^{s\ln N} = N^{s\ln q_c}$ Since $e^x \geq x+1$, we observe that $q_c < 1$. Setting $s\ln q_c < -1$ gives the inequality $s > -1/\ln q_c > 0$ from which we can express the upper bound on the seek distance as

$$s_2(c) = \frac{-1}{\ln q_c} = \frac{1}{c-1-\ln c}$$

To prove the assertion regarding $s_1(c)$ we need the following combinatorial lemma which can be proved using an inductive argument from the main property of the dyadic process.

LEMMA 1. *Let $I$ be a dyadic interval of size $2^k$. For all $m > k$ let $I_m$ be a unique dyadic interval of size $2^m$ which contains $I$. Assume that $I$ became fully marked at time $t$ and that $I$ was not self marking. Then, there exists a self marking dyadic interval $I'$, of the form $I_m - I_{m-1}$, which became fully marked at time $t' < t$.*

$S_j > l$ only if $I = I(X_j, k)$ is fully marked. By the lemma, either $I$ was self marking, or a specific interval of size $2^{m-1} = l2^{m-1-k} \geq l$ was self marking. From our previous computation, the probability that a dyadic interval of size $l'$ (where $l' = l2^{m-1-k}$) is self marking by time $j$ is approximately $(2\pi l')^{-1/2}q_c^{l'}$. Summing over all possibilities we obtain the estimate

$$\begin{aligned}
P(S_j > l) &\leq (2\pi l)^{-1/2}q_c^l + \sum_{h=1}^{\infty}\left(2\pi l 2^h\right)^{-1/2} q_c^{l2^{h-1}} \\
&\leq (2\pi l)^{-1/2}\left(q_c^l + \sum_{h=1}^{\infty} q_c^{lh}\right) \\
&\leq 2(2\pi l)^{-1/2}\left(\frac{q_c^l}{1-q_c}\right) \\
&\leq \frac{2q_c^l}{1-q_c}.
\end{aligned}$$

Using this estimate for all $2^k \leq h < 2^{k+1}$ we obtain the following bound on the average seek at time $j$

$$\begin{aligned}
s_1(c) &= \sum_{h=0}^{\infty} hP(S_j = h) \leq \frac{2}{1-q_c}\sum_{i=0}^{\infty} lq_c^i \\
&= \frac{2}{(1-q_c)}\frac{1}{(1-q_c)^2} \\
&= \frac{2}{(1-ce^{c-1})^3}
\end{aligned}$$

Next we demonstrate how to extend the results to the situation when $X_j = T_{j-1}$. Since we derived $s_1(c)$ from the existence of $s_2(c)$, it is sufficient to establish the existence of the latter constant. Since reading a single track takes $H + R$ time, the number of tracks reconstructed between busy periods is essentially exponentially distributed with parameter $\lambda_1 = \lambda(H+R)$ because of the exponentially distributed foreground request arrival times.

Instead of considering the maximal seek after $j = cN$ tracks have been reconstructed, we consider the maximal seek after $cN\lambda_1$ vacation periods. By the law of large numbers, the number of tracks reconstructed during those periods is essentially equal to $j = cN$, so this change is immaterial.

Repeating our arguments from the previous case, the number of vacation periods that began with the disk head positioned in a dyadic interval $I$ of size $l$ may be approximated by a $\mathsf{Poisson}(c\lambda_1 l)$ distribution. Since each vacation period entails an exponentially distributed number of track reconstructions, we need to estimate the probability that the sum of (approximately) $c\lambda_1 l$ exponential distributions with parameter $\lambda_1$ will be greater then $l$.

This sum has a Gamma distribution and the probability of our interest is given by

$$e^{-\lambda_1 l}\left(1 + \lambda_1 l + \frac{(\lambda_1 l)^2}{2!} + \ldots + \frac{(\lambda_1 l)^{c\lambda_1 l}}{(c\lambda_1 l)!}\right).$$

Since $c < 1$, this distribution is dominated by the last term, which, by Stirling's formula, has order of magnitude $r_c^l = \left(e^{c-1}/c^c\right)^{\lambda_1 l}$. Hence, $s_2(c) = -1/\ln r_c$ and we express $s_1(c)$ as before. $\square$

The dyadic reconstruction algorithm is closely related to linear probe hashing. The main difference is that linear probe hashing assumes a circular geometry where bins (disk drive tracks) 0 and $N-1$ are adjacent to each other. The circular symmetry allows exact calculations of the maximal seek distance for the hashing (reconstruction) process [11, 12] . Because the tracks of a real disk are not laid out in this way, we had to use the dyadic algorithm to avoid costly full-strobe seeks between tracks 0 and $N-1$.

Despite the difference in geometry, our maximal seek distance calculations are in complete agreement with previous work on linear probe hashing with sparse tables [12]. Our average seek calculations lead to the same orders of magnitude as those of linear probe hashing [5].

# 6. EVALUATION

This section compares the performance of the ordered and greedy algorithms. We show numerical results for the analytic solutions presented in Section 5 and validate those against measurements of an experimental system with a SCSI disk connected to a PC workstation.

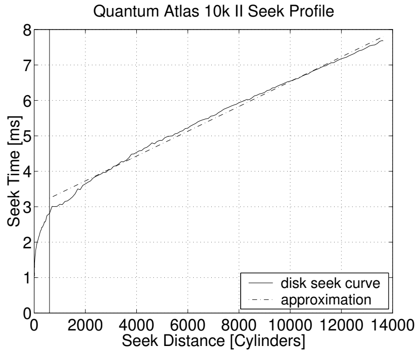
## 6.1 Experimental setup

All experiments were performed on a 550 MHz Pentium III PC workstation with 128 MB of memory running the Linux 2.4.2 kernel. We report results for Quantum Atlas 10k II and Seagate Cheetah X15 disks. Both disks were connected via an Adaptec AHA-7892A 160 MB SCSI adapter. Basic characteristics of the two disks are listed in Figure 2(a).

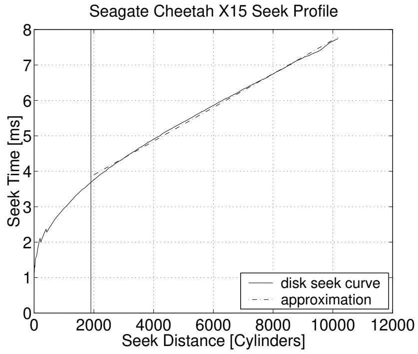
We wrote a user-level application that generates both foreground and background requests. In this application, the background task is a full disk scan which is the same as RAID1 reconstruction after

| | Quantum Atlas 10k II | Seagate Cheetah X15 |
|---|---|---|
| RPM | 10000 | 15000 |
| Head switch | 0.6 ms | 0.8 ms |
| Avg. seek | 4.7 ms | 3.9 ms |
| Zero-latency | yes | no |
| Cylinders | 13630 | 10377 |
| Surfaces | 3 | 10 |
| Sectors/track | 353—528 | 286—386 |
| Capacity | 9 GB | 18 GB |

(a) **Disk Characteristics.**




(b) **Quantum Atlas 10k II.**




(c) **Seagate Cheetah X15.**

**Figure 2: Characteristics of disks used in the experiments. The graphs plot the seek time as a function of distance in cylinders. The dashed line shows best fit to the linear portion of the seek curve. The vertical line shows the cutoff point.**

a disk failure. Thus, when all blocks of the disk are collected, the application stops. The requests are directly issued to the disk via Linux' SCSI generic driver, thus bypassing the file system cache.

The foreground requests are uniformly distributed across the entire disk and have exponential inter-arrival rates. The size of each request is 4 KB. The application issues only one request at a time to avoid possible reordering at the disk and keeps the remaining outstanding requests in its internal queue.

Background requests are issued by the greedy or ordered algorithm according to the vacationing or permanent customer models. The request sizes are set to read the full track and are adjusted to account for different numbers of sectors per track for different disk zones. Using the precise layout information obtained from the disk by the DIXtrac tool [17] ensures that a single background request never crosses a track boundary even when the track contains defects or spare sectors. The application is instrumented to measure response times and to keep various statistics about queue depths and seek distances. The seek distances were obtained from the information DIXtrac provides.

## 6.2 Numerical results

To numerically evaluate the results of our analysis in Section 5, we first need to determine the values for the model parameters. From the Atlas 10k II disk characteristics listed in Figure 2(a) we have $H = 0.6$ ms, $X = 0.1$ ms, and $R = 6.0$ ms. To determine the values for parameters $A$ and $B$, we approximate the real disk seek profile (depicted in Figure 2(b)) with a piecewise seek equation described by Ruemmler and Wilkes [16]. The piecewise seek equation $F(d)$ for seek distance $d$ is defined as

$$F(d) = \begin{cases} A + Bd & \text{for } d \geq C \\ D + E\sqrt{d} & \text{for } 0 < d < C \end{cases}$$

Using linear best fit to the Quantum Atlas 10k II seek profile and normalizing for $0 < d \leq 1$, we determine $C = 0.0587$ (800 cylinders), $A = 3.02$, and $B = 4.77$. The dashed line in Figure 2(b) shows the best fit for the linear portion of the seek curve.

The numerical results for the vacationing server model with ordered and greedy algorithms are shown in Table 1(a) under the column *basic model*. The table lists, for a given arrival rate of foreground requests $\lambda$ (and the corresponding load $\rho = \lambda E[S]$ with $E[S]$ computed from Equation 1), the average response time of a foreground request with $r = 0.25$. This value of $r$ corresponds to an average seek of 1/4 cylinders after a vacation. Varying $r$ from 0.15 to 0.5 (the maximal value for our assumptions) yields response time differences of only a few tenths of a millisecond.

The greedy algorithm yields 7.3%–10.6% shorter response times for $\rho \leq 0.77$ ($\lambda \leq 100$) and 3.1% for $\rho = 0.93$ ($\lambda = 120$). However, note that for high loads of the system ($\rho > 0.62$, $\lambda > 80$), the average response time grows to 25 and 63 ms.

## 6.3 Experimental results

Using our experimental setup, we measured the response times and total run times for both the vacationing server and permanent customer models using ordered and greedy algorithms with different arrival rates of foreground requests. The results from these measurements of the Atlas 10k II disk are summarized in Figure 3. The first graph shows the average of the observed foreground request response times as a function of arrival rate $\lambda$. The second graph shows the total run time for a given model and algorithm used. The individual data points in both graphs represent an average of three runs.

### *6.3.1 Response time*

With increasing arrival rate of foreground requests, $\lambda$, the average response time of a foreground request increases for both the ordered and greedy algorithms. For the vacationing server model, the average response time of the foreground task is 3.8%–6.6% better when the greedy algorithm is used instead of the ordered algorithm. Similarly, the greedy algorithm with the permanent customer model yields 6.3%–14.7% better average response time than the ordered algorithm.

For all loads, the permanent customer model with ordered reconstruction gives worse average response time for the foreground requests than the vacationing server model for both greedy and ordered reconstruction. And finally, for a given load, the foreground requests mean response time for the greedy algorithm is always smaller than for the ordered reconstruction for both vacationing server and permanent customer models.

| Workload | | Avg. Response Time [ms] | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | *basic model* | | | *model with bus transfer* | | | *experiments* | |
| $\lambda$ | $\rho$ | Ordered | Greedy | *% diff* | Ordered | Greedy | *% diff* | Ordered | Greedy |
| 20 | 0.15 | 12.92 | 11.77 | 9.8% | 13.91 | 12.98 | 7.2% | 13.48 | 13.10 |
| 40 | 0.31 | 14.22 | 12.86 | 10.6% | 15.04 | 14.07 | 6.9% | 14.87 | 14.08 |
| 60 | 0.46 | 16.13 | 14.58 | 10.6% | 16.78 | 15.78 | 6.3% | 16.62 | 15.67 |
| 80 | 0.62 | 19.39 | 17.70 | 9.6% | 19.91 | 18.90 | 5.4% | 19.85 | 18.40 |
| 100 | 0.77 | 26.86 | 25.02 | 7.3% | 27.26 | 26.22 | 3.9% | 26.37 | 24.84 |
| 120 | 0.93 | 65.04 | 63.08 | 3.1% | 65.33 | 64.38 | 1.6% | 53.88 | 51.24 |

(a) **Quantum Atlas 10k II.**

| Workload | | Avg. Response Time [ms] | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | *basic model* | | | *model with bus transfer* | | | *experiments* | |
| $\lambda$ | $\rho$ | Ordered | Greedy | *% diff* | Ordered | Greedy | *% diff* | Ordered | Greedy |
| 20 | 0.13 | 11.27 | 10.60 | 6.4% | 13.54 | 13.04 | 3.8% | 13.30 | 13.01 |
| 40 | 0.29 | 11.87 | 11.34 | 4.7% | 14.16 | 13.94 | 1.6% | 14.85 | 14.05 |
| 60 | 0.43 | 12.81 | 12.40 | 3.3% | 15.28 | 15.28 | 0.0% | 15.72 | 15.46 |
| 80 | 0.57 | 14.38 | 14.08 | 2.1% | 17.33 | 17.52 | 1.1% | 17.43 | 17.80 |
| 100 | 0.72 | 17.29 | 17.09 | 1.2% | 22.68 | 22.03 | 1.6% | 22.86 | 22.30 |
| 120 | 0.85 | 24.28 | 24.09 | 0.5% | 35.21 | 35.69 | 1.4% | 36.44 | 36.23 |

(b) **Seagate Cheetah X15.**

**Table 1: Vacationing server model numerical results. The tables list the mean response time of a foreground request for different arrival rates $\lambda$ (requests/s) for the vacationing server model with ordered and greedy algorithms with $r = 0.25$. The *% diff* values show improvement of greedy over ordered. The columns *basic model* and *model with bus transfer* list numerical solutions with bus transfer $U$ set to 0 and to the disks' respective values of 2.4 and 2.2 ms. The column *experiments* shows the values measured on a real disk.**

The average response time for the background requests using the greedy reconstruction algorithm with the vacationing server model is 9.6 ms. This consists of a 0.6 ms head switch, 6.0 ms rotation and media transfer, 2.4 ms bus transfer, and 0.6 ms system overhead. In comparison, the average response time of background requests for the ordered reconstruction with the vacationing server model is 11.4 ms because it includes a seek time for moving to the location just beyond the last reconstructed track.

### *6.3.2 Total reconstruction time*

In addition to achieving better average response time, the greedy algorithm with the vacationing server model yields shorter total reconstruction time. Compared to ordered reconstruction, the improvement of total run time for the greedy reconstruction ranges from 4.9% for $\rho = 0.15$ ($\lambda = 20$), to 19.2% for $\rho = 0.62$ ($\lambda = 80$). For higher loads, the improvement is even greater.

The comparison of the reconstruction policies under different models gives an interesting result. For light loads ($\rho \leq 0.62$), the total reconstruction time of the greedy algorithm with the permanent customer model is at most 2.1% longer than that of the greedy algorithm under the vacationing server model.

This surprising result is completely consistent with our analysis which predicted equal completion times. For high loads ($\rho > 0.9$), however, the total reconstruction time of the greedy permanent customer model is shorter by 43%, giving an interesting trade-off between total reconstruction time and performance of foreground applications.

## 6.4 Validation of analytical solutions

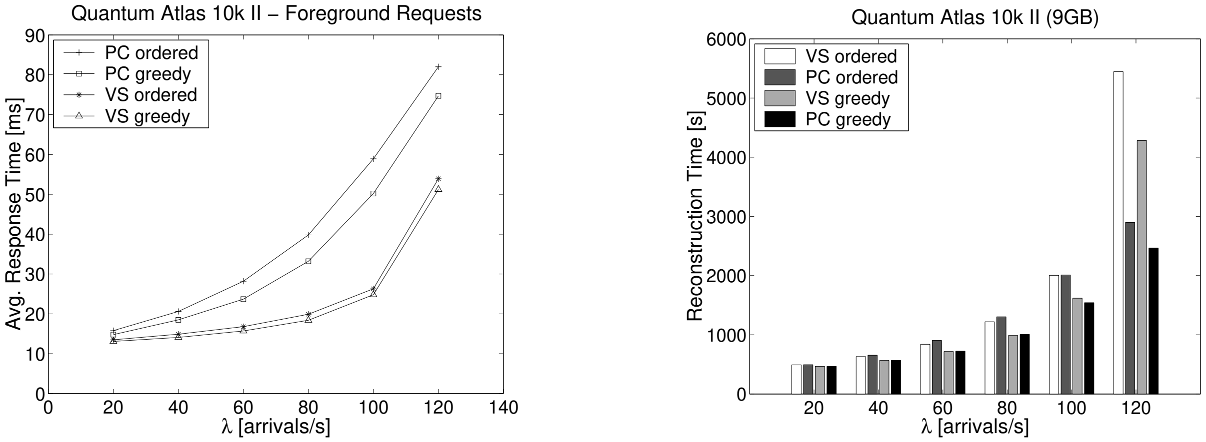
To be able to directly compare the numerical results of our analytical models with the measured response times, we introduce an additional parameter, $U$, which corresponds to the bus transfer time of the reconstruction unit. Thus we get $V_1 = J_r(t) + R + U$ for the first vacation and $V_i = H + R + U$ for the $i$-th vacation. From measurements on our experimental setup, $U = 2.4$ ms given an average track size of 200 KB for the Atlas 10k II disk.

With the exception of the values for $\rho = 0.93$ ($\lambda = 120$), the average response time for foreground requests obtained numerically and listed in Table 1(a) under the column labeled *model with bus transfer* differs from the experimental results listed in column *experiments* by at most 3% for both greedy and ordered vacationing server model. This favorable comparison thus validates our analytical model.

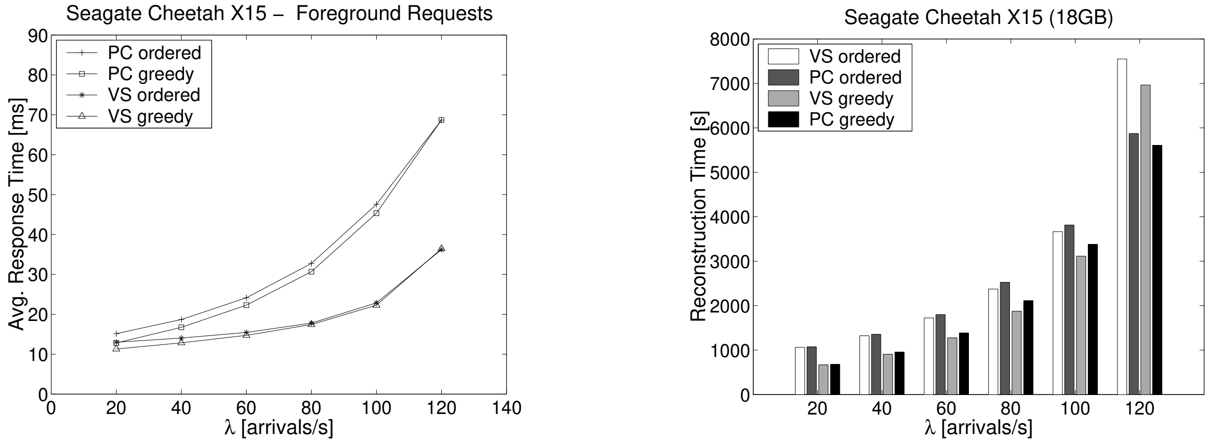
Notice that the numerical results for the *basic model* showed an average response time reduction of 10% for the greedy algorithm compared to the ordered algorithm, while for the *model with bus transfer*, the benefit is only 6%. However, when a background task does not need to transfer the data from the disk to the host, as is typically done in disk scrubbing implemented by the SCSI VERIFY command, we still expect the full 10% reduction in response time.

## 6.5 Results for other disks

We repeated the same experiments for an 18GB Seagate Cheetah X15 disk. Using the seek profile in Figure 2(c) and the data in Figure 2(a), we determined the values of the model parameters to be $C = 0.17$ (1800 cylinders), $A = 2.95$, $B = 4.83$, $H = 0.8$ ms,




(a) **Quantum Atlas 10k II.**




(b) **Seagate Cheetah X15.**

**Figure 3: Foreground request mean response time and total reconstruction time measured on a real disk. The x-axis show the foreground requests arrival rate $\lambda$ (requests/s). The data labeled PC is for the permanent customer model and VS is for the vacationing server model.**

$X = 0.1$ ms, $R = 4$ ms, and $U = 2.2$ ms (the Cheetah X15 disk has fewer sectors per track). Since this disk does not implement zero-latency access, we use $V_1 = J_r(t) + 3R/2 + U$.

The numerical results for the Cheetah X15 vacationing server model are displayed in Table 1(b). The trends are similar to the Atlas 10k II disk. However, there are a couple of interesting points. Notice that the greedy algorithm performs only slightly better than the ordered one (0%–3.8% for the model with bus transfer).

Since the Cheetah X15 disk does not implement zero-latency access, reading the track immediately next to the current head position still includes an average rotational latency of 2 ms (half of revolution). Note that the foreground response times are smaller than those of the Atlas 10k II disk, because the disk rotates at 15,000 RPM. Thus, the load on the disk is smaller given the same arrival rate $\lambda$.

The experimental results are displayed in Figure 3(b). Again, the trends are similar to the trends for the Atlas 10k II disk. Interestingly, even though the Cheetah X15 has double the capacity of the Atlas 10k II, the total reconstruction time is not twice as long. This is because of the shorter response times as discussed above.

## 6.6 Average seek distance

We now examine the average seek distance of background requests for the greedy algorithm to experimentally verify the result of Theorem 1. As can be seen from Table 2, for the majority of loads, there are no larger than logarithmic seeks when reconstructing up to 50% of the disk. In fact, the average seek distance in this range is less than 1 cylinder.

When reconstructing more than 50% of the disk, the number of "missed" seeks (i.e., seeks larger than logarithmic) grows moderately. Notice that for up to 90% of the disk, the average seek for a background request is at most 93 cylinders. Finally, reconstructing the last 5% of the disk accounts for almost half of the total missed seeks.

# 7. CONCLUSIONS

We have compared several algorithms for scheduling background tasks. We have also presented a new algorithm that has less impact on foreground activity of a storage system. Using an example of data reconstruction in RAID1 configurations, we have shown, both analytically and experimentally, a 6–10% shorter average response

| | *number of missed seeks* | | | | | | *average seek distance (cylinders)* | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *% disk* | 20 | 40 | 60 | 80 | 100 | 120 | 20 | 40 | 60 | 80 | 100 | 120 |
| 10% | 0 | 0 | 0 | 0 | 0 | 0 | 0.2 | 0.2 | 0.1 | 0.1 | 0.1 | 0.0 |
| 25% | 0 | 0 | 0 | 0 | 0 | 0 | 0.3 | 0.2 | 0.2 | 0.1 | 0.1 | 0.1 |
| 50% | 17 | 4 | 0 | 0 | 0 | 0 | 0.4 | 0.3 | 0.3 | 0.2 | 0.2 | 0.2 |
| 75% | 203 | 137 | 108 | 37 | 49 | 62 | 0.7 | 0.6 | 0.6 | 0.5 | 0.5 | 0.5 |
| 80% | 332 | 279 | 233 | 121 | 148 | 192 | 1.0 | 0.8 | 0.8 | 0.6 | 0.6 | 0.7 |
| 90% | 683 | 750 | 882 | 699 | 848 | 986 | 2.0 | 1.9 | 2.6 | 1.6 | 2.3 | 2.4 |
| 95% | 997 | 1243 | 1557 | 1469 | 1727 | 2070 | 7.3 | 10.3 | 37.3 | 36.3 | 42.7 | 92.9 |
| 99% | 1352 | 1800 | 2306 | 2437 | 2828 | 3356 | 51.1 | 79.0 | 135.6 | 158.1 | 193.7 | 280.1 |
| 100% | 1449 | 1978 | 2564 | 2773 | 3160 | 3756 | 63.9 | 105.5 | 178.4 | 211.2 | 248.8 | 344.9 |

**Table 2: Background request seek statistics for the Atlas 10k II disk. The column *% disk* represents the percentage of disk reconstructed so far. Missed seeks are seeks greater than $\log_2(\mathbf{MAXSEEK}) = 14$ cylinders. The other columns show the measured quantities for foreground request arrival rate $\lambda$ ranging from 20 to 120 reqs/s. The total number of background requests to reconstruct 100% of the disk was 40889.**

time for foreground requests when using this algorithm. The algorithm also reduces the total reconstruction time by up to 20%.

Furthermore, we have presented a theorem that shows that greedy reconstruction proceeds very efficiently for almost the entire reconstruction process and confirmed the experimental results of previous work by Lumb et al. [7]. Our future work will focus on quantifying later stages of the reconstruction process.

# 8. ACKNOWLEDGMENTS

We would like to thank Daniel Behrend and Benjamin Weiss for numerous discussions that helped us clarify our thoughts and Andy Klosterman for valuable suggestions on drafts of this paper.

# 9. REFERENCES

[1] Nikhil Bansal and Mor Harchol-Balter. Analysis of SRPT scheduling: investigating unfairness. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Cambridge, MA), pages 279–290, June 2001.

[2] O. J. Boxma and J. W. Cohen. The M/G/1 queue with permanent customers. *Journal of Selected Areas in Communications*, **9**(2):179–184, February 1991.

[3] Peter M. Chen and David A. Patterson. *Maximizing performance in a striped disk array*. UCB/CSD 90/559. Computer Science Div., Department of Electrical Engineering and Computer Science, University of California at Berkeley, February 1990.

[4] Mark Holland, Garth A. Gibson, and Daniel P. Siewiorek. Fast, on-line failure recovery in redundant disk arrays. *23rd International Symposium on Fault-Tolerant Compter Systems* (Toulouse, France, 22–24 June 1993), pages 422–431. IEEE Computing Services, August 1993.

[5] Svante Janson. Asymptotic distribution for the cost of linear probing hashing. *Random structures and algorithms*, **19**:438-471, 2001.

[6] Leonard Kleinrock. *Queueing systems volume I: theory*. John Wiley and Sons, 1975.

[7] Christopher R. Lumb, Jiri Schindler, Gregory R. Ganger, David F. Nagle, and Erik Riedel. Towards higher disk head utilization: extracting free bandwidth from busy disk drives. *Symposium on Operating Systems Design and Implementation* (San Diego, CA, 23–25 October 2000), pages 87–102. USENIX Association, 2000.

[8] A. Merchant and P. S. Yu. An analytical model of reconstruction time in mirrored disks. *Performance Evaluation*, **20**(1–3):115–129, May 1994.

[9] Richard R. Muntz and John C. S. Lui. Performance analysis of disk arrays under failure. *International Conference on Very Large Databases* (Brisbane, Australia), pages 162–173, 13–16 August 1990.

[10] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD International Conference on Management of Data* (Chicago, IL), pages 109–116, 1–3 June 1988.

[11] Yuri Pavlov. The asymptotic distribution of maximum tree size in a random forest. *Theory of probability and applications*, **22**:509-520, 1977.

[12] Boris Pittel. Linear probing: The probable largest search time grows logarithmically with the number of records. *Journal of algorithms*, **8**:236-249, 1987.

[13] Quantum Corporation. *Quantum Atlas 10K II 9.2/18.4/36.7/73.4 GB Ultra 160/m S product manual*, Document number 81-122517-04, June 2000.

[14] Erik Riedel, Christos Faloutsos, Greg Ganger, and David Nagle. *Data mining on an OLTP system (nearly) for free*. Technical report CMU–CS–99–151. Carnegie-Mellon University, Pittsburgh, PA, June 1999.

[15] Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. *ACM Symposium on Operating System Principles* (Pacific Grove, CA, 13–16 October 1991). Published as *Operating Systems Review*, **25**(5):1–15, 1991.

[16] Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *IEEE Computer*, **27**(3):17–28, March 1994.

[17] Jiri Schindler and Gregory R. Ganger. *Automated disk drive characterization*. Technical report CMU–CS–99–176. Carnegie-Mellon University, Pittsburgh, PA, December 1999.

[18] Jiri Schindler, John Linwood Griffin, Christopher R. Lumb, and Gregory R. Ganger. Track-aligned extents: matching access patterns to disk drive characteristics. *Conference on File and Storage Technologies* (Monterey, CA, 28–30 January 2002), pages 259–274. USENIX Association, 2002.

[19] Seagate. *Seagate Cheetah X15 FC disk drive ST318451FC/FCV product manual, volume 1*, Document number 83329486, June 2000.

[20] Hideaki Takagi. *Queueing Analysis Volume 1: Vacations and Priority Systems*. North-Holland, 1991.

[21] A. Thomasian and A. Tantawi. Approximate solutions for M/G/1 fork-join synchronization. *Winter Simulation Conference*, December 1994.

[22] Alexander Thomasian and Jai Menon. Performance analysis of RAID5 disk arrays with a vacationing server model for rebuild mode operation. *International Conference on Data Engineering* (Houston, TX), pages 111–119, February 1994.