An Age-Threshold Algorithm for Garbage Collection in Log-Structured Arrays and File Systems

Jai Menon
Larry Stockmeyer

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Abstract. In this paper, we propose and study a new algorithm for choosing segments for garbage collection in Log-Structured File Systems (LFS) and Log-Structured Arrays (LSA). We compare the performance of our new algorithm against previously known algorithms such as *greedy* and *cost-benefit* through simulation. The basic idea of our algorithm is that segments which have been recently filled by writes from the system should be forced to wait for a certain amount of time (the age-threshold) before they are allowed to become candidates for garbage collection. The expectation is that if the age-threshold is properly chosen, segments that have reached the age-threshold are unlikely to get significantly emptier due to future rewrites. Among segments that pass the age-threshold and become candidates for garbage collection, we select ones that will yield the most amount of free space. We show, through simulation, that our age-threshold algorithm is more efficient at garbage collection (produces more free space per garbage-collected segment) than greedy or cost-benefit; this means that designs using age-threshold will give better system performance than designs using greedy or cost-benefit. It is also simpler to implement a scalable version of the age-threshold algorithm than to implement a scalable version of the cost-benefit algorithm. The performance of the age-threshold algorithm depends on good choice of an age-threshold; therefore, we also give an analysis which can be used to choose an optimal age-threshold under certain workload assumptions. We also suggest how to choose good age-thresholds when nothing is known about the workload.

A short version of this paper appears in: *High Performance Computing Systems and Applications*, J. Schaeffer, ed., Kluwer Academic Publishers, 1998, 119-132.

1 Introduction

In this paper, we propose and study a new algorithm for garbage collection in Log-Structured File Systems (LFS), Log-Structured Disks (LSD) and Log-Structured Arrays (LSA). LSD and LSA are relatively new types of disk architectures [8] which are important because they can improve write performance over standard disk architectures, and because they can support disk compression.

Log-Structured Disks (LSD) and Log-Structured Arrays (LSA) borrow heavily from the log-structured file system (LFS) approach pioneered by Rosenblum and Ousterhout [10] at UC Berkeley. Our results apply equally well to LSD, LSA and to LFS systems, however our focus in this paper is on LSA. The LSA technique combines LSD and RAID and is typically executed in an outboard disk controller rather than in the file system. Because LSA is implemented in an outboard controller which has no understanding of files, it is more appropriate to think of LSA as a log-structured track manager rather than a log-structured file system. In this respect, LSA has some similarities to Loge [4] and to Logical Disk [3], both of which are implemented below the file system.

In LSA, updated data is written into new disk locations instead of being written in place. Large amounts of updated data are collected in controller memory and written together to a contiguous segment on the disks. Parity on this data is also computed in controller memory and written to the segment at the same time. This technique avoids the standard write penalty of RAID arrays [9], but requires a process called *garbage collection* to continually create new empty segments to substitute for the ones that got filled during the writing process. When data that went into a segment is rewritten by the system, the data will be placed into a new segment, and a hole or empty space forms in the segment which originally had the data. Garbage collection is the process of compacting some number of partially empty segments into a fewer number of full segments, thus creating some completely empty segments. This garbage collection process is sometimes called *segment cleaning* in the literature, e.g., [10].

The focus of this paper is on garbage collection in LSA and LFS systems. Specifically, we look at the impact on performance of the algorithm used to decide which segments should be chosen for garbage collection. Two algorithms have been previously proposed in the paper that introduced the log-structured approach [10]: a *greedy* algorithm which selects segments that will yield the most amount of free space; and a *cost-benefit* algorithm which selects segments based on both how much free space it has and how long it has been since the segment was last filled with a bunch of data by the controller (its age). The latter algorithm includes the age of the segment in the selection criterion, because the expectation is that younger segments will have data which is likely to be rewritten shortly, so if we wait a little longer, these segments will yield even more free space in the future. This argues for not selecting young segments. Although amount of free space and age are clearly important criteria to use in deciding which segments should be chosen for garbage collection, it is less clear how to combine these two criteria into a single criterion which can be used to order segments according to their desirability for garbage collection. A particular *cost-benefit* method for combining amount of free space and age into a single number is suggested in [10]; the details of this method are discussed later in the paper.

We introduce a new class of algorithms for selecting segments for garbage collection based on an age-threshold. The basic idea is that segments which have been recently filled by writes from the system should be forced to wait for a certain amount of time (the age-threshold) before they are allowed to become candidates for garbage collection. This way, we give the system a reasonable amount of time to rewrite any of the data in the segment, before we make it a candidate for garbage collection. The expectation is that if the age-threshold is properly chosen, segments that have reached the age-threshold are unlikely to get significantly emptier due to future rewrites. Among segments that pass the age-threshold and become candidates for garbage collection, we select ones that will yield the most amount of free space. We show, through simulation, that our age-threshold algorithms are better in performance than greedy or cost-benefit [10], where performance is measured by the average amount of free space produced per garbage-collected segment; this means that designs using age-threshold will give better system performance than designs using greedy or cost-benefit. The age-threshold algorithms are also simpler to implement than the cost-benefit algorithm, if we want a scalable design that continually maintains an ordering of the segments according to their desirability for garbage collection. The performance of the age-threshold algorithms depends on good choice of an age-threshold; therefore, we also give an analysis that can be used to choose an optimal age-threshold under certain workload assumptions. We also suggest how to choose good age-thresholds when nothing is known about the workload.

The rest of this paper is organized as follows. We begin with overviews of LSA and of the garbage collection process in LSA, including the definition of a measure we use to judge the relative merits of various garbage collection algorithms. This measure, called GCU, is the average utilization of garbage collected segments, where the *utilization* of a segment is the fraction of the segment containing live data (so that a segment with utilization u contains the fraction $1-u$ of free space). The smaller the GCU, the better the performance of a particular garbage collection algorithm. Next, we describe our simulation procedure and introduce two models of track writing – a uniform model where all tracks are equally likely to be written by the host system, and a hot-and-cold model where some tracks are more likely to be written than others. We then analyze the performance of the greedy algorithm for garbage collection in a simple model of the garbage collection process where data taken from garbage-collected segments is mixed with newly-written data in the same segment; we call this model the *mixing* model. The analysis is verified with simulation. Both analysis and simulation show that the inclusion of an age-threshold has no significant effect on GCU in the mixing model. This is followed by analysis and simulation of the basic age-threshold algorithm in the more realistic (e.g., it is the implementation of LFS in [10]) *separation* model, where data taken from garbage-collected segments is placed into segments separate from segments used for newly-written data. Here we find that as the age-threshold increases, GCU first decreases and then increases. For the basic age-threshold algorithm, our analysis can be used to select the optimal age-threshold which produces the lowest

GCU, under the hot-and-cold model of track writing. We also study several variations of the age-threshold algorithm. We compare these algorithms to the cost-benefit algorithm and show that the former algorithms are superior, over a range of age-thresholds. We then consider another synthetic trace model, based on a power law for cache miss ratio, where younger data is more likely to be rewritten than older data; in this model, the "hot" data gradually changes, rather than being fixed as in the hot-and-cold model. Comparing the various algorithms on this trace model, we again find that age-threshold algorithms are better. We then look at the performance of the various algorithms using an actual trace of I/O requests obtained from a running system [11]. This supports our findings that age-threshold algorithms are superior to cost-benefit and greedy, over a range of age-thresholds. We conclude by suggesting two schemes for choosing age-thresholds when nothing is known about the workload.

2 Overview of LSA

We begin by describing LSA which uses a parity technique similar to that used in RAIDs for improving reliability and availability. As we have mentioned before, our results apply also to LFS and LSD which do not use parity techniques; however, our main interest is in describing our results in the context of LSA.

A Log-Structured Array (LSA) operates as follows. It consists of a disk controller and $N+1$ physical disks, where each disk is divided into large consecutive areas called *segment-columns.* A segment-column consists of some number of consecutive sectors on the disk and is typically as large as a physical cylinder on a physical disk. Corresponding segment-columns from the $N+1$ disks constitute a *segment.* The array has as many segments as there are segment-columns on a disk in the array. One of the segment-columns of a segment contains the parity (XOR) of the remaining N segment-columns of the segment.

Logical devices are mapped and stored in this Log-Structured Array, and host programs access logical tracks stored on logical devices; the $N+1$ physical disks that make up the LSA are not visible to host programs, only logical devices are. A *logical track*, or simply track, which we will define for this paper as the smallest unit writable by the host system, is stored entirely within some segment-column of some physical disk of the array; many logical tracks can be stored in the same segment-column. The location of a logical track in an LSA changes each time the track is rewritten by a host program. A directory maintained by the LSA in the controller, called the LSA directory, indicates the current location of each logical track. It has an entry for each logical track. The entry has the logical track number, the physical disk and segment-column within the disk that it is mapped to, the starting sector within the column at which the logical track starts, and the length of the logical track in physical sectors. Given a request to read a logical track, the controller examines the LSA directory to determine the physical disk, starting sector and length in sectors to which the logical track is currently mapped, then reads the relevant sectors from the relevant disk.

Writes to the array operate as follows, using a section of controller memory, logically organized as $N+1$ segment-columns, called the *memory segment.* It consists of $N+1$

memory segment-columns: N data memory segment-columns and 1 parity memory segment-column. When a logical track is updated by the system, the entire logical track is written into one of the N data memory segment-columns (and the host is told that the write is done). When the memory segment is full (all data memory segment-columns are full) and cannot hold any new logical tracks from the system, we XOR all the data memory segment-columns to create the parity memory segment-column, then all $N + 1$ memory segment-columns are written to an empty segment on the disk array. All logical tracks that are written to disk from the memory segment must have their entries in the LSA directory updated to reflect their new disk locations. If these logical tracks had been written before by the system, the LSA directory would have contained their previous physical disk locations; else the LSA directory would have indicated that the logical track had never been written, so has no address. Note that writing to the disk is more efficient in LSA than in RAID-5, where 4 disk accesses are needed for an update [9]. However, LSA needs to do a process called *garbage collection*, since holes (garbage) form in segments that previously contained one or more of the logical tracks that were just written. To ensure that we always have an empty segment to write to, the controller garbage collects segments in the background. It could select for garbage collection those segments that have lots of holes (garbage) and/or those segments which are not anticipated to produce more holes for a long time. All logical tracks from a segment selected for garbage collection that are still in that segment (are still pointed to by the LSA directory) are read from disk and placed in a memory segment. They may be placed in the same memory segment used for holding logical tracks written by the system (the mixing model) or they may be placed in a different memory segment of its own (the separation model). In any case, these logical tracks will be written back to disk when the memory segment fills. Garbage collected segments are returned to the empty segment pool and are available when needed.

3 Overview of Garbage Collection

As we mentioned before, we call the smallest unit of data that is written by the host system a *track*. In a log-structured file system or LSA, these tracks are organized into *segments*. At any time, each track is *live* in exactly one segment. A basic operation of the system is to *write* a particular track, that is, change the contents of the track. We can imagine that the tracks being written are placed into (i.e., become live in) an initially empty segment s_0. While s_0 is being filled, s_0 resides in controller (possibly non-volatile) memory. If track k is written and if track k was previously live in some other segment s before this write, then k becomes *dead* in s, and k becomes live in the segment s_0 being filled. This continues until s_0 is filled to capacity, at which point s_0 is *destaged*, i.e., written to the disk (or disk array). Then another empty segment is chosen to be filled. As writing proceeds, storage becomes fragmented; there can be many segments that are partially filled with live tracks. At any point, the *utilization* of a segment is the fraction of the segment containing live tracks, i.e., if the segment contains L live tracks and if segment capacity is C tracks, then its utilization is L/C. For simplicity, in our simulations we assume that all the data (tracks) are loaded into the LSA at initialization time, and no new tracks are added or existing tracks deleted following this initial loading or initialization phase. That is, we assume that tracks are not added or deleted, just rewritten; therefore, the *average segment utilization* (ASU) of the system is constant. If there are S segments, if each segment has a capacity of C tracks, and if there are T tracks, then

$$\mathrm{ASU} = T/CS.$$

Since the writing process will eventually deplete the empty segments, garbage collection is done to create empty segments. This is done by choosing a certain number of segments and compacting the live tracks in these segments into a fewer number of full segments, so some empty segments are created. For example, if we collect 3 segments, each having utilization 2/3, the live tracks can be reorganized into 2 full segments, thus creating a net of one new empty segment.

3.1 Issues in the choice of a garbage collection algorithm

As discussed in [10], several issues must be resolved to make the garbage collection algorithm precise:

1. When should garbage collection be invoked?

2. How many empty segments should be created during each phase of garbage collection?

3. What segments should be chosen for garbage collection? For example, a possible criterion would be to choose those having smallest utilization, since they yield the most free space.

   Another possible criterion is based on the age of a segment. The younger a segment, the more likely it is to contain hot data which might be rewritten in the near future. Rewriting will cause the utilization of the segment to decrease, so such segments are not good candidates for garbage collection. This reasoning argues that old segments with lots of cold data are better candidates for garbage collection.

4. How should the live tracks collected by garbage collection be reorganized before packing them into segments? For example, one possibility studied in [10] is to group together tracks of similar age, i.e., tracks that were last written at around the same time.

One conclusion of [10] is that the last two issues are more important than the first two in affecting the performance of a log-structured file system. The main emphasis in this paper is on the selection criterion (issue 3). We also investigated, to a lesser extent, the effect of reorganization by age grouping.

In addition, we consider another issue, related to the fact that a track can move from one segment to another for two reasons: first, when the track is written (it moves from the segment where is was previously live to the segment being filled); and second, when it is in one of the segments chosen for garbage collection (it could move as the result of reorganization).

5. Should newly written tracks and garbage collected tracks be mixed together in segments (the *mixing* model), or should these two types of tracks be kept in separate segments (the *separation* model)? (In [10], only the separation model is considered.)

3.2 Measures of performance

We measure the performance of a garbage collection algorithm by the average utilization of the garbage collected segments. We call this measure *garbage collection utilization* (GCU). This seems a reasonable measure of performance since small GCU means that the garbage collector is doing a good job of collecting free space, at an average of $C(1 - \text{GCU})$ per collected segment. GCU is related to other performance measures that have been considered elsewhere. The *write cost* of [10] is equal to $2/(1 - \text{GCU})$, and the *moves per write* of [7] is equal to $\text{GCU}/(1 - \text{GCU})$. Both of these are increasing functions of GCU, so that the relative ordering of different algorithms by GCU is the same ordering that would be obtained using either of the other measures. Note also that write cost and moves per write both approach infinity as GCU approaches 1; therefore, to get reasonable performance under these measures, GCU should not be too close to 1. This is consistent with the fact that if GCU gets too close to 1, the garbage collector will not be collecting much free space.

Previous studies of garbage collection algorithms show (not surprisingly) that GCU increases as the average utilization ASU increases, and that ASU should be somewhat less than 1 to allow some build-up of dead tracks, since garbage collection converts dead tracks to free space.

4 Models of Track Writing

The GCU of a particular garbage collection algorithm depends on the sequence of track writes. We begin by considering a synthetic model of track writing, used in [10], that assumes random choice of the track to be written, possibly allowing a certain fraction of the tracks (the "hot tracks") to be more likely to be chosen than the others (the "cold tracks"). More precisely, two models of track writing we consider (with the first being a special case of the second) are:

- *Uniform.* The next track to be written is chosen uniformly at random from among all the tracks.

- *Hot-and-Cold.* The degree of "hotness" is specified by two numbers h and p with $0 < h \le p < 1$, where h is the fraction of tracks that are hot and p is the probability of choosing from among the hot tracks. A particular subset $\mathcal{H}$ of the tracks is designated as the "hot tracks", such that $\mathcal{H}$ contains a fraction h of the tracks, i.e., $|\mathcal{H}| = hT$. Whenever a track choice k is needed:
  with probability p choose k at random from $\mathcal{H}$,
  or with probability $1 - p$ choose k at random from the tracks not in $\mathcal{H}$.

(The uniform version is equivalent to the hot-and-cold version in any case where $p = h$.)

Later in Section 16, we consider another synthetic trace model, based on a "power law" for cache miss ratio. In this model, younger tracks are more likely to be chosen than older tracks, and the set of "hot tracks" gradually changes, instead of being fixed as in the hot-and-cold model. Then in Section 17 we compare various garbage collection algorithms on an actual I/O trace, the "snake" trace collected by Ruemmler and Wilkes [11].

5 Notes on the Simulation Procedure

When simulating a particular garbage collection algorithm, the details of the simulation depend, of course, on the particular algorithm. Some details of the simulations, however, are common to all of our simulations. For simulations done with the uniform and hot-and-cold models described above, we used $S = 3000$ segments, each having capacity $C = 300$ tracks. These particular numbers were chosen to roughly correspond to a situation where a segment-column is a cylinder, a typical disk has 3000 cylinders, the LSA has 7 data disks and 1 parity disk per array, a segment consists of 7 cylinders (1 cylinder from each of 7 disks), a cylinder has 15 physical tracks, and a segment can hold approximately 300 logical tracks assuming that a logical track is 1/3 of a physical track. As it turns out, however, our analysis suggests that the results are relatively independent of the specific numbers we selected here. Other values for S and C are used in Sections 16 and 17.

Having chosen S and C, the number T of tracks is chosen to give a desired ASU ($T = \text{ASU} \times CS$). For hot-and-cold track choice, the set $\mathcal{H}$ of hot tracks is initially chosen to be a random set of the correct size to give a desired h ($|\mathcal{H}| = hT$). Unless stated otherwise, this choice $\mathcal{H}$ of the hot tracks is not changed as the simulation proceeds (although, as described later, we did investigate in some cases the effect of changing the set of hot tracks). At the start of the simulation, the tracks are packed into $\lceil T/C \rceil$ segments in sequential order; thus, at the start, $\lfloor T/C \rfloor$ segments are full (utilization 1) and $S - \lceil T/C \rceil$ are empty (utilization 0). To begin the simulation, there is an initial filling process where track writing is invoked to fill each empty segment, in turn, until there are no empty segments. Then simulation of the particular garbage collection algorithm is started. For synthetic (randomly generated) traces, the simulations were run until GCU was observed to stabilize, indicating that the simulation had reached steady state. Reported GCU values are the average utilization of the segments selected by the garbage collector, averaged over several tens of thousands of segment selections in steady state, and rounded to three decimal places.

We allow the number of empty segments to reach zero before starting a phase of garbage collection. In practice, to make sure that an empty segment will always be available for track writing, garbage collection would be started when the number of empty segments reaches some lower threshold (greater than zero). If we were to use a lower threshold in the simulations, thus forcing some number of segments to remain empty, the effect would be to increase the ASU in the segments that are not forced to be empty. Therefore, a lower threshold greater than zero could be simulated with a lower threshold of zero by increasing the ASU appropriately. So for simplicity, we only consider a lower threshold of zero.

If track writing causes a segment to become empty, that is, if all the tracks placed in the segment when it was last filled are rewritten so that all tracks in the segment become dead, then this segment is immediately placed in the pool of empty segments. In the computation of GCU, whenever this event occurs after the initial filling process it is viewed as though a segment of utilization 0 was collected by the garbage collector, since if a segment becomes empty through track writing the garbage collection algorithm should get "credit" for this.

6 The Mixing Model

We begin by investigating the mixing model (newly written tracks and garbage collected tracks are mixed in the same segment) with the greedy criterion for selecting a segment for garbage collection, where a segment having smallest utilization is selected. The procedure is to alternately execute the following two steps:

M1. Select a segment s having smallest utilization.

M2. Fill s by track writing.

In step M1, if more than one segment has smallest utilization, the oldest segment is selected, where the *age* of segment s is the number of iterations of this procedure that have occurred since the last time that s was selected in step M1. This procedure seems to contradict the description above, where only empty segments are filled by track writing. However, this procedure is shorthand for the equivalent procedure where the L live tracks in s are first moved to an empty segment of controller memory, this latter segment is further filled by $C - L$ track writes, and the filled segment is moved back to s.

6.1 Uniform track choice

We begin by analyzing how GCU depends on ASU under uniform track choice. Even though the uniform case is a special case of the hot-and-cold case, considering the uniform case first has the advantage of introducing the analysis method in a simple case. We can also begin to gain some confidence that the analysis is reasonable by comparing the results of analysis to the results of simulation in a simple case where GCU depends on only one "input" parameter, namely ASU. (Although other parameters such as the number of segments enter into the analysis, after making a reasonable approximation all parameters except ASU and GCU drop out.)

We recall some of the parameters introduced above and give shorter names to ASU and GCU:

$$\begin{aligned}
S &= \text{number of segments} \\
C &= \text{number of tracks per segment} \\
T &= \text{number of tracks} \\
a &= T/CS \quad (= \text{ASU}) \\
g &= \text{GCU}
\end{aligned}$$

	GCU	
ASU	analysis	simulation
.6	.324	.322
.7	.467	.464
.8	.629	.626
.9	.807	.804

Figure 1: Analysis vs simulation for the mixing model with uniform track choice.

In steady state, the life of a particular segment s follows a cycle. A cycle starts when s is selected in step M1 and filled in step M2. The utilization of s then decreases as tracks that were live in s are written to other segments, so that these tracks become dead in s. This continues until the age of s reaches S, at which point s the oldest segment (because there are only S segments); assuming that utilization decreases with age, the segment having age S is selected as the segment having smallest utilization, and the cycle repeats.

Thus, it is important to understand how the utilization of a segment depends on its age. As a first step, we consider how a single track write affects the expected utilization of a segment. Suppose that a segment s contains n (live) tracks. The probability that a track in s is chosen as the next written track is n/T. So the expected number of tracks in s after one track write is $n - n/T = n(1 - 1/T)$. Put another way, the expected number of tracks in s is multiplied by the factor $(1 - 1/T)$ each time a track is written.

Since each execution of step M2 involves $(1 - g)C$ track writes, there are a total of roughly $(1 - g)CS$ track writes between the time when s has utilization 1 and the time when its utilization has dropped to g. Therefore,

$$g = (1 - 1/T)^{(1-g)CS} \approx e^{-(1-g)CS/T} = e^{-(1-g)/a}.$$

Taking logs and rearranging, this gives[1]

$$a = \frac{1 - g}{\log_e(1/g)}. \tag{1}$$

For a given value of a (= ASU), (1) can be solved numerically to give the value of g (= GCU). Figure 1 shows, for four values of ASU, the GCU obtained from this analysis and the GCU obtained by simulation. In all cases, the analysis GCU is within 0.7% of the simulation GCU.

6.2 Hot-and-cold track choice

Now the analysis is extended to hot-and-cold track choice. There are now two new "input" parameters to the analysis, the fraction h of hot tracks and the probability p of choosing

[1] Equation (1) was first discovered by Jim Roche (personal communication), using a different argument.

from among the hot tracks. Again it is useful to see how the utilization of a segment decreases with age, but now we must take into account that some tracks in a segment are hot and some are cold. Suppose that a segment s contains n hot tracks. The probability that the next track write is one of these hot tracks is pn/hT, since some hot track is chosen with probability p, and segment s contains the fraction n/hT of the hT hot tracks. As above, this means that the expected number of hot tracks in s is multiplied by $(1 - p/hT)$ every time a track is written. Similarly, the multiplier for the expected number of cold tracks in s is $(1 - (1 - p)/((1 - h)T))$.

For a segment of age 1 (i.e., a segment that has just been filled by step M2 of the above procedure), let h' be the average fraction of the segment containing hot tracks; at this point, h' is unknown. As before, $(1 - g)CS$ track writes cause a segment, initially containing the fraction h' hot tracks and the fraction $(1 - h')$ cold tracks, to have its utilization drop from 1 to g. This gives

$$g = h'\left(1 - \frac{p}{hT}\right)^{(1-g)CS} + (1 - h')\left(1 - \frac{1 - p}{(1 - h)T}\right)^{(1-g)CS}.$$

Again using the approximation $(1 - x) \approx e^{-x}$ for small x, and using that $CS/T = 1/a$, this equation becomes

$$g = h'e^{-p(1-g)/ha} + (1 - h')e^{-(1-p)(1-g)/(1-h)a}. \qquad (2)$$

This equation is not sufficient because it contains two unknowns, g and h'. The unknown h' can be removed by noting that the number of hot tracks in a segment of age 1 (namely, $h'C$) equals the number of hot tracks in a segment of age S (namely, $h'Ce^{-p(1-g)/ha}$) plus the number of hot tracks added to the segment of age S when it is filled by track writing in step M2 (namely, $p(1 - g)C$), thereby converting it to a segment of age 1. That is,

$$h' = h'e^{-p(1-g)/ha} + p(1 - g). \qquad (3)$$

Since (3) is linear in h', we can solve for h' and substitute the result into (2), thus obtaining one equation in one unknown g. (If $p = h$, this equation reduces to the equation (1) derived for the uniform case.)

Figure 2 shows how the GCU obtained from this analysis compares to the GCU obtained from simulation for two choices of "hotness" and three values of ASU. In all cases tested, the analysis was accurate to within 0.7%.

Figure 3 illustrates how GCU depends on the degree of hotness in the case ASU = .8 and $h = .1$, where the GCU values were obtained from the analysis. GCU increases as the degree of hotness is increased from $p = .1$ (uniform track choice) to $p = .9$.

We also investigated how the inclusion of an age-threshold affects these results. With an age-threshold, the criterion for segment selection by the garbage collector is modified so that a segment is considered for collection only if its age exceeds the age-threshold, where the age of a segment is defined as above. The analysis predicts that an age-threshold should have no effect on the results, since the oldest segments tend to have the lowest utilization. This was verified by simulation.

			GCU	
ASU	h	p	analysis	simulation
.7	.1	.9	.613	.612
.8	.1	.9	.717	.716
.9	.1	.9	.838	.836
.7	.2	.8	.545	.543
.8	.2	.8	.671	.668
.9	.2	.8	.819	.817

Figure 2: Analysis vs simulation for the mixing model with hot-and-cold track choice.

p	GCU
.1	.629
.3	.639
.5	.662
.7	.690
.9	.717

Figure 3: GCU vs hotness in the mixing model, for ASU = .8 and $h = .1$.

7 The Separation Model

In the separation model, segments filled with newly written tracks are separate from segments filled by live tracks collected by the garbage collector. A parameter in the separation model is *max-empty*, the number of empty segments created during each phase of garbage collection.

At a high level, the procedure repeatedly executes the following two steps (after the initial segment filling process has reached a state where there are no empty segments):

S1. Run the garbage collector to create max-empty empty segments. Since the garbage collector compacts tracks from a certain number of somewhat empty segments to a fewer number of full segments, it should be clear that the garbage collector will need to examine and process greater than max-empty segments in order to create max-empty empty segments.

S2. Fill the empty segments, one at a time, by track writing.

It is useful in the separation model to draw a distinction between two types of segments: (1) segments that were last filled by track writing in step S2 (call these *TW-filled segments*); and (2) segments that were last filled with live tracks by the garbage collector in step S1

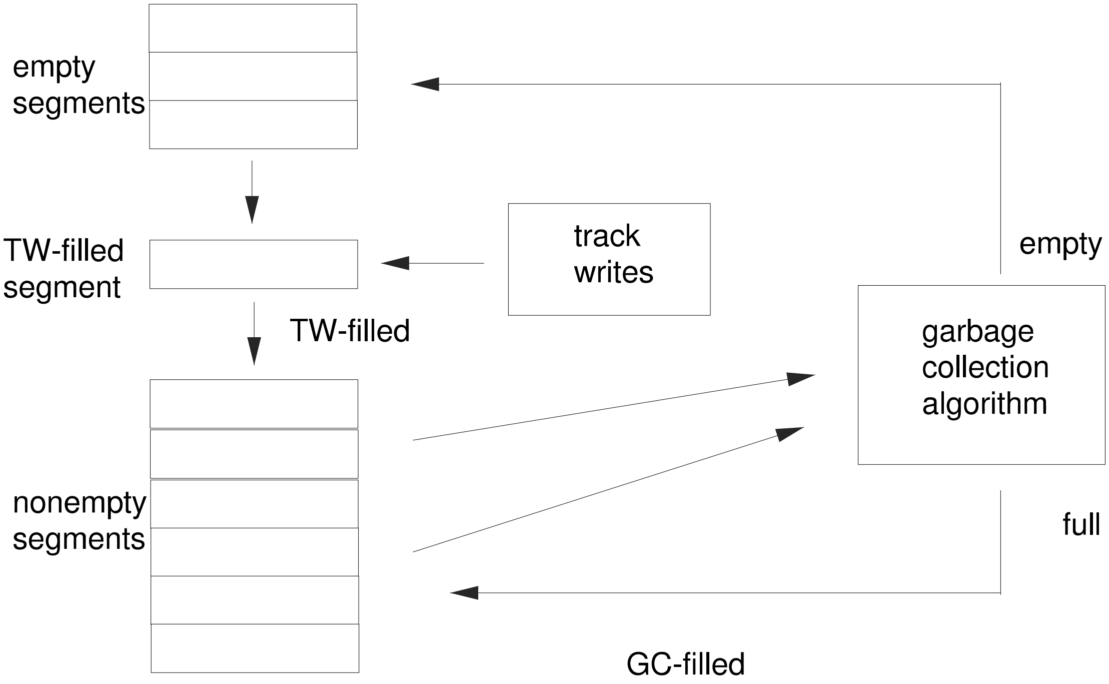

Figure 4: High level illustration of the separation model.

(call these *GC-filled segments*). The flow of segments in the separation model is illustrated in Figure 4.

During a phase of garbage collection (step S1), it is unlikely that the live tracks taken from the garbage collected segments will exactly fill a whole number of segments. To deal with this, our simulation keeps a set of *leftover tracks*; this set is initially empty. At any point, there are less than C leftover tracks. Whenever the garbage collector is invoked, it selects enough segments so that, if $M =$ max-empty, if N is the number of selected segments, and L is the total number of live tracks taken from the selected segments plus the live leftover tracks, then $N - \lfloor L/C \rfloor = M$. This ensures that the L live tracks can be packed into $N - M$ segments, creating M empty segments and $L \bmod C$ leftover tracks. Our basic simulation uses a simple procedure for the garbage collector to pack live tracks into segments: the leftover tracks are packed first, followed by the tracks taken from the selected segments in the order in which these segments were selected. Therefore, no complicated reorganization of the tracks is done. Later we investigate the effect of grouping tracks by age. (There are two alternatives to keeping leftover tracks, both of which have undesirable effects for our study. One alternative would be to create a partially filled segment. The undesirable effect is that this segment will often have artificially small utilization, so it would tend to make the GCU artificially small, especially in the case max-empty = 1 considered in the next section. A second alternative would be to finish filling the partially filled segment with newly-written tracks. This has the undesirable effect of violating the separation of newly-written tracks from garbage collected tracks. Both of these effects become less problematic as max-empty increases. For consistency, we also have leftover tracks in the case that max-empty is large.)

It is sometimes convenient to express max-empty as a fraction of the number S of segments. Define the *normalized max-empty* m by:

$$m = \text{normalized max-empty} = \text{max-empty}/S.$$

8 The Age-Threshold Algorithm

We now describe the basic age-threshold algorithm. The general rationale for the age-threshold algorithm is discussed in the Introduction. An important parameter in this algorithm is the *age-threshold*, which places a lower bound on the time that a segment must wait, after it is filled with track writing, before it can be selected by the garbage collector.

The age-threshold algorithm selects segments based on smallest utilization, although it considers a segment for selection only if its age exceeds the age-threshold. The age of a segment is defined as follows.

Definition of segment age. There is a *destage clock*, initially 0. Whenever a TW-filled segment s is filled by track writing (either initially or in step S2) the timestamp of s is set to the current value of the destage clock, and the destage clock is incremented by 1. Whenever a GC-filled segment s is produced, the timestamp of s is set to the largest timestamp of any of the segments that contributed tracks to s. At any point, the age of s is the difference between the current value of the destage clock and the timestamp of s. (Thus, the age of a GC-filled segment is initially set to the age of the youngest segment that contributed tracks to s.)

Note that once a segment s passes the age-threshold (i.e., its age exceeds the age-threshold), it will pass the age-threshold at all times in the future, until it is selected by the garbage collector. Note also that if s is filled with live tracks by the garbage collector, s immediately passes the age-threshold, since the segments that contributed tracks to s must have passed the age-threshold since they were selected by the garbage collector. In effect, only the TW-filled segments must wait to pass the age-threshold. Later we investigate how the results are affected by requiring all newly filled segments (both TW-filled and GC-filled) to wait before they can be selected for garbage collection.

Among the segments whose age exceeds the age-threshold, the garbage collector first selects segments with smallest utilization. As a tie-breaker among segments with the same (smallest) utilization, it first selects segments of largest age. (The rationale for giving preference to the older segments is that older segments tend to have fewer hot tracks than younger ones, so older segments have less potential for decreasing utilization in the future.) In the age-threshold algorithm, the age of a GC-filled segment has a secondary effect. In fact, in the more efficient implementation described in Section 13, the timestamp of a GC-filled segment can simply be set to zero. The age of a GC-filled segment is more relevant in one of the cost-benefit algorithms described later.

It is often useful to express the age-threshold as a fraction of the number S of segments. Define the *normalized age-threshold* t by:

$$t = \text{normalized age-threshold} = \text{age-threshold}/S.$$

9 The Separation Model with at Most One Empty Segment

We first consider the case max-empty = 1, where there is at most one empty segment (after the initial segment filling process is completed). Repeatedly and alternately, the garbage collector creates one empty segment, and this empty segment is filled by track writing. This models (in an extreme way) the situation where garbage collection and track writing are operating in parallel and in equilibrium. It is also an appropriate model to compare the separation model with the mixing model, since the mixing model selects one segment at a time and then fills this segment. Of course, it would be very difficult to maintain this tight equilibrium in practice, and we consider a more realistic case in the next section, where many empty segments are created during each phase of garbage collection.

9.1 Analysis

Since uniform track choice is a special case of hot-and-cold track choice, and since basic components of the analysis have already been introduced in Section 6, we proceed directly to the hot-and-cold case.

To analyze the separation model, we consider TW-filled and GC-filled segments separately: Think of these two types of segments as belonging to two "generations", with the TW-filled segments in generation 0 and the GC-filled segments in generation 1.[2] Each generation has a "beginning" where segments are added just after they are filled. The utilization of the segment then decreases until it reaches the "end" of the generation where the segment is removed by the garbage collector. It is reasonable to assume that, in steady state, the number of segments in each generation is preserved. With this assumption, we see that in one iteration of steps S1 and S2 in steady state, the garbage collector removes one segment from the end of generation 0 and some number n_1 of segments from the end of generation 1.[3] The live tracks in the selected segments are packed into n_1 (full) segments, which enter the beginning of generation 1. One new empty segment is created; this segment

[2] In this section, one generation of GC-filled segments suffices, although in the next section we use two generations of GC-filled segments.

[3] The garbage collection algorithm does not keep track of which generation each segment is in. Generations are introduced here only for the purpose of the analysis.

is filled by track writing and enters the beginning of generation 0. So the number of segments in each generation is preserved. Assuming that the age-threshold is large enough, the single segment selected from generation 0 is the segment that did not pass the age-threshold during the last phase of garbage collection, but does pass during the current phase since one more destage (step S2) has occurred in the interim. If the age-threshold is sufficiently large, the utilization of this segment will be sufficiently low for it to be selected.

We now introduce some parameters and use this description of the steady state to derive relationships among the parameters. First, recall that

$$t = \text{normalized age-threshold} = \text{age-threshold}/S.$$

Other parameters used are:

g_i = average utilization of segment removed by the garbage collector from the end of generation i

h_i = average fraction of segment containing hot tracks, for segments at the end of generation i

h'_i = average fraction of segment containing hot tracks, for segments at the beginning of generation i

n_i = average number of segments selected from generation i by the garbage collector during each phase of garbage collection (step S1)

Note that $h'_0 = p$ since the fraction of hot tracks in a TW-filled segment is initially p, and that $n_0 = 1$ since in steady state the garbage collector takes one segment from generation 0 and adds one empty segment to generation 0 during each phase.

The expression of certain equations is simplified by introducing two functions. For a segment initially having utilization 1 and having the fraction x of hot tracks, the function $\mathit{shrink}(x, y)$ is the expected utilization of the segment after ySC track writes. The function $\mathit{hot\text{-}shrink}(x, y)$ is similar except that it gives the expected fraction of the segment containing hot tracks after ySC track writes. By an argument in Section 6.2,

$$\begin{aligned} \mathit{shrink}(x, y) &= xe^{-yp/ha} + (1-x)e^{-y(1-p)/(1-h)a} \\ \mathit{hot\text{-}shrink}(x, y) &= xe^{-yp/ha}. \end{aligned}$$

Assuming for now that the age-threshold is large enough that a segment is selected from generation 0 just after it passes the age-threshold[4] it follows that generation 0 contains tS segments and generation 1 contains $(1-t)S$ segments. So tSC track writes occur between the time when a segment enters generation 0 and the time when it reaches the end. Therefore, g_0 and h_0 depend only on the "input" parameters a, h, p, t:

$$\begin{aligned} g_0 &= \mathit{shrink}(p, t) \\ h_0 &= \mathit{hot\text{-}shrink}(p, t). \end{aligned}$$

[4] We will see below how to detect when this assumption is not true and how to deal with it.

During each phase of garbage collection, g_0C live tracks are taken from a segment at the end of generation 0 and n_1g_1C lives tracks are taken from n_1 segments at the end of generation 1. Together, these live tracks must yield n_1C tracks to fill n_1 segments that enter the beginning of generation 1. A similar conservation argument applies to just the hot tracks. Therefore,

$$\begin{aligned} n_1 &= g_0 + n_1 g_1 \quad \text{or} \quad n_1 = g_0/(1-g_1) && (4) \\ n_1 h_1' &= h_0 + n_1 h_1. && (5) \end{aligned}$$

A segment that enters generation 1 stays there for $(1-t)S/n_1$ iterations of steps S1 and S2 (on the average), during which time $((1-t)/n_1)SC$ track writes occur. Therefore,

$$\begin{aligned} g_1 &= \mathit{shrink}(h_1', (1-t)/n_1) && (6) \\ h_1 &= \mathit{hot\text{-}shrink}(h_1', (1-t)/n_1). && (7) \end{aligned}$$

Since $\mathit{hot\text{-}shrink}(x, y)$ is linear in x, we can substitute (7) into (5) and solve for h_1' as a function of g_1. Substituting this expression for h_1' into (6) gives one equation in one unknown g_1.

The overall GCU g is the average of g_0 and g_1, weighted by the number of segments collected from each generation, that is,

$$g = \frac{g_0 + n_1 g_1}{1 + n_1} = \frac{g_0}{g_0 + 1 - g_1}.$$

All of this was done under the assumption that the age-threshold is sufficiently large that when the equation is solved we get $g_0 \leq g_1$. If $g_0 > g_1$, then the garbage collector would not select segments of age tS from generation 0, since their utilization is too large. Rather, segments selected from the two generations would have about the same utilization, namely, the lowest utilization. When solving the equation numerically, if the solution gives $g_0 > g_1$, we then find a t (larger than the given normalized age-threshold) that makes $g_0 = g_1$. In this case, we have one more unknown (t) and one more equation ($g_0 = g_1$).

9.2 Comparison with simulation

Figure 5 shows how GCU depends on the normalized age-threshold when GCU is obtained from the above analysis (dotted line) and from simulation (small circles). The first case has a degree of hotness $h = .1, p = .9$, while the second case has a lower degree of hotness $h = .1, p = .7$. ASU $= .8$ was used in both cases.

As is evident from both analysis and simulation, as age-threshold increases, GCU first stays constant, then decreases, and then increases. This makes sense intuitively. For a range of age-thresholds sufficiently near to 0, the age-threshold algorithm is essentially the same as the greedy algorithm, since the greedy algorithm will not select a segment based on smallest utilization until the age of the segment has passed the age-threshold anyway. At some value of age-threshold (the value at which GCU starts to decrease), the age-threshold

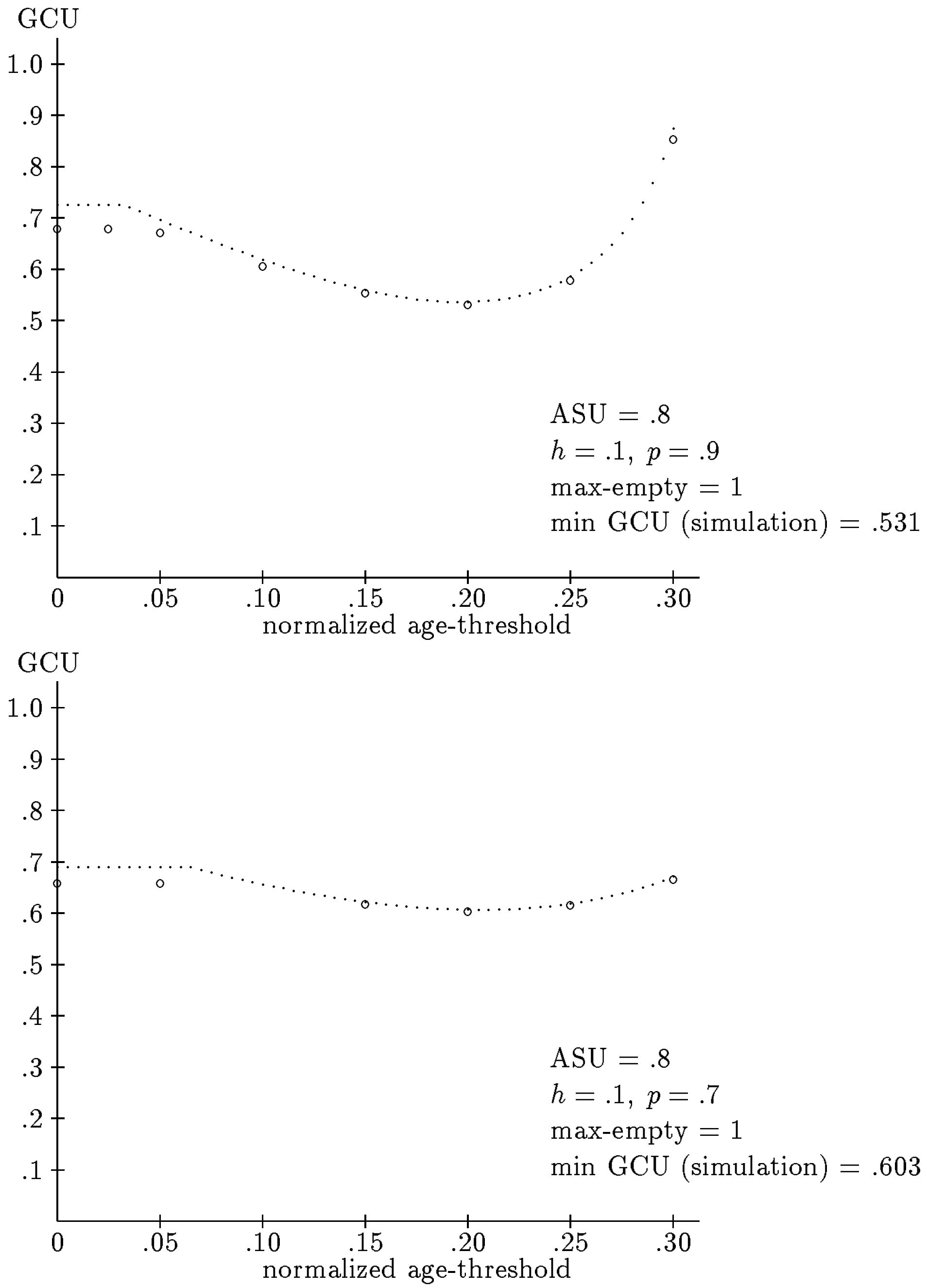

Figure 5: GCU vs normalized age-threshold for the age-threshold algorithm in the separation model with max-empty = 1. The dotted line was obtained from the analysis. Small circles are points obtained from simulation. Results are shown for two degrees of hotness, with the same ASU.

will "protect" a segment that the greedy algorithm would have selected. If the age-threshold is too small, however, the segments in generation 0 (that start with relatively many hot tracks) are collected too soon, before they have fulfilled their potential for rapidly decreasing utilization. But as the age-threshold continues to increase, we eventually reach a point of diminishing returns in generation 0 because too many low-utilization segments are being protected, forcing the algorithm to select higher-utilization segments from generation 1, and this causes GCU to increase.

Another fact that can be seen in the figure is that the change in GCU as a function of age-threshold is smaller at the lower degree of hotness. This also makes sense, since in the boundary case of uniform track choice, GCU does not depend on the age-threshold at all (unless the age-threshold is so large that the segments in generation 1 do have enough time to decrease their utilization, which will cause GCU to increase). With ASU $= .8$ and uniform track choice, the analysis gives GCU $= .629$ for all normalized age-thresholds up to .375. For larger age-thresholds, GCU increases. The GCU obtained from simulation in this case remained constant at .626 over a range of normalized age-thresholds between 0 and .3.

We also see from this figure that the minimum GCU value is 0.531 at the higher degree of hotness and 0.603 at the lower degree of hotness. That is, the higher the degree of hotness, the lower the value of the minimum GCU.

9.3 Choosing an age-threshold

Given the parameters ASU and degree of hotness (p, h), the analysis above can be used to find an "optimal" age-threshold, i.e., one that minimizes GCU. A difficulty is that, while the ASU of a system is not hard to measure, the degree of hotness of the data is more difficult to measure. The fact that GCU does not depend very much on the age-threshold at low degrees of hotness suggests a solution to this difficulty: Choose the age-threshold based on a high degree of hotness. If the degree of hotness is indeed high, our choice is close to the true optimal value. If the degree of hotness is low, then although our choice for the age-threshold might be far from the optimal value, it does not matter much since GCU is not very dependent on the age-threshold at low degrees of hotness.

To test this idea, we first found (from the analysis) that for ASU $= .8$, $h = .1$ and $p = .9$, the optimal value for the normalized age-threshold is $t = .2$ (to within $\pm .005$). Then for lower degrees of hotness, $h = .1$ and $p = .1, .3, .5, .7$, and $h = .3$ and $p = .3, .5, .7, .9$, we compared the GCU at the optimal t (optimal for the particular h and p) with the GCU at the fixed $t = .2$ (where GCU values were obtained from the analysis). In all cases, the two values were within 2% of each other.

9.4 Comparison of the mixing and separation models

As shown previously in Figure 3, GCU increases with increasing degree of hotness in the mixing model. In contrast, in the separation model, using the same ASU $= .8$ and using a fixed normalized age-threshold $t = .2$ suggested in Section 9.3, GCU first increases slightly and then decreases with increasing degree of hotness. This is shown in Figure 6, where

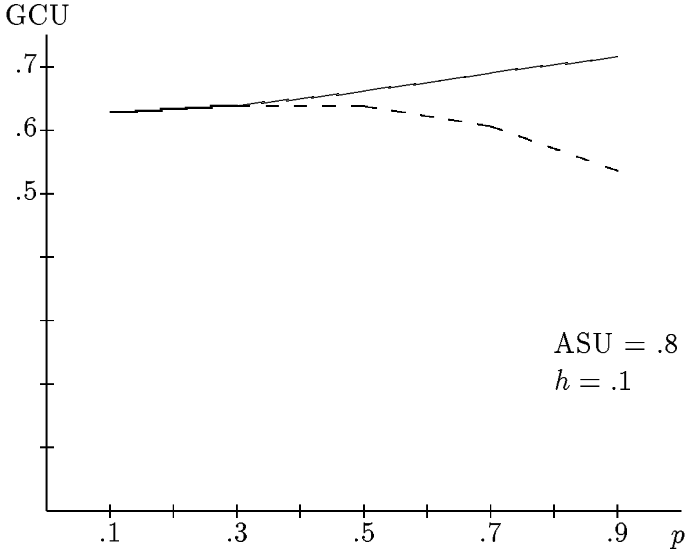

Figure 6: GCU vs degree of hotness (p) for the mixing model (solid line) and the separation model (dashed line) with max-empty = 1 and a fixed age-threshold ($t = .2$).

GCU values were obtained from analysis. For this particular case, we also see that the separation model is better than the mixing model, since it never has worse GCU and often has better GCU. We also looked at the results of analysis for a range of ASU's $(.4, .6, .7, .8, .9)$ and degrees of hotness $((h, p) = (.01, .99), (.05, .95), (.1, .9), (.2, .8))$. In all 20 cases, the age-threshold algorithm with the optimal age-threshold in the separation model has significantly better GCU than the greedy algorithm in the mixing model (and recall that an age-threshold does not improve GCU over that of the greedy algorithm in the mixing model). Even the greedy algorithm in the separation model never has significantly worse GCU (more than 2% higher) when compared to the greedy algorithm in the mixing model, over the same set of cases. From this it seems reasonable to conclude that the separation model is preferable to the mixing model, especially if a well-chosen age-threshold is used. This makes sense intuitively: segments that are completely filled with hot newly-written data have more potential for decreasing utilization than segments containing a mixture of hot newly-written data and cold garbage-collected data.

10 The Separation Model with Many Empty Segments

We now consider the age-threshold algorithm in the separation model where max-empty is viewed as a fraction of the segments, rather than a constant. Recall that

$$m = \text{normalized max-empty} = \text{max-empty}/S.$$

10.1 Analysis

We again use the parameters defined in Section 9.1. There are two differences in the analysis. The first difference is that, during each phase of garbage collection in steady state, mS segments are taken from generation 0, since mS initially empty segments are added to generation 0 during each phase. It is convenient to view the number n_i of segments removed from generation i as a fraction of the segments, rather than as an absolute number as before. So we define for each generation i,

$$\eta_i = n_i/S.$$

In particular, $\eta_0 = m$. A second difference is caused by the fact that, from among the (many) segments selected by the garbage collector at each phase, the ones selected from generation 0 typically have lower utilization than those selected from generation 1 (assuming that the age-threshold is well-chosen). Since the garbage collector selects segments based on lowest utilization, it will first select segments from generation 0. So the live tracks taken from generation 0 will be packed together into full segments, and the live tracks taken from generation 1 will be packed together (with a small overlap in one full segment that could contain live tracks from both generations). Since the two types of full segments could have different fractions of hot tracks initially, we introduce a second generation of GC-filled segments. Now the flow of live tracks is from the end of generation 0 to the beginning of generation 1, from the end of generation 1 to the beginning of generation 2, and from the end of generation 2 to the beginning of generation 2. We let s_i denote the fraction of segments in generation i:

$$s_i = (\text{average number of segments in generation } i)/S.$$

Of course,

$$s_0 + s_1 + s_2 = 1.$$

Assuming that the age-threshold is large enough, we also have $s_0 = t + m$. To see this, consider a time just before garbage collection is invoked. A fraction tS of segments in generation 0 have not passed the age-threshold, and another fraction mS have just passed since the last time that garbage collection was done. After this phase of garbage collection, mS segments are removed from the end of generation 0, and mS empty segments are added.

We now derive some equations among the parameters. We begin by finding expressions for g_0 and h_0, the average utilization and fraction of hot tracks in the segments removed from the end of generation 0. These are now averages over segments with ages between tS and $(t+m)S$:

$$\begin{aligned} h_0 &= \frac{1}{m}\int_t^{t+m} \textit{hot-shrink}(p, y)\, dy \\ &= \frac{ha}{m}\left(e^{-tp/ha} - e^{-(t+m)p/ha}\right), \end{aligned}$$

$$\begin{aligned} g_0 &= \frac{1}{m}\int_t^{t+m} \mathit{shrink}(p,y)\, dy \\ &= \frac{ha}{m}\left(e^{-tp/ha} - e^{-(t+m)p/ha}\right) + \frac{(1-h)a}{m}\left(e^{-t(1-p)/(1-h)a} - e^{-(t+m)(1-p)/(1-h)a}\right). \end{aligned}$$

Conservation of tracks and hot tracks moving from the end of generation 0 to the beginning of generation 1 gives

$$\begin{aligned} \eta_1 &= g_0\eta_0 \\ h_1'\eta_1 &= h_0\eta_0. \end{aligned}$$

Conservation of tracks and hot tracks moving from the end of generations 1 and 2 to the beginning of generation 2 gives

$$\begin{aligned} \eta_2 &= g_1\eta_1 + g_2\eta_2 \\ h_2'\eta_2 &= h_1\eta_1 + h_2\eta_2. \end{aligned}$$

A segment stays in generation i $(i = 1, 2)$ for an average of s_i/η_i phases, where mSC track writes occur in each phase. Therefore, for $i = 1, 2$,

$$\begin{aligned} g_i &= \mathit{shrink}(h_i', (s_i/\eta_i)m) \\ h_i &= \mathit{hot\text{-}shrink}(h_i', (s_i/\eta_i)m). \end{aligned}$$

Finally, the overall GCU is the weighted average of g_0, g_1, and g_2:

$$g = \frac{mg_0 + \eta_1 g_1 + \eta_2 g_2}{m + \eta_1 + \eta_2} = \frac{g_0 - g_0 g_2 + g_0 g_1}{1 - g_2 + g_0 - g_0 g_2 + g_0 g_1}.$$

Simplifying as before, these equations can be reduced to one equation in two unknowns, g_2 and s_2. We get another equation from the fact that the garbage collector, selecting segments based on lowest utilization, selects $\eta_1 S$ from generation 1 and $\eta_2 S$ from generation 2, where the utilization of segments selected from generations 1 and 2 must be roughly the same. Our first try at forcing this was to require $g_1 = g_2$. We got better results (closer to simulation) by requiring that the utilizations be balanced at the point where the garbage collector stops collecting from each generation. That is, we set $g_1' = g_2'$ where

$$g_i' = \mathit{shrink}(h_i', ((s_i - \eta_i)/\eta_i)m).$$

This is still consistent with $\eta_i S$ segments being selected from generation i $(i = 1, 2)$, and it allows $g_1 \neq g_2$.

As with the previous analysis, this analysis was done under the assumption that the age-threshold is large enough. As an indication that the age-threshold is large enough, we use $g_0' \leq g_1'$ where $g_0' = \mathit{shrink}(p, t)$, i.e., g_0' is the utilization of segments of age tS in generation 0, which is the point at which the garbage collector stops collecting from generation 0. If after solving the above equations we see that $g_0' > g_1'$, we then find another t (larger than the given t) to make $g_0' = g_1'$. This gives another equation and another unknown.

10.2 Comparison with simulation

Figure 7 shows how GCU depends on normalized age-threshold from analysis (dotted line) and simulation (small circles). In both cases shown in the figure, ASU $= .8$ and the degree of hotness is $h = .1$, $p = .9$. The two cases differ in the choice of max-empty, with $m = .05$ in the first case and $m = .01$ in the second case. Although the analysis GCU does not match the simulation GCU as closely as when max-empty $= 1$, it does a reasonable job of predicting the optimal age-threshold.

Comparing Figure 7 with the top half of Figure 5 (which has the same ASU, h and p), we see that the optimal normalized age-threshold decreases by about m when compared to the max-empty $= 1$ case. With max-empty $= 1$, the optimal t is .196 (from analysis, to within $\pm .001$). With $m = .05$ the optimal t is .145, and with $m = .01$ the optimal t is .186. This makes sense, since with a larger max-empty, the segments selected from generation 0 have ages in the range from tS to $(t+m)S$, instead of just tS which is the case when max-empty $= 1$.

We also plotted GCU against normalized age-threshold for a lower degree of hotness ($h = .1$, $p = .7$), and observed as in the max-empty $= 1$ case that the curve becomes flatter. We also observed that the analysis becomes more accurate at the lower degree of hotness. In the case of uniform track choice with ASU $= .8$ and $m = .05$, the analysis is very accurate, giving GCU $= .664$ versus GCU $= .661$ for simulation. As for choosing an age-threshold, we redid the exercise described in Section 9.3 with $m = .05$, and came to the same conclusion: an age-threshold chosen to be optimal at a high degree of hotness is also a reasonable choice at lower degrees of hotness.

Figure 8 shows the distribution of segment utilizations in the case ASU $= .8$, $h = .1$, $p = .9$, $m = .05$, and using $t = .145$ which is the optimal t from the analysis. For each utilization (horizontal axis) the height of the graph gives the fraction of segments having that utilization, where the segments are restricted to those that pass the age-threshold; that is, we restricted the segments to those available to the garbage collector. Each point is an average over many phases of garbage collection, and each contribution to the average was taken just before a phase of garbage collection starts. The distribution shows the desired "bimodal" distribution discussed in [10], where some collected segments have much smaller utilization than others. The part of the distribution at lower utilization contains roughly .05 of the segments (and recall that $m = .05$). These segments are the ones that did not pass the age-threshold during the last phase, but do pass now. The analysis does a reasonable job of predicting the location of the two parts of the distribution: solving the equations for the case shown in Figure 8 gives $g_0 = .23$ which is close to the center of the lower-utilization part, and gives $(\eta_1 g_1 + \eta_2 g_2)/(\eta_1 + \eta_2) = .80$ which is close to the left "edge" of the higher-utilization part, which is the part of this distribution where the garbage collector concentrates.

We did not attempt to improve the analysis by using more generations, for several reasons. First, what we wanted most from the analysis was some guidance in choosing an age-threshold, and it seems to be doing a reasonable job of this. Second, solution of the equations shows that the segments entering generation 2 have few hot tracks (h'_2 is small).

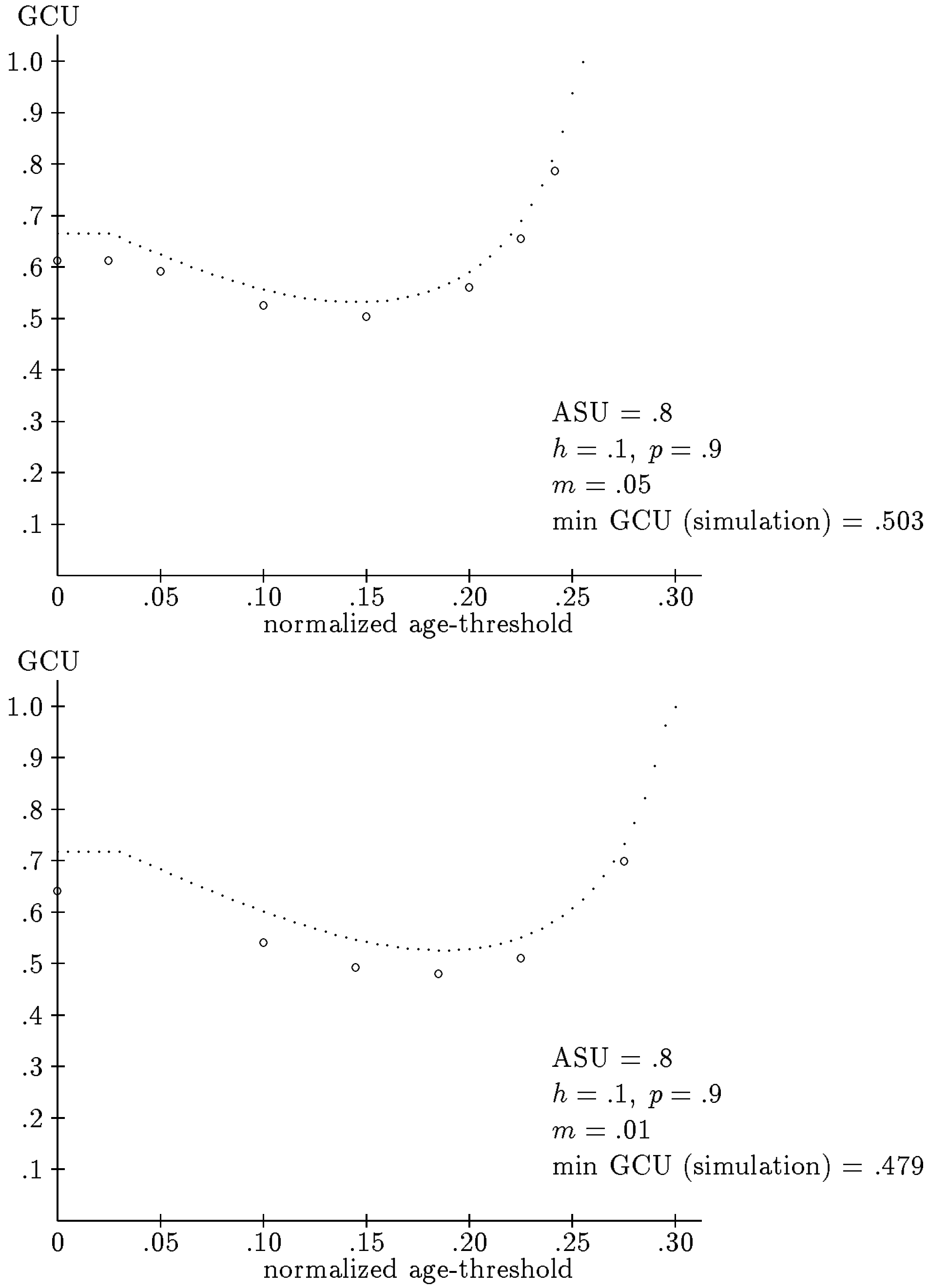

Figure 7: GCU vs normalized age-threshold for the age-threshold algorithm in the separation model with $m > 0$. The dotted line was obtained from the analysis. Small circles are points obtained from simulation. Results are shown for two m's, with the same ASU and degree of hotness.

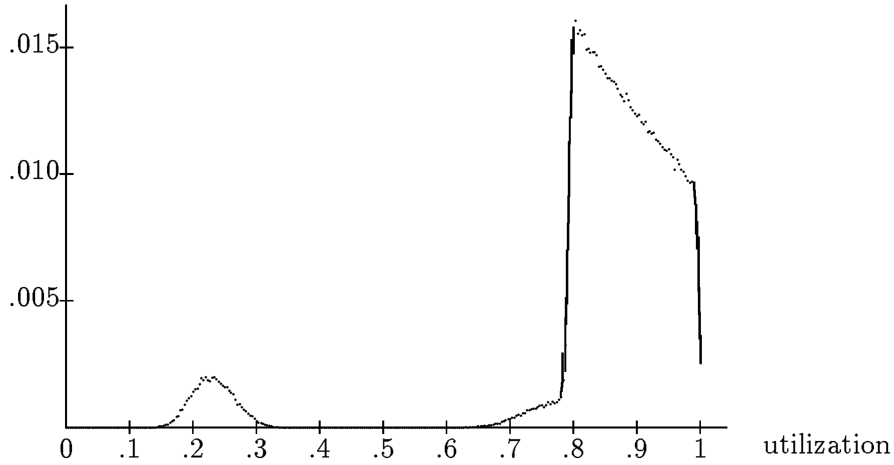

Figure 8: Distribution of segment utilization at the time when garbage collection is invoked, restricted to segments that pass the age-threshold. Parameters used are ASU $= .8$, $h = .1$, $p = .9$, $m = .05$, and $t = .145$.

This should be expected since tracks entering this generation have already spent time in generations 0 and 1 without being written. So segments entering generations 2, 3, etc., will all look pretty much the same, since the only thing that distinguishes one full segment from another (in the analysis) is the fraction of hot tracks in it. A third reason is that each additional generation introduces another unknown (the fraction of segments in the generation) which complicates solution of the equations.

11 Effect of Requiring All Segments to Wait

In the age-threshold algorithm considered previously, only the TW-filled segments must wait to pass the age-threshold, since the definition of segment age implies that a GC-filled segment passes the age-threshold immediately after it is filled. In this section we consider a modification where all newly-filled segments must wait before they can be selected for garbage collection. The rationale for making all segments wait is that, even though a GC-filled segment will probably contain fewer hot tracks than a TW-filled segment, a GC-filled segment could still contain a significant fraction of hot tracks, so letting the GC-filled segments age might be beneficial. (For example, in the case ASU $= .8$, $h = .1$, $p = .9$, $m = .05$, and $t = .145$, analysis predicts that a first-generation GC-filled segment contains a fraction .58 of hot tracks.) We investigated a simple version of this idea where all segments wait about the same amount of time.

A convenient way to make all segments wait about the same amount of time is to place each newly-filled segment (both TW-filled and GC-filled) into a FIFO queue, called the *waiting list*. Whenever the destage clock is incremented (which still happens only when a TW-filled segment is destaged) we look at the head of the queue to see if the segment there passes the age-threshold. If so, we remove segments from the head of the queue as long as they pass the age-threshold; these removed segments are now available to the garbage collector. Even though the age of GC-filled segments in the waiting list is large enough that they pass the age-threshold, they get stuck in the queue behind TW-filled segments that have not passed yet. In effect, for the purpose of passing the age-threshold, we give a GC-filled segment s the same "age" as the TW-filled segment that was filled most recently before s was filled. The age of a segment as defined in Section 8 is still used as a tie-breaker among segments having the same utilization, i.e., the oldest segment is selected. To distinguish this algorithm from the one considered previously, call this algorithm the *all-age-threshold* algorithm and call the previous algorithm the *TW-age-threshold* algorithm.

We first modify the analysis of Section 10.1 for the all-age-threshold algorithm. The main difference in the analysis is that a GC-filled segment in generation 1 waits for $k = \lceil t/m \rceil$ phases (iterations of steps S1 and S2) before it is removed by the garbage collector. During this time, $kmCS$ track writes occur. Therefore,

$$\begin{aligned} g_1 &= \mathit{shrink}(h'_1, \lceil t/m \rceil m) \\ h_1 &= \mathit{hot\text{-}shrink}(h'_1, \lceil t/m \rceil m). \end{aligned}$$

The fraction of segments in generation 1 is now $s_1 = \lceil t/m \rceil \eta_1$. The rest of the analysis is the same. Now, s_1 is not an unknown that must be solved for (there is only one unknown g_2), provided that the age-threshold is large enough that $g'_0 \leq g'_2$ and $g'_1 \leq g'_2$. If these two inequalities hold, we can ignore that segments in generation 2 also pass through the waiting list, because their utilization does not drop to a level that would cause them to be selected until after they leave the waiting list. Since we are concerned mainly with the case that the age-threshold is large enough that these inequalities hold, we did not extend the analysis here to the case that they do not hold.

Another condition that should be met is that the number of second-generation segments in the waiting list does not exceed the total number of second-generation segments, that is, $\lceil t/m \rceil \eta_2 \leq s_2$. We found that this inequality can fail if the age-threshold is chosen too large. What this means in reality is that the garbage collector would not be able to collect enough free space from segments outside the waiting list, so the garbage collector would be stuck. One way to avoid this disaster in practice is to have the garbage collector take segments from the head of the waiting list if there are no segments outside the waiting list. This rule was used in our simulations due to its simplicity and failsafe property.

Figure 9 is the analogue of Figure 7 for the all-age-threshold algorithm, showing analysis versus simulation for the same two choices of parameters. For ASU $= .8$, $h = .1$, $p = .9$, and $m = .05$, the best GCU found for the TW-age-threshold algorithm was .503 (at $t = .145$). The all-age-threshold algorithm gave about a 9% improvement in GCU to .457 (at $t = .14$). At a smaller $m = .01$, the improvement was about 11% from .479 (at $t = .185$)

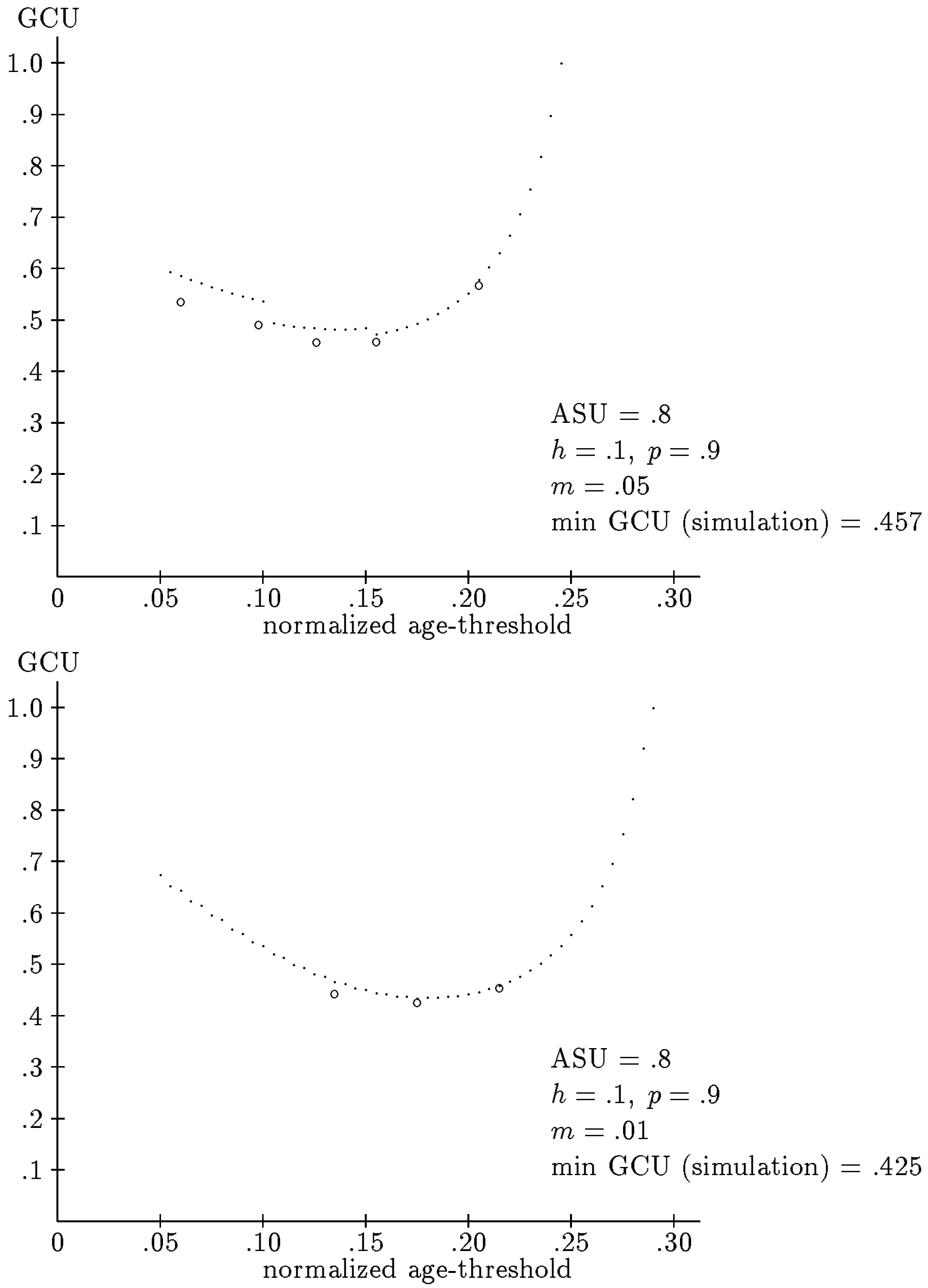

Figure 9: GCU vs normalized age-threshold for the all-age-threshold algorithm in the separation model with $m > 0$.

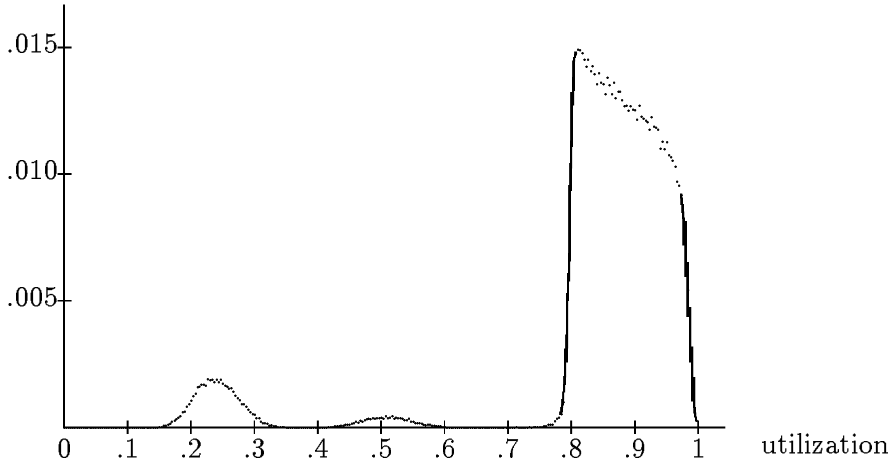

Figure 10: For the all-age-threshold algorithm, distribution of segment utilization at the time when garbage collection is invoked, restricted to segments outside the waiting list. Parameters used are ASU $= .8$, $h = .1$, $p = .9$, $m = .05$, and $t = .13$.

to .425 (at $t = .175$). However, as described later in the paper, in simulations on another type of synthetic trace and on an actual I/O trace, the TW-age and all-age versions have essentially identical GCU for all reasonably small values of the age-threshold (in particular, at the optimal value); the improvement of all-age over TW-age is not evident until the age-threshold is increased substantially above the optimal value.

Since the GC-filled segments initially contain fewer hot tracks than the TW-filled segments, possibly we could decrease GCU further by using two different age-thresholds, one for the TW-filled segments and one for the GC-filled segments. We investigated this possibility by simulation in one case, ASU $= .8$, $h = .1$, $p = .9$, $m = .01$, considered above. With a single age-threshold, we found that .175 is a good normalized age-threshold, giving GCU $= .425$. Keeping this age-threshold for the TW-filled segments, we tried varying the second age-threshold used for the GC-filled segments. Decreasing the second normalized age-threshold t_2 to .1 caused GCU to increase to .443. Increasing t_2 to .25 caused GCU to decrease slightly to .421, but increasing it further to .3 caused GCU to increase to .424. This suggests that the improvement in GCU from using two different age-thresholds is not worth the overhead of implementing it and the complications of choosing two different age-thresholds.

Figure 10 is the analogue of Figure 8, showing the distribution of segment utilizations at the point where garbage collection is invoked, but restricted to segments outside the waiting list. For the all-age-threshold algorithm, there is a “trimodal” distribution. The likely reason is that the middle part of the distribution (around utilization .5) contains GC-filled segments that have just exited from the waiting list, since the last time that garbage collection was done. For the hot-and-cold model, the existence of this middle part shows that requiring GC-filled segments to wait does cause the utilization of some of them to drop to a lower point, i.e., lower than if these segments were immediately made available to the garbage collector. The analysis does a reasonable job of predicting the location of the three parts of the distribution. For the parameters used in Figure 10, the analysis described above in this section gives $g_0 = .26$ (close to the center of the leftmost part), $g_1 = .49$ (close to the center of the middle part), and $g_2 = .81$ (close to the left edge of the rightmost part).

12 Comparison with Cost-Benefit Selection Criterion

In the paper that introduced log-structured file systems [10], a different criterion, the *cost-benefit criterion*, is used to decide which segments to select for garbage collection. In this section, we compare the age-threshold criterion with the cost-benefit criterion. The two criteria have in common that they use the utilization and the age of a segment to order the segments, but they do the ordering in different ways. The age-threshold method keeps a segment out of the ordering until it passes the age-threshold; the segments that do pass are ordered first by utilization (with smaller being better) and second by age (with older being better). The cost-benefit method orders the segments by their benefit/cost ratio (with larger being better): if a segment has age A and utilization u, its benefit/cost ratio is

$$\frac{\text{benefit}}{\text{cost}} = \frac{(1-u)A}{1+u}.$$

A segment is good to select if a large fraction of the segment is dead and if the segment is old. The benefit (numerator) is taken to be the product of the dead fraction $(1-u)$ and the age A. The denominator represents the cost of selecting the segment, since a whole segment is read and a fraction u (the live tracks) is written back to disk.

[10] uses a definition of segment age different than the one we use: the age of a segment is the age of the youngest track in the segment, where the age of a track is the elapsed time since the track was last modified. For simulation, we take the age of a track k to be the number of track writes that have occurred since k was last written. To distinguish the two definitions, call this definition the *track-based* definition, and call our definition given in Section 8 the *segment-based* definition. In the segment-based definition the age of a segment depends on the timestamp of the segment, whereas in the track-based definition the age depends on timestamps of tracks in the segment.

[10] also suggests that the garbage collector group the collected live tracks by age before packing them into segments, the idea being that tracks last written at around the same time have some temporal affinity, so it could be advantageous to have them together in the same segment.

Segment-Age	Selection-Criterion	Age-Grouping?	
		Yes	No
segment	greedy	.610	.612
track	cost-benefit	.688	.535
track2	cost-benefit	.580	.610
segment	cost-benefit	.528	.526
segment	TW-age-threshold	.501	.503
segment	all-age-threshold	.457	.457

Figure 11: GCU values obtained by simulation for various garbage collection algorithms, where ASU $= .8$, $h = .1$, $p = .9$, and $m = .05$.

12.1 Comparison of GCU

By simulation, we compared the cost-benefit selection criterion using both the segment-based and the track-based definition of segment age, the age-threshold criterion with the segment-based definition, and the greedy algorithm. For the age-threshold criterion, both the TW-age-threshold and the all-age-threshold versions were considered. In each case, the simulation was done both with and without age-grouping. Initial simulations were done with ASU $= .8$, degree of hotness $h = .1$ and $p = .9$, and max-empty $= .05$ of the segments. The age-thresholds used for the age-threshold algorithms were the optimal values obtained from the analysis. When age-grouping was in effect, the tracks were sorted by age in batches of about 3000, representing 10 segments worth of tracks in our simulations (where segment capacity is $C = 300$).

The GCU number obtained for the cost-benefit criterion, with track-based segment age and age-grouping, was not very good (.688). Some segments were being selected at very high utilization. The likely reason is that these segments were very old, so that their benefit/cost was larger than younger lower-utilization segments. This problem was just exacerbated by age-grouping. To attempt to alleviate this problem, we also considered a modified version of the track-based definition of segment age, where the age of a segment is reset to zero when any live track in the segment is written, even though this write causes the track to become dead in that segment. Call this modified definition the *track2-based* definition.

Figure 11 shows the GCU values obtained in each case. Age-grouping had little effect in the cases using the segment-based definition of segment age. The age-threshold algorithms gave the best GCU. Since the performance of these algorithms depends on a good choice for the age-threshold, it is also worth mentioning that the TW-version is still better than the next better competitor (.526 for the cost-benefit criterion and segment-based age) for a range of normalized age-thresholds between roughly .1 and .175. The all-age version has GCU better than .526 for a larger range of age-thresholds between roughly .07 and .19.

Simulations were also done at a lower ASU $= .75$ and a lower max-empty $= .025$ of the segments, with the same degree of hotness. For each of the cost-benefit algorithms, we only

Segment-Age	Selection-Criterion	Age-Grouping?	GCU
segment	greedy	no	.551
track	cost-benefit	no	.424
track2	cost-benefit	yes	.420
segment	cost-benefit	no	.420
segment	TW-age-threshold	no	.381
segment	all-age-threshold	no	.353

Figure 12: GCU values obtained by simulation for various garbage collection algorithms, where ASU $= .75$, $h = .1$, $p = .9$, and $m = .025$.

did the simulation for the choice of age-grouping (yes or no) that gave the best result in the previous experiment. Again, optimal (from analysis) age-thresholds were used for the age-threshold algorithms. These GCU values are shown in Figure 12.

Although not a definitive study, the results indicate at least that the age-threshold algorithm is competitive with algorithms using the cost-benefit selection criterion. Moreover, for a range of age-thresholds, the age-threshold algorithm is better.

12.2 Example scenarios to illustrate differences between algorithms

A scenario where the age-threshold algorithm outperforms the cost-benefit algorithm is one like the hot-and-cold model, where very old segments containing only (or almost only) cold tracks become emptier very slowly as cold tracks are rewritten. Before the utilization of these segments has decreased very far, their age becomes large, causing their benefit/cost ratio to become large enough for them to be the best choice for selection, even though they have high utilization. In the case of track-based segment age, age-grouping just exacerbates this effect.

On the other hand, a reason to garbage collect very-old high-utilization segments is to reclaim the free space in these segments. This consideration suggests a scenario where cost-benefit outperforms age-threshold (this was verified by simulation): 90% of the tracks (the "frozen" tracks) are *never* rewritten; 10% of the tracks are written uniformly; initially each full segment contains 90% frozen tracks; ASU $= .9$; and $m = .01$. Cost-benefit does better than age-threshold at reorganizing the frozen tracks into full segments, so it has better GCU after the reorganization is done. To handle this situation, the age-threshold algorithm could be augmented by another process that collects free space from very-old high-utilization segments. Since this process has relatively high overhead and relatively low priority, it could be done during idle periods. In the context of log-structured file systems, there is existing work on scheduling garbage collection during idle periods [1].

12.3 Implementation issues for scalable designs

In our opinion, the cost-benefit algorithm is more difficult to implement and consumes more resources than the age-threshold algorithm, provided that a list of the segments (actually, segment names) is to be maintained with the segments sorted according to their desirability for garbage collection. To implement a garbage collection algorithm, one of the following needs to be done: (1) keep all segments sorted according to their desirability for garbage collection (e.g., by the cost-benefit criterion in the cost-benefit algorithm), so when segments are needed for garbage collection, they can be easily found; or (2) sort all segments according to desirability whenever we need to do garbage collection. [10] proposes method (2). However, method (2) may require significant CPU resources for large systems with an extremely large number of segments. This means that method (2) is not scalable, since we may get reasonable performance in small systems, but poorer performance in larger systems when the sort may take considerable resources. Remember that, in many controllers, the CPU resource in the controller is often the bottleneck to performance; method (2) makes this situation worse. Furthermore, the sorting would have to be interruptible by other activities so as not to produce a noticeable degradation of performance while the sorting is being done, and this could complicate the design of the controller in an LSA system. Therefore, we may want to consider method (1), where the segments are always maintained in sorted order. However, it is difficult to maintain segments in sorted order by the cost-benefit criterion. Because of the way this criterion is defined, it is possible that the positions of two segments in the list may need to be reversed even if there was no activity to either of those two segments (or to any other segment), just because the two segments each got a little older. For the age-threshold algorithm, on the other hand, a segment changes position in the list only if some live track in the segment is written, causing its utilization to decrease. In the next section, we show some very attractive implementations of the age-threshold algorithm.

13 More Efficient Implementations of Age-Threshold Algorithms

Using the definition of the age-threshold algorithm given in Section 8, the garbage collector needs an ordered list of the segments available to it (i.e., those outside the waiting list) where the segments are sorted by decreasing utilization, and within each sublist containing segments with the same utilization the segments are sorted by increasing age. To distinguish this algorithm from the variation described below, call it the *full-sort* algorithm. Suppose that we want to maintain this ordering as track writing occurs (so that whenever the garbage collector needs to select a segment, it just has to remove the segment at the lowest-utilization end of the list). For any two segments in the list, during any period of time in which no live track in either segment is written, the two segments will not change their relative order since their utilizations do not change and their ages increase at the same rate. But whenever a live track in segment s is written, segment s may change its position in the list because its

	TW-age	all-age
full-sort	.503	.457
bucket-sort	.471	.452

$m = .05$

	TW-age	all-age
full-sort	.479	.425
bucket-sort	.450	.425

$m = .01$

Figure 13: GCU values for the full-sort and bucket-sort versions of the TW-age-threshold and all-age-threshold algorithms, with ASU = .8, $h = .1$, and $p = .9$. 10 buckets were used for the bucket-sort algorithm.

utilization has decreased. Changing a segment's position involves considerable overhead.

One way to decrease the number of repositionings in the list is to use coarse-grained accuracy for the utilizations. The range of possible utilizations is divided into intervals of the same length. All segments having utilizations in the same interval are placed together in the same "bucket". If there are b buckets, then the i-th bucket $(1 \leq i \leq b)$ contains segments whose utilization u lies in the range $(i-1)/b < u \leq i/b$. For example, we could have 10 buckets, one bucket containing segments whose utilizations lie between 0 and 0.1, a second bucket for segments whose utilizations are between 0.1 and 0.2, and so on. The segments in each bucket are organized in a list. In the *bucket-sort* algorithm, each list is a FIFO queue. The age-threshold is implemented as in Section 11 by a *waiting list*, which is a FIFO queue. Each FIFO queue has a "tail" where segments (actually, segment names) enter, and a "head" where segments are removed.

The bucket-sort algorithm has low overhead and is easy to implement. After a TW-filled segment is filled with newly-written tracks, it enters the tail of the waiting list. Segments at the head of the waiting list that pass the age-threshold are removed and enter the appropriate bucket list depending on their utilization. In the TW-age-threshold version, a GC-filled segment enters the highest-utilization bucket. In the all-age-threshold version, a GC-filled segment enters the waiting list. If segment s is outside the waiting list and a write to a live track in segment s causes the utilization of s to cross an interval boundary, s is removed from its current bucket and enters the next lower one. Whenever the garbage collector needs a segment, it finds the lowest-numbered non-empty bucket, and removes the segment from the head of that list. If all buckets are empty, it removes the segment from the head of the waiting list. We used a separate bucket for segments of utilization 1 that have passed the age-threshold; the garbage collector never takes segments from this bucket, because there would be no point in collecting a segment that has no free space.

We simulated these algorithms with parameters ASU = .8, degree of hotness $h = .1$ and $p = .9$, and two values of m, .05 and .01. Both the TW-age-threshold and all-age-threshold algorithms were simulated. 10 buckets were used for the bucket-sort algorithm. The results are shown in Figure 13. In each case, we attempted to locate a good age-threshold, guided by the analysis. In all cases except one, the same age-threshold was used to obtain both GCU values in the same column. The exception was the column corresponding to $m = .05$ and the

TW-age-threshold algorithm, where the age-threshold used for the bucket-sort algorithm was slightly smaller than the one used for the full-sort algorithm. Two conclusions from these results are: (1) the all-age version does better than the TW-age version in all cases examined; and (2) for both the TW-age and all-age versions, we do not pay a price in increased GCU by using the more efficient bucket-sort algorithm with 10 buckets, instead of the full-sort algorithm.

In fact, one surprising result is that, in the TW-age case, bucket-sort actually gives better GCU than full-sort. Before running the simulations, we thought that the course-grained accuracy would cause the GCU to increase somewhat, but it actually decreased somewhat. A possible explanation for the smaller GCU obtained from the bucket-sort algorithm, compared with the full-sort algorithm, is that the former algorithm allows some extra aging of the GC-filled segments. As noted in Section 11, allowing GC-filled segments to age can be beneficial. For the bucket-sort algorithm, whenever a segment changes bucket it moves to the tail of its new queue so it has to move through the entire queue before being selected. A segment having a significant fraction of hot tracks will change buckets several times and so will incur some "queueing delay" for each change. The improvement of bucket-sort over full-sort is much smaller or nonexistent in the all-age case. This is consistent with the reasoning above: if GC-filled segments are being made to wait in the waiting list, then having them wait more in the buckets has less of an effect.

In two cases we also investigated the effect of changing the number of buckets, using the same age-threshold. The same ASU, h, and p as above was used, with $m = .01$. For the TW-age-threshold algorithm, decreasing the number of buckets from 10 to 5 had no significant effect on GCU (to 3 decimal places GCU did not change). Increasing the number of buckets from 10 to 30 caused GCU to increase from .450 to .472, which is approaching the GCU .479 of the full-sort algorithm in this case. For the all-age-threshold algorithm, changing the number of buckets to 5 or 30 caused no significant change in GCU.

14 Effect of Changing the Hot Tracks

For GCU values reported up to this point, values were obtained by letting the simulation reach steady state without changing the hot tracks. In some cases, we also tried changing the hot tracks after steady state was reached. After the hot tracks were changed, GCU was tracked by computing the average GCU over each phase of garbage collection. Figure 14 shows how GCU varied over time (number of track writes) after all hot tracks were changed to new randomly chosen tracks, all different than the original hot tracks. Results are shown for three algorithms: the cost-benefit algorithm, the bucket-sort TW-age-threshold algorithm, and the bucket-sort all-age-threshold algorithm. Segment-based segment age was used for all three, and 10 buckets were used for the age-threshold algorithms. Age-grouping was not done. Parameters were ASU = .8, $h = .1$, $p = .9$, and $m = .05$. In all three cases, GCU eventually reconverged to the original steady-state value. If only a fraction 20% of the hot tracks were changed, there was a similar effect, although the deviation from the original steady-state value was smaller. For example, for the all-age-threshold algorithm,

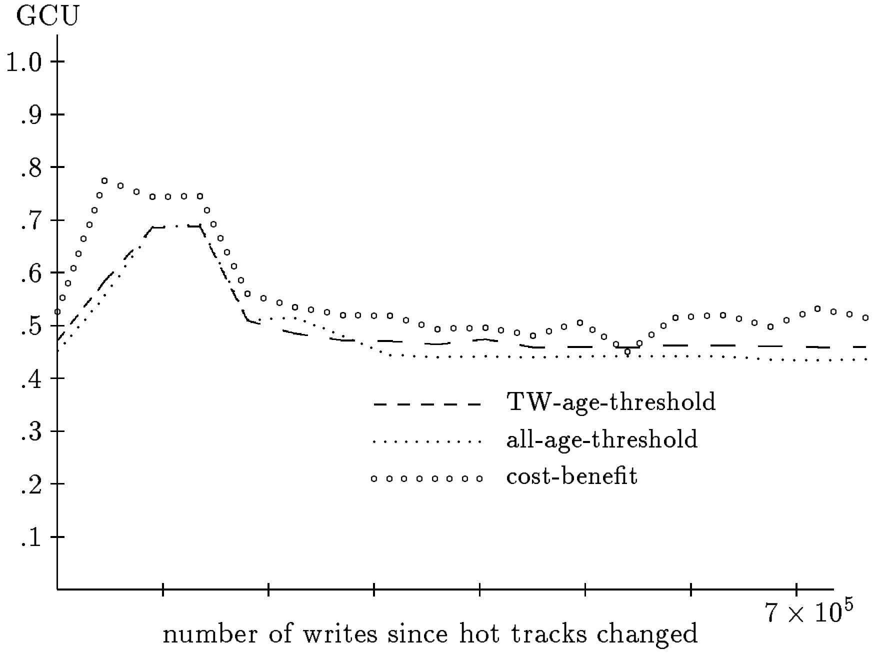

Figure 14: Change in GCU when all hot tracks are changed, for the cost-benefit algorithm, the bucket-sort TW-age-threshold algorithm, and the bucket-sort all-age-threshold algorithm. Segment-based segment age was used for all three cases, and age-grouping was not done. Parameters are ASU = .8, $h = .1$, $p = .9$, $m = .05$.

GCU increased from .452 to .55 when 20% of the hot tracks were changed, compared to an increase to .69 when all hot tracks were changed. Based on these experiments, none of the algorithms seems especially better or worse than the others in its reaction to a change in the hot tracks.

There was a different effect in the mixing model. Here GCU initially decreased by a small amount (about 3%) and then slowly reconverged to the original steady state value.

15 Conclusions for Hot-and-Cold Track Writing

We summarize our results under the hot-and-cold model of track writing. First, we have given an analysis that can yield an optimal age-threshold to use, given the ASU and the degree of hotness. We also suggested how to select a good age-threshold to use when the degree of hotness is not known; the basic idea is to assume a high degree of hotness. The results also show that if we are not sure what age-threshold to use, it is better to use one that is too small than to use one that is too large. Another result is that an age-threshold has an advantage over the greedy method in the separation model, but not in the mixing model; and the separation model is superior to the mixing model if the age-threshold is reasonably chosen. We found that the version of the age-threshold algorithm where all segments had to wait (all-age-threshold) was superior to one where only the TW-filled segments had to wait (TW-age-threshold). We found that there was no loss in performance in using the bucketized version of the algorithm. Finally, we showed that the age-threshold algorithm has better performance than the cost-benefit algorithm over a range of age-thresholds.

16 Simulations Using a Power Law Trace Model

One way of characterizing temporal locality in a trace is the power law for cache miss ratio as observed, for example, by Chow [2]. This states that the miss ratio M is related to the cache size Z by $M = cZ^{-\alpha}$ for some positive constants c and α. This power law has also been derived analytically from other empirically-observed properties of traces; see, for example, Thiebaut [13] and McNutt [6]. Further discussion of this can be found in [12].

A synthetic trace can be generated based on the power law as follows. Maintain an LRU list of the tracks, $r_1, r_2, \ldots, r_T$, where r_i is the i-th most recently written track. The next written track k is chosen according to the distribution

$$\Pr[\, k = r_i \text{ for some } i \geq x \,] = x^{-\alpha}.$$

Thus, younger tracks are more likely to be chosen than older tracks, and the set of "hot" tracks continually changes. (We take $c = 1$ because when $x = 1$ the probability should be 1.) If $i > T$ results, this choice is discarded and we try again until $1 \leq i \leq T$.

For the sake of efficiency, the size of the simulation was reduced to $S = 500$ segments of capacity $C = 50$, so $T = 20000$ for ASU = 0.8. Also for efficiency, we did not maintain the LRU list exactly, but used an approximation of it: if the chosen track is r_i where $2000 < i \leq T$, then the choice is redone to choose a track uniformly from $r_{2001}, \ldots, r_T$. Essentially, we view the youngest 2000 tracks (10% of the tracks) as the "hot" tracks which follow a power law distribution, and the remainder as "cold" tracks which follow a uniform distribution. (Unlike the hot-and-cold model used in previous sections, the set of hot tracks continually changes, and the distribution used for the hot tracks is not uniform.) We used $\alpha = 1/3$, which is a value suggested by McNutt [6] based on empirical studies of actual I/O traces. The max-empty fraction $m = .05$ was used. To better simulate the situation that track writing and garbage collection can occur concurrently, during each phase of garbage collection (started whenever the number of empty segments reached zero) the simulation alternately created 10 new empty segments and filled 5 empty segments until the number of empty segments reached max-empty.

In traces generated by the power-law model, it often happens that there are many writes to the same young track over a short interval of the trace. Therefore, it often happens that the track k being written is already in the segment being filled (the controller memory segment in LSA). For this reason, an LRU write cache was employed in the simulation. A track is placed in the segment being filled only when it is pushed out of the write cache. A write cache of size 50 (the segment capacity) was used. (For the power law model described above, the write cache is simply the youngest 50 tracks in the LRU list used to implement

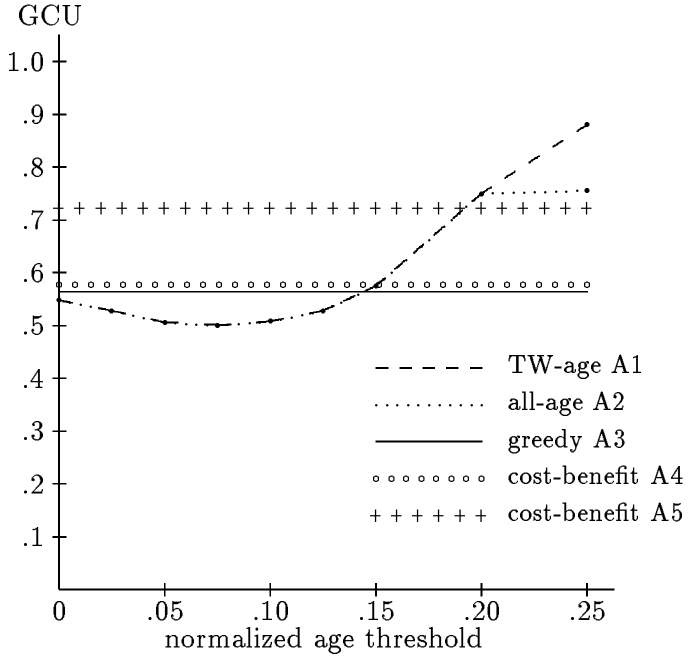

Figure 15: Results of simulation on a power law trace with ASU = .8 and $m = .05$.

the power law distribution. Whenever the chosen track is r_i for $i > 50$, the track that moves from r_{50} to r_{51} is added to the segment being filled.) In the hot-and-cold model considered previously, there are enough hot tracks that it is relatively unlikely that the same track is written twice over a short time interval, so using a write cache there would have little effect on the results.

The following five algorithms were simulated:

A1. the bucket-sort TW-age-threshold algorithm with 10 buckets;

A2. the bucket-sort all-age-threshold algorithm with 10 buckets;

A3. the greedy algorithm;

A4. the cost-benefit algorithm with segment-based segment age and no age-grouping;

A5. the cost-benefit algorithm as in [10] using track-based segment age and age-grouping.

Results are shown in Figure 15. Running algorithm A5 without age-grouping caused its GCU to improve from .723 to .584. Two differences can be noted between these results and the results obtained above for the hot-and-cold model with $h = .1$ and $p = .9$. First, the optimal normalized age-threshold is smaller, .075 versus about .15 for the hot-and-cold model. Second, the TW-age and all-age algorithms have virtually identical GCU for reasonably small values of age-threshold, versus all-age having smaller GCU for the hot-and-cold model. The likely reason for both differences is that young segments empty more quickly using the power law trace. Thus, a smaller age-threshold suffices, and GC-filled segments have less "hot" data in them. An explanation for why all-age has better GCU than TW-age at the large normalized age-threshold .25 is that all-age is forced to take segments from the waiting list more often than TW-age at this large age-threshold, thus effectively lowering the age-threshold used by all-age. Therefore, all-age is somewhat self-correcting when given a too-large age-threshold, giving another reason to prefer all-age over TW-age.

17 Simulations Using An Actual I/O Trace

In this section, we compare garbage collection algorithms using an I/O trace collected from a running system. We used the "snake" trace collected by Ruemmler and Wilkes [11]. The trace was collected on a 3GB file server over a period of two months. It contains about 6.8 million writes (only the writes are relevant to the GCU of a garbage collection algorithm). Further information about the trace can be found in [11]. Each write I/O was converted to a sequence of one or more track writes depending on the disk, starting address, and length of the I/O. Track size was taken to be 32KB. This yielded a total of $T = 98224$ tracks in the system. Of these, 36607 tracks were written (at least once) in the trace. To achieve ASU $= .8$, the simulations were done with $S = 2455$ segments of capacity $C = 50$. Max-empty was 122, i.e., 5% of the segments. As in the previous section, concurrent track writing and garbage collection was simulated by alternately creating 10 new empty segments and filling 5 empty segments, and an LRU write cache was used in the simulation with cache size equal to one segment's worth of tracks (50). We allowed the system to warm-up from its initial configuration for 1 million writes, and then calculated GCU over the remaining 5.8 million writes. (GCU was also computed over the entire trace; these GCU numbers were slightly lower but generally followed the same pattern as those computed only for the last 5.8 million writes.)

The same five algorithms A1–A5 as in the previous section were simulated. For cost-benefit algorithm A5, the I/O start times were used to determine the age of a track. Results are shown in Figure 16. The age-threshold algorithms A1 and A2 at normalized age-threshold $t = .075$ had the best GCU (.130), followed by greedy A3 (.175), cost-benefit A4 (.203), and cost-benefit A5 (.233). Without age-grouping, the GCU of cost-benefit algorithm A5 improved to .212. These results are consistent with our previous findings. Increasing the age-threshold from zero causes GCU to improve until an "optimal" value of age-threshold is reached, after which GCU increases. For a reasonably chosen age-threshold, the age-threshold algorithm has better GCU than the cost-benefit algorithm. Comparing TW-age with all-age, the result here is similar to that of the power law trace: all-age and TW-age have virtually identical GCU for reasonably small values of the age-threshold, but all-age is better at a large age-threshold.

Another experiment was done where the system contains only the tracks that are actually

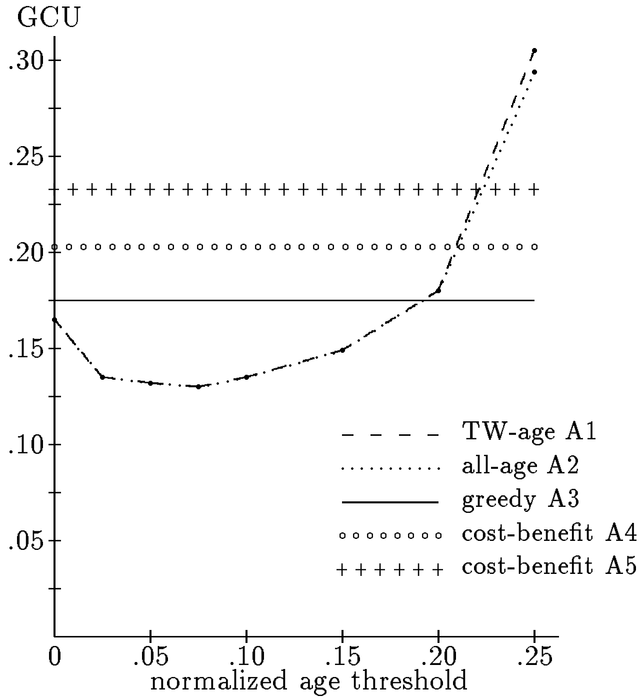

Figure 16: Results of simulations on the snake trace with ASU = .8 and m = .05.

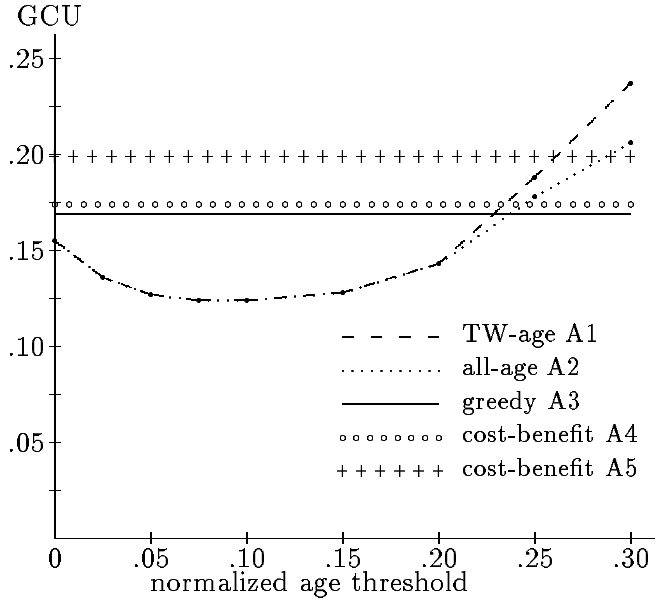

Figure 17: Results of simulations on the snake trace, restricted to the tracks that are written in the trace, with ASU = .8 and m = .05.

written, and each written track is loaded the first time it is written (unlike the previous experiment where all tracks, including those that are never written in the trace, are loaded in sequential order at the start of the simulation). Since 36607 tracks are written, we took $C = 50$ and $S = 915$ to achieve ASU = .8. The results, shown in Figure 17, are similar to those obtained above. The age-threshold algorithms at normalized age-threshold $t = .075$ had the best GCU (.124), followed by greedy (.169), cost-benefit A4 (.174), and cost-benefit A5 (.199). Without age-grouping, the GCU of algorithm A5 improved to .191. One difference between this experiment and the previous one is that the GCU of the age-threshold algorithms increases more slowly for large normalized age-thresholds above .15.

For both experiments we tried varying the number of buckets used in the bucket-sort age-threshold algorithms A1 and A2 in the region, .05 to .10, of the optimal normalized age-threshold. Increasing the number of buckets to 20 and decreasing it to 5 had little effect on the results. In both cases, the curve remained essentially flat in this region, and the minimum GCU changed by less than 2.5% as compared to the 10 bucket case.

For both types of experiments, additional simulations were done by varying ASU to .7 and .9, or by eliminating the write cache, while keeping other parameters unchanged. Although the absolute GCU numbers varied depending on the case, the age-threshold algorithm (at the optimal normalized age-threshold) had smaller GCU than greedy and cost-benefit in all cases.

18 Age-Thresholds for Unknown Workloads

Earlier, we had shown how to select an optimal age-threshold for a workload that follows the hot-and-cold model of track writing. In the absence of any knowledge of the workload, we propose a dynamic learning algorithm for choosing an age-threshold. We start the system with an age-threshold of 0, and set the value of a tuning knob called "direction" to *up*. The algorithm then repeatedly performs the following three steps: (1) Measure GCU over a long period of time. (2) If the measured GCU is larger than that computed during the previous period (and this is not the first iteration), then change the value of direction (for example, from *up* to *down*); otherwise keep it the same. (3) Increase or decrease the value of age-threshold depending on the value of the direction knob. Of course the age-threshold should not be allowed to go below zero, and it is probably also a good idea to set some upper bound above which the age-threshold is not allowed to go. Under a workload for which GCU as a function of age-threshold has only one local minimum[5], this learning algorithm will eventually home in on a good age-threshold, provided that the period over which GCU is calculated is sufficiently long and the amount that the age-threshold is increased or decreased is sufficiently small. (Both of these parameters must be determined to make the learning algorithm precise.) For example, we have observed that there is a unique local minimum in the hot-and-cold model of track writing for a number of ASU's and degrees of

[5] It is also assumed for this simple learning algorithm that ASU is fairly constant, so that changes in GCU are not being caused by changes in ASU.

hotness. And our simulations using the power-law trace and the snake trace also indicate a single local minimum.

If the dynamic learning algorithm is considered too difficult to implement, we propose using the following heuristic value for the normalized age-threshold which seems to give reasonable results in many situations. The heuristic is of the form $F \times (1 - \mathrm{ASU} - m)$, where F is a fraction and m is the normalized max-empty; we suggest taking $F = 0.5$. For example, with $\mathrm{ASU} = .8$ and $m = .05$, the heuristic with $F = 0.5$ gives a normalized age-threshold of .075, which is close to the optimal age-threshold for both the power-law trace and the snake trace. Additional simulations using the snake trace were done with $(\mathrm{ASU}, m) = (.7, .05), (.9, .05), (.9, .01)$; in each case examined, this heuristic gave a close-to-optimal age-threshold.

If the implementation of garbage collection also has a min-empty, such that garbage collection is started whenever the number of empty segments drops to min-empty, then m should be taken as $(\text{max-empty} - \text{min-empty})/(S - \text{min-empty})$. The value of ASU might also need to be modified to account for min-empty; for example, we might want to take ASU as $T/(C(S - \text{min-empty}))$.

19 Conclusion

This paper examined garbage collection algorithms in LSA and LFS systems. We proposed a new age-threshold algorithm for deciding what segments to garbage collect. By simulations on synthetic traces and simulations on a real trace, we compared the age-threshold algorithm to previously proposed algorithms such as cost-benefit and greedy. The age-threshold algorithm was found to be superior to the cost-benefit algorithm and the greedy algorithm, over a range of age-thresholds. We noted that the age-threshold algorithm has some implementation advantages over the cost-benefit algorithm, if we want to maintain a list of the segments sorted according to their desirability for garbage collection. Maintaining such a sorted list is important to producing scalable designs.

The age-threshold algorithm forces segments newly filled by writes from the host system to wait a certain time (the age-threshold) before they become candidates for garbage collection. From the segments that become candidates, we select the least utilized segments for garbage collection. We looked at several variations of this algorithm. We found that a variation in which both segments filled by host writes and segments filled by garbage collection are made to wait has best performance. We also found that performance is not hurt if we do not strictly sort all candidate segments by utilization, but instead use a more coarse-grained scheme in which segments are separated into buckets based on their utilization, each bucket covering a certain range of utilizations.

We found that proper choice of age-threshold is critical to the performance of the age-threshold algorithm. An optimal choice can lead to performance superior to the alternatives (cost-benefit and greedy). A poor choice of age-threshold, on the other hand, can lead to performance inferior to the alternatives. In particular, we found that too high a choice of age-threshold can be quite bad, so if we are going to guess wrong, it is better to guess too low than too high. Fortunately, it is fairly easy to avoid bad choices for age-threshold.

For the hot-and-cold model of track writing, we were able to provide an accurate formula for calculating the optimal age-threshold if we knew certain things about the workload like degree of hotness. In the absence of any knowledge of the workload, we proposed two possible schemes for selecting an age-threshold: a dynamic algorithm that "learns" and adjusts the age-threshold, or a simple heuristic formula for age-threshold that gives reasonable results in many situations.

One direction for future research is additional methods for automatically choosing a good age-threshold. One complication is the possibility that multiple applications could be running concurrently with different requirements. There is existing work on adaptive methods for garbage collection in log-structured file systems [5], although of a different type than the methods considered here. Another direction for the future is to test the age-threshold garbage collection algorithm in a prototype LSA system, including methods for dynamic selection of the age-threshold value.

Acknowledgement. We thank Bruce McNutt for helpful discussions.

References

[1] T. Blackwell, J. Harris, and M. Seltzer, Heuristic cleaning algorithms in log-structured file systems, *Proc. USENIX 1995 Winter Conference*, Jan. 1995, pp. 277–288.

[2] C. K. Chow, Determination of cache's capacity and its matching storage hierarchy, *IEEE Trans. Computers* C-25 (1976), pp. 157–164.

[3] W. de Jonge, M. F. Kaashoek, and W. C. Hsieh, The logical disk: a new approach to improving file systems, *Proc. 14th ACM Symposium on Operating System Principles*, Dec. 1993, Asheville, NC, pp. 15–28.

[4] R. M. English and A. A. Stepanov, Loge: a self-organizing disk controller, *Proc. USENIX 1992 Winter Conference*, Jan. 1992, pp. 237–251.

[5] J. N. Matthews, D. Roselli, A. M. Costello, R. Y. Wang, and T. E. Anderson, Improving the performance of log-structured file systems with adaptive methods, *Proc. 16th ACM Symposium on Operating System Principles*, Oct. 1997, pp. 238–251.

[6] B. McNutt, A simple statistical model of cache reference locality and its application to cache planning, measurement, and control, Tech. Report TR 03.411, IBM Santa Teresa Laboratory, San Jose, CA, Sept. 1991.

[7] B. McNutt, Background data movement in a log-structured disk subsystem, *IBM Journal of Research and Development* 38, 1 (Jan. 1994), pp. 47–58.

[8] J. Menon, A performance comparison of RAID-5 and log-structured arrays, *Fourth IEEE Symposium on High-Performance Distributed Computing*, Aug. 1995, Charlottesville, Virginia, pp. 167–178.

[9] D. A. Patterson, G. Gibson and R. H. Katz, A case for redundant arrays of inexpensive disks (RAID), *Proc. ACM SIGMOD International Conference on Management of Data*, June 1988, pp. 109–116.

[10] M. Rosenblum and J. K. Ousterhout, The design and implementation of a log-structured file system, *ACM Trans. Computer Systems* 10 (1992), pp. 26–52.

[11] C. Ruemmler and J. Wilkes, UNIX disk access patterns, *Proc. USENIX 1993 Winter Conference*, Jan. 1993, pp. 405–420.

[12] J. P. Singh, H. S. Stone, and D. F. Thiebaut, A model of workloads and its use in miss-rate prediction for fully associative caches, *IEEE Trans. Computers* 41 (1992), pp. 811–825.

[13] D. F. Thiebaut, On the fractal dimension of computer programs and its application to the prediction of the cache miss ratio, *IEEE Trans. Computers* 38 (1989), pp. 1012–1026.