Simulations of the Age-Threshold and Fitness Free Space Collection Algorithms on a Long Trace

Larry Stockmeyer

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

Abstract. The purpose of this paper is to report results of simulations of two algorithms for free space collection in log-structured storage systems. The algorithms considered are the age-threshold algorithm of Menon and Stockmeyer and the fitness algorithm of Butterworth. The simulations were done using a trace collected by Ruemmler and Wilkes from a file system over a period of two months. The performance of an algorithm is measured by the amount of disk I/O done as a result of free space collection. The performance of the algorithms and several variations of them are compared.

IBM Research Report RJ10222, Almaden Research Center, San Jose, CA, 2001.

1 Introduction

A critical part of a Log Structured Array (LSA) is the procedure for Free Space Collection (FSC). The purpose of this paper is to report results of simulations of two FSC algorithms on a long trace. The algorithms considered are the *age-threshold algorithm* of Menon and Stockmeyer [3], specifically, the bucket-sort version described in [3, §13], and the more recently invented *fitness algorithm* of Butterworth [1]. The principal concepts used in these algorithms will be described briefly. An introduction to the log-structured architecture can be found, for example, in [4] in the context of file systems and in [2, 3] in the context of disk arrays. In a typical log-structured array (LSA), newly-written (virtual) tracks enter a stream of tracks called the *destage stream*. When the destage stream has enough tracks to fill a segment, an empty segment is filled with these tracks and *closed* (written to disk). Whenever one of these tracks is rewritten, the physical track where this track was stored becomes "dead" or "garbage", so it is potential free space. Free space collection is the process of selecting certain segments and extracting the live tracks from them, thus producing new empty segments. The live tracks so produced enter another stream called the *FSC stream*. As with the destage stream, when the FSC stream has enough tracks to fill a segment, an empty segment is filled with these tracks and closed.

The age-threshold algorithm takes a numerical parameter called the age-threshold (for short, AT). When a segment is closed, it must wait until its age exceeds AT before it becomes a candidate for free space collection. The age of a segment is measured with respect to a clock that is incremented by one every time a segment is closed from the destage stream. (There is another version where only segments closed from the destage stream must wait. See Note 1. Notes are collected in Section 10.) Among the segments whose age is larger than AT, the algorithm uses a *greedy* method to select segments. The greedy method selects the segment whose utilization is smallest, where the *utilization* of a segment is the fraction of the segment containing live tracks. Thus, when a segment of utilization u is collected, a fraction u of the tracks in the segment enter the FSC stream, and the remaining $1-u$ fraction is free space. An issue in using the age-threshold algorithm is that a value for AT must be chosen. It is shown in [3] that the optimal value depends on the workload and the amount of free space in the LSA system, and two methods for choosing an AT are suggested. One of these methods is a heuristic that chooses the AT based only on the total amount of free space in the system, a quantity that is easy to measure. It is also shown in [3] that a natural way to express AT is as a fraction of the number of segments in the system; this fraction is called the *normalized age-threshold*, or NAT for short.

The fitness algorithm does not have any workload-dependent parameters to be chosen. There are two innovations in the fitness algorithm [1]. The first is a new criterion for selecting segments for FSC. This criterion uses the age of a segment as well as its utilization. (The cost-benefit criterion of [4] also depends on both age and utilization, but cost-benefit and fitness use different algebraic combinations of age and utilization.) The fitness of a segment having age A and utilization u is

$$Fitness = A \times \frac{(1-u)^2}{u}. \tag{1}$$

The second innovation is the use of multiple destage and FSC streams. When a track, say a destage track, is to be placed in a destage stream, it first goes through a binary decision tree to route it to one of the destage streams. The routing is based on the age of the track, with the goals of placing tracks of similar age into the same stream, and closing segments from the streams at about the same rate. The age of a track is meant to approximate the time before the track is likely to be written in the future; the calculation of track age is described in Section 2.3, following the description in [1].

Another issue that is relevant to our study is the collection of segments whose utilization is high, although not 1 (think, say, utilization 0.9), and whose utilization is not decreasing with time because the tracks in the segment are not being written; call these "frozen segments". Because the age-threshold algorithm uses the greedy selection criterion, it might never collect frozen segments of sufficiently high utilization no matter how old they become; thus, the free space in these segments is lost. For this reason, it is suggested in [3] to augment the age-threshold algorithm with another process (that might run during relatively idle periods) to collect frozen segments. For the fitness algorithm, however, the fitness function increases with increasing age, so a frozen segment will eventually become old enough to be collected, obviating the need for a separate process for this.

This study has five main goals:

1. For the fitness algorithm, study how performance depends on the number of destage streams and FSC streams;

2. For the age-threshold algorithm, study how the addition of multiple destage streams and FSC streams affects the performance;

3. Compare the FSC performance of the fitness and age-threshold algorithms;

4. Investigate how well the algorithms handle scenarios where there are a significant number tracks that are never written, having the potential to create frozen segments;

5. See how the addition of an age-threshold to the fitness algorithm affects its performance.

In the next section we describe in more detail the trace, the performance measures used, the simulation program and its parameters, and how the parameters were chosen.

2 Preliminaries

2.1 The trace

We used the "snake" trace collected by Ruemmler and Wilkes [5]. The trace was collected on a 3GB file server over a period of two months. It contains about 6.76 million writes. Further information about the trace can be found in [5]. The trace was preprocessed to ignore the reads and convert the writes into a sequence of track writes. The storage in the traced system was first divided into tracks of length 32KB. This yielded a total of about 98000 tracks in the system. Each write event is described in the original trace by a disk ID, starting address, and length of the write. Each such write event was converted to a sequence of one or more track writes; a track was included if any part of the track was written. Of the total 98000 tracks, 36607 tracks were written (at least once) in the trace. Another preprocessing step was done to rename the written tracks with ID's from 0 to 36606. In the case of many small writes in sequence to the same track, these conversions can cause the same track ID to appear several times consecutively in the converted trace. To filter out these "redundant" writes, a write cache of size at least one track was used in the simulations. For simulations that went through the trace once, the preprocessed trace was split into two parts: the first part containing the first one million writes, which was used as a warm-up; and the second part called the *working trace* containing the rest, about 5.76 million writes.

2.2 Performance measures

The performance measure used in [3] is Garbage Collection Utilization (GCU). This is the average utilization of segments collected by the FSC algorithm. The GCU also includes segments that become empty "naturally" because all the tracks in the segment are written before the segment is collected. Such segments do not go through the FSC process; they are placed immediately in the pool of empty segments. However, in the computation of GCU, each such event is treated as though a segment of utilization zero was collected, thus giving the FSC algorithm "credit" for this.

Preliminary simulations uncovered situations where GCU was not a good measure of performance, due to caching effects in the destage streams. (These situations do not arise in the simulation model of [3], and GCU does provide a good measure of performance in that paper; see Note 2.) When the amount of caching increases, one would expect GCU to increase, but the number of collected segments to decrease. Therefore, we also consider two measure of the FSC "work" done, which depend on both GCU and the number of collected segments. These two measures model two extremes of the amount of reading work that is done when collecting a segment: for the *WorkRdLive* measure, only the live tracks are read out of the segment; for the *WorkRdAll* measure, all tracks (the number of tracks the segment can hold) are read.

For each simulation run, the program reports GCU, the number of segments collected including naturally emptied segments (call this number CltdCount), and the number of naturally emptied segments (call this number EmptyCount). Let SegTrkCount be the number of tracks per segment. The first definition of work measures the number of times a track is read or written during FSC, assuming that only the live tracks are read from a collected segment. Thus, when a segment of utilization u is collected, SegTrkCount $\times\, u$ tracks are both read and written. So the first work measure, which we call WorkRdLive, is

$$\mathrm{WorkRdLive} = 2 \times \mathrm{SegTrkCount} \times \mathrm{GCU} \times \mathrm{CltdCount}.$$

The second definition is similar, except that the assumption is that the collection of a non-empty segment causes reading of SegTrkCount tracks to extract the live tracks from the segment. (This assumption was used by Rosenblum and Ousterhout [4] in their definition of "write cost".) Under this assumption, the total number of tracks read is SegTrkCount$\times$(CltdCount $-$ EmptyCount), and the total number of (live) tracks that are eventually written back is SegTrkCount$\times$GCU$\times$CltdCount. The second work measure, WorkRdAll, is thus

$$\mathrm{WorkRdAll} = \mathrm{SegTrkCount} \times \mathrm{GCU} \times \mathrm{CltdCount} \ + \ \mathrm{SegTrkCount} \times (\mathrm{CltdCount} - \mathrm{EmptyCount}).$$

By definition, for a particular run of a particular algorithm, WorkRdLive $\leq$ WorkRdAll. Another way to view the difference between the two measures is to note that WorkRdAll gives "extra credit" to the FSC algorithm whenever a segment becomes empty naturally, whereas WorkRdLive does not. For example, when comparing the difference in work between a segment emptying naturally and a segment with one live track being collected, the difference is 2 for WorkRdLive, and SegTrkCount+1 for WorkRdAll. When a statement is made about "Work", the implication is that the statement holds for both WorkRdLive and WorkRdAll.

When comparing two different algorithms (e.g., the fitness and age-threshold algorithms), we give the comparison for GCU and both work measures. However, for simplicity, when giving performance results for a single algorithm, we report only GCU and WorkRdLive.

2.3 The simulation program and its parameters

The simulation program was written by Harry E. Butterworth, then at IBM Hursley. Both the age-threshold and the fitness selection criteria use the age of a segment. The age of a segment is computed, as in [3], as follows. The age is based on the *destage sequence number* of the segment. There is a destage sequence counter, initially zero. Whenever a segment is closed from a destage stream, its destage sequence number is set to the current value of this counter, and the counter is incremented by one. Whenever a segment S is closed from an FSC stream, its destage sequence number is set to the maximum of the destage sequence numbers of the segments that contributed tracks to S. The age of a segment is the difference between its destage sequence number and the current value of the destage sequence counter. The sorting of tracks into streams is based on the "age" of tracks. Whenever a track T is removed from a segment S (either because the track is rewritten or because S is chosen for free space collection), the age of T is the difference between the destage sequence number of S and the current value of the destage sequence counter. The age of a track is intended to approximate the time since the track was last written. (The elapsed time could be obtained exactly by storing with each track the time when it was last written, and this is a direction for further research.) All segment and track ages defined in this way can be determined using a small amount of information stored in controller memory, namely, one number per segment.

For the following description, it is useful to note a detail of how the simulation is done. When there is a write to a track in any of the destage or FSC streams, the track is removed from the stream and put through the binary decision tree for destage streams to route it to a new stream. Thus, the streams act as a cache (although not an LRU cache), and we would expect the hits to this cache to occur more often to the destage streams than to the FSC streams. This simulation choice was made to model a particular LSA system under development. The simulations of [3] do this differently (see Note 2).

Next are listed the parameters given to the program and how they were chosen in our experiments.

- *LsaTrkCount*: the number of tracks containing data. For the simulations of Sections 4 and 5, where there are no unwritten tracks, LsaTrkCount was set to the number of written tracks in the trace, that is, 36607. In Sections 7 and 8, we consider cases where the system initially contains data tracks that are never written during the simulation on the trace. In these cases, LsaTrkCount was set to three times the number of written tracks, that is, 109821. In the sequel a "track" means a virtual track containing data, as opposed to an empty virtual track or a physical track.

- *SegTrkCount*: the number of tracks per segment. Because actual LSA systems typically have a large number of segments, and because the number of (written) tracks was fixed by the trace, it was desired to keep SegTrkCount fairly small. In all simulations, SegTrkCount was fixed at 20.

- *LsaSegCount*: the number of segments. This parameter was chosen to give a desired Average Segment Utilization (ASU), where

$$\mathrm{ASU} = \frac{\mathrm{LsaTrkCount}}{\mathrm{LsaSegCount} \times \mathrm{SegTrkCount}}.$$

  With the exception of one case in Section 8, ASU was held fixed at 0.8.

- *CdmBucketCount*: the number of buckets. For the purpose of selecting segments based on utilization or fitness, the range $[0, 1]$ of utilization values is split into CdmBucketCount subranges of equal size, and all segments having utilization in the same subrange are kept in the same bucket. CdmBucketCount was held fixed at 10.

- *Sorting*: either Fifo or Tree. If Sorting = Fifo, each bucket is maintained as a FIFO queue. If Sorting = Tree, the segments in each bucket are kept sorted by age, with the oldest segment at the "head" of its bucket. For the fitness algorithm, both values of Sorting were explored. For the age-threshold algorithm, only Sorting = Fifo was done, because Sorting = Fifo is used in the reference version of the age-threshold algorithm [3], and it was not a goal of this paper to investigate how Sorting = Tree affects the age-threshold algorithm.

- *Selection*: either Greedy or Fitness. If Selection = Greedy, the next segment selected is the segment at the head of the lowest-utilization non-empty bucket. If Selection = Fitness, the fitness function (1) is computed for each segment at the head of each bucket, and the one having largest fitness is selected. Note that these methods perform only approximations to greedy and fitness selection, and for the fitness algorithm the method is more accurate if Sorting = Tree.

- *Threshold*: the age-threshold, AT, used in simulations of the age-threshold algorithm. A segment is not allowed to be collected until its age exceeds AT, where age is measured by the number of segments closed from a destage stream. Threshold was set to 0 for simulations of the fitness algorithm, with the exception of simulations of Section 9 where the fitness algorithm was tried with a positive AT.

- *DtgStreamExp* and *FscStreamExp*: $\log_2$ of the number of destage and FSC streams, respectively. During preliminary simulations, the exponents were varied from 0 to 4. Butterworth observed that the results were being skewed because increasing DtgStreamExp had the side effect of increasing the size of the "destage stream cache" described at the beginning of this section. Butterworth suggested keeping DtgStreamExp and FscStreamExp fixed at 4, thus keeping the size of the stream caches fixed, and using two other parameters (described next) to vary the amount of age sorting done by the decision trees.

- *DtgSortExp* and *FscSortExp*. The nodes at the highest DtgSortExp levels of the destage decision tree route tracks based on their age, while the remaining lower levels route incoming tracks randomly; and similarly for FscSortExp. These two parameters were varied from 0 to 4. In particular, the value 0 indicates the extreme case where no sorting was done, and 4 indicates the extreme case where sorting was done at all nodes of the tree. We sometimes specify a particular choice of these parameters by an ordered pair (DtgSortExp, FscSortExp). The program varies the sort boundaries dynamically, with the goal that all streams receive tracks at about the same rate: when sorting is done at a node of the tree, the node maintains a relatively long-term average value of the ages of the tracks that pass through the node; tracks are sent left or right depending on whether their age is above or below the average for that node.

- *CacheTrkCount*. The program can simulate an LRU cache placed in front of the LSA, and CacheTrkCount is the number of tracks the cache can hold. Two cases were considered: *Cache = 1* where CacheTrkCount = 1 and *Cache = 320* where CacheTrkCount = 320. The case Cache = 1 is essentially no cache; as described in Section 2.1 the value 1 was chosen to filter out consecutive writes to the same track. The size 320 of the "large cache" was chosen so that the maximum number of tracks in the intended cache is at least as large as the maximum number of tracks in the "destage streams cache" comprised of 16 destage streams, each of which can hold hold up to SegTrkCount = 20 tracks before a segment is closed from the stream. For the case Cache = 1, the cache hit ratio over the working trace was 0.223; for Cache = 320, the hit ratio was 0.950.

- *MaxEmpty*: This parameter is the number of empty segments produced by FSC in the simulation before FSC is stopped and track writing is done to fill these empty segments; then the cycle repeats. MaxEmpty was fixed at 1. This models the situation where track writing and FSC are operating in parallel and in equilibrium. (Results of [3] on the hot-and-cold synthetic trace seem to indicate that choosing MaxEmpty somewhat greater than 1 improves the performance of the age-threshold algorithm. This was found not to be true for the simulation program and trace used here. So we are not putting the age-threshold algorithm at a disadvantage by fixing MaxEmpty = 1. See Note 3 for more information.)

- *DtgDecay* and *FscDecay*. These parameters affect how the sort boundary (the long-term average) is computed at each sorting node of a binary decision tree. As suggested by Butterworth, both were fixed at 10.

In addition, the inputs to the program include a sequence of one or more trace files. At the start of the simulation, all LsaTrkCount tracks are placed into segments in sequential order. Then 10×LsaTrkCount uniformly random writes are made to the tracks, including any extra "unwritten" tracks that are introduced as described in item *LsaTrkCount* above. The program then reads indices of written tracks from the trace files in sequence. Statistics such as GCU and CltdCount are computed separately for each trace file. To summarize, for all of our simulation runs, with the exception of certain runs in Section 8: first there is a warm-up of $10\times$LsaTrkCount uniform writes to place the LSA in a random state; then there is a warm-up using the first one million writes from the trace; and then the simulation is run on the working trace and performance values are obtained for this part of the trace. The first million writes are used to provide a transition from the random state to a typical trace state.

3 Summary of Results

The GCU and Work values for various cases are shown in Figure 1, at the optimal DtgSortExp and FscStreamExp for each case. For each case shown in the table, the optimal GCU, WorkRdLive, and WorkRdAll values occur in the case (DtgSortExp, FscSortExp) $= (4, 4)$. Because 4 was the value used for DtgStreamExp and FscStreamExp in the experiments, we could only consider $0 \leq$ DtgSortExp, FscSortExp ≤ 4. It is possible that performance would continue to improve for values of the stream and sort exponents greater than 4. However, as explained above, there are complications in comparing the same algorithm with different values of DtgStreamExp, because the size of the "destage streams cache" varies with DtgStreamExp.

The AT of the age-threshold algorithm was varied to find approximately the lowest GCU, WorkRdLive, and WorkRdAll with respect to the choice of AT. (Sometimes GCU, WorkRdLive, and WorkRdAll are minimized at different AT values. As described in Section 5, the discrepancy is not significant.) While it is useful to compare other algorithms against the best case of the age-threshold algorithm at the optimal AT, it is not clear how the optimal AT can be found in practice for a changing workload. Therefore, the table also contains performance numbers for the age-threshold algorithm at NAT $= 0.10$. This is the value given by the simpler heuristic in [3, §18], which is NAT $= 0.5\times(1 - \text{ASU})$. In the table, this algorithm is called "age-thr-nat.10". We also considered the effect of adding a positive age-threshold to the fitness algorithm. For both Sorting = Tree and Sorting = Fifo, three values of AT were considered, corresponding to NAT's of 0.01, 0.05, and 0.10. In each case, the best performance was found at NAT $= 0.10$, so we report results only for this case. This algorithm is called "fit-age-nat.10". This algorithm can be compared to the fitness algorithm to see the effect of the age-threshold on the fitness algorithm, and it can be compared with the age-thr-nat.10 algorithm to see the difference between fitness selection and greedy selection with a fixed NAT $= 0.10$.

For comparison, performance numbers for the greedy algorithm are also shown. The greedy algorithm was simulated by choosing Selection = Greedy, Threshold = 0, and DtgSortExp = FscSortExp = 0. However, the greedy algorithm is ignored in the discussion to follow.

Some observations about the values in Table 1 can be made. The data in the table divides naturally into eight cases depending on: Unwritten Tracks = No or Yes; Cache = 1 or 320; and performance measure = WorkRdLive or WorkRdAll. In each case, there are six algorithms (not counting greedy).

1. *The range of differences.* Comparing the six algorithms in each case, the percentage by which the Work of the worst exceeds that of the best is maximized at 24% in the case (Y, 1, Live). The percentages in the other seven cases vary from roughly 9% to 16%.

Unwritten Tracks?	Algorithm	Cache Size	Bucket Sorting	GCU	WorkRdLive (K)	WorkRdAll (K)	EmptyCount (K)
N	fitness	1	fifo	0.183	480	1094	22.7
N	fitness	1	tree	0.193	513	1016	28.4
N	age-threshold	1	fifo	0.194	517	946	32.1
N	age-thr-nat.10	1	fifo	0.200	536	1020	29.3
N	fit-age-nat.10	1	fifo	0.179	467	946	29.7
N	fit-age-nat.10	1	tree	0.194	517	991	30.0
N	greedy	1	fifo	0.581	1528	2079	0.0
Y	fitness	1	fifo	0.134	336	793	31.4
Y	fitness	1	tree	0.135	339	739	34.2
Y	age-threshold	1	fifo	0.144	366	694	37.9
Y	age-thr-nat.10	1	fifo	0.157	404	787	35.0
Y	fit-age-nat.10	1	fifo	0.131	326	698	35.6
Y	fit-age-nat.10	1	tree	0.131	329	698	35.9
Y	greedy	1	fifo	0.576	1493	2043	0.0

(a) Cache = 1

Unwritten Tracks?	Algorithm	Cache Size	Bucket Sorting	GCU	WorkRdLive (K)	WorkRdAll (K)	EmptyCount (K)
N	fitness	320	fifo	0.359	313	564	1.4
N	fitness	320	tree	0.378	340	579	2.0
N	age-threshold	320	fifo	0.370	328	534	3.9
N	age-thr-nat.10	320	fifo	0.370	328	549	2.9
N	fit-age-nat.10	320	fifo	0.355	308	528	3.0
N	fit-age-nat.10	320	tree	0.377	339	556	3.1
N	greedy	320	fifo	0.583	754	1023	0.0
Y	fitness	320	fifo	0.309	251	480	2.6
Y	fitness	320	tree	0.302	243	458	3.2
Y	age-threshold	320	fifo	0.310	253	411	6.1
Y	age-thr-nat.10	320	fifo	0.324	270	451	5.0
Y	fit-age-nat.10	320	fifo	0.307	249	429	5.1
Y	fit-age-nat.10	320	tree	0.299	240	418	5.1
Y	greedy	320	fifo	0.576	733	1003	0.0

(b) Cache = 320

Figure 1: Summary of performance results at the optimal DtgSortExp and FscSortExp.

2. *Fitness-fifo versus fitness-tree.* Fitness-fifo has smaller WorkRdLive than fitness-tree with the exception of case (Y, 320, Live). Fitness-tree has smaller WorkRdAll than fitness-fifo with the exception of case (N, 320, All). As shown by the more detailed results in Section 4, if DtgSortExp ≥ 2 and FscSortExp ≥ 2: fitness-fifo does less WorkRdLive than fitness-tree; fitness-fifo does less WorkRdAll than fitness-tree for Cache = 320; and fitness-tree does less WorkRdAll than fitness-fifo for Cache = 1. If DtgSortExp ≥ 1 and FscSortExp ≥ 1, there are only three exceptions. (See Figure 6.)

3. *Fitness (with $AT = 0$) versus age-threshold.* In the four cases where the measure is WorkRdLive, a fitness algorithm finishes first, and fitness-fifo finishes ahead of fitness-tree in all except (Y, 320, Live). In the four cases where the measure is WorkRdAll, the age-threshold algorithm (with optimal AT) finishes first. The likely explanation for this difference between the two measures is that the age-threshold algorithm gives segments a longer time to empty naturally before they are collected, and, as noted above, WorkRdAll gives "extra credit" to segments that empty naturally. The first point is supported by the EmptyCount results in the table: the age-threshold algorithm has noticeably larger EmptyCount than fitness-fifo and fitness-tree in all four cases. This is consistent with the explanation that a large enough AT is more effective at letting segments become completely empty when compared with the fitness selection method with AT = 0. (There is a rough correlation in the table between increasing EmptyCount and decreasing WorkRdAll, but not a perfect one because GCU and CltdCount also enter into the computation of WorkRdAll.) As can be seen from the more detailed results given in Section 6 comparing the fitness-fifo and age-threshold algorithms for all 25 values of (DtgSortExp, FscSortExp), the general trend is that the fitness-fifo algorithm performs better than the age-threshold (optimal AT) algorithm under GCU and WorkRdLive, and the age-threshold (both optimal AT and NAT = 0.10) algorithm performs better than the fitness-fifo algorithm under WorkRdAll. Most of the exceptions occur when either DtgSortExp = 0 or FscSortExp = 0. (See Figure 13.)

4. *Fitness (NAT = 0) versus fitness (NAT = 0.10).* In each of the eight cases we can compare the performance of the fitness algorithm with that of the fit-age-nat.10 algorithm. In each case, there are two subcases of this, Sorting = Tree or Fifo. In all sixteen cases except one, the introduction of the positive age-threshold causes a decrease in Work; the exception is the case (N, 1, Live, Tree) where there is a $< 1\%$ increase in WorkRdLive. Under WorkRdLive, the improvement is at most 3% in each case. Under WorkRdAll, the improvement is larger (as high as 13.5% in the case (N, 1, All, Fifo)). As in the previous item, the larger improvement in WorkRdAll is probably due to an increase in EmptyCount when the age-threshold is used.

A few experiments were done to compare the fitness selection method with the age-threshold method independently of the issue of multiple streams. This was done by setting DtgStreamExp = FscStreamExp = DtgSortExp = FscSortExp = 0. (This is not completely fair to the fitness method because it was designed to be used in conjunction with multiple streams.) The results are shown in Figure 2. They are similar to those in Figure 1; in particular, the fitness-fifo algorithm has smallest WorkRdLive, and the age-threshold (optimal AT) algorithm has smallest WorkRdAll, for both Cache = 1 and 320. These results are not mentioned further in the paper.

Unwritten Tracks?	Algorithm	Cache Size	Bucket Sorting	GCU	WorkRdLive (K)	WorkRdAll (K)	EmptyCount (K)
N	fitness	1	fifo	0.129	682	1825	57.7
N	fitness	1	tree	0.138	738	1759	63.8
N	age-threshold	1	fifo	0.136	725	1574	72.4
N	age-thr-nat.10	1	fifo	0.143	769	1746	66.0
N	fitness	320	fifo	0.351	313	564	1.9
N	fitness	320	tree	0.373	344	595	1.9
N	age-threshold	320	fifo	0.369	334	549	4.0
N	age-thr-nat.10	320	fifo	0.373	343	570	3.1

Figure 2: Results for the case of one destage stream and one FSC stream.

The following additional observations can be made about the results given later.

5. In almost all cases, GCU, WorkRdLive, and WorkRdAll decrease as the SortExp's increase. That is, if $0 \leq i \leq i' \leq 4$, $0 \leq j \leq j' \leq 4$, and $(i,j) \neq (i',j')$, then GCU (resp., Work) at the point (DtgSortExp, FscSortExp) $= (i',j')$ is smaller than GCU (resp., Work) at the point (i,j). There are a few sporadic instances where an increase of 1 in one of the exponents causes GCU or Work to increase by at most 1%. There was a significant increase only in a very few case where WorkRdAll at $(1,j)$ is larger than WorkRdAll at $(0,j)$. In particular, the smallest GCU, WorkRdLive, and WorkRdAll are always found in the case $(4,4)$.

6. When the algorithms are run many times on the trace and there are unwritten tracks, the fitness algorithm shows a more significant decrease in GCU, WorkRdLive, and WorkRdAll than either the age-threshold algorithm with unwritten tracks or the fitness algorithm with no unwritten tracks. This suggests that the fitness algorithm with unwritten tracks is reorganizing the unwritten tracks better than the age-threshold algorithm. (The simulation of the age-threshold algorithm does not have a process for collecting frozen segments.)

The rest of the paper contains the following. For the case where every track is written at least once (see item *LsaTrkCount* above), Sections 4 and 5 contains results on the fitness algorithm and the age-threshold algorithm, respectively. In Section 6, the results obtained from the two algorithms are compared. In Section 7 we turn to the case where there are twice as many unwritten tracks as written tracks. In Section 8, again in the case of unwritten tracks, we describe results obtained by running the algorithms on the trace ten times in sequence, in an attempt to see how well the algorithms reorganize the unwritten tracks. Section 9 contains results for the fit-age-nat.10 algorithm.

4 The Fitness Algorithm

Results for the fitness algorithm, with no unwritten tracks, are shown in Figure 5. (Figures showing detailed results and plots are collected in the Appendix.) Each table entry contains three numbers: GCU, CltdCount (in thousands), and WorkRdLive (in thousands). Some observations can be made about this data.

1. As mentioned earlier, in all cases GCU and WorkRdLive generally decrease as the SortExp's increase (there is only one exception in Figure 5). The only significant increase is an increase by 2.3% to 6.5% in WorkRdAll (not shown in the tables) when going from point $(0, j)$ to $(1, j)$, for $j = 0, 1, 2, 3, 4$ in the case Sorting = Fifo, and $j = 0$ in the case Sorting = Tree, both in the case Cache = 1.

2. CltdCount (the number of segments collected, including naturally emptied ones) remains fairly constant across each row of the table. When Cache = 1, CltdCount increases down each column. Increasing CltdCount together with decreasing GCU is an indication that caching in the destage streams is less effective as DtgSortExp increases. An explanation for this behavior, due to Harry Butterworth, is that as DtgSortExp increases, young (and presumably hotter) tracks are concentrated in fewer destage streams, with the consequence that they spend on average less time in the streams before they are put in a closed segment. However, the increase in CltdCount is accompanied by a decrease in GCU, which is enough to make WorkRdLive decrease. In the case Cache = 320, CltdCount decreases slowly with increasing DtgSortExp. This is consistent with the explanation: if there is an LRU cache in front of the LSA, "hits" in the destage streams are less likely to occur.

3. To compare Sorting = Tree with Sorting = Fifo, Figure 6 gives the percentage by which the GCU (first number in entry), WorkRdLive (second number in entry), and WorkRdAll (third number in entry) in the case Sorting = Tree is larger (positive percentage) or smaller (negative percentage) than that in the case Sorting = Fifo. So + (resp., −) means that fitness-fifo (resp., fitness-tree) performs better. With four insignificant exceptions: Fifo has better WorkRdLive than Tree when FscSortExp > 0; Tree has better WorkRdAll than Fifo when Cache = 1 and DtgSortExp > 0; Fifo has better WorkRdAll than Tree when Cache = 320 and DtgSortExp > 0.

Figures 7, 8, and 9 show plots of GCU of fitness-fifo, WorkRdLive of fitness-fifo, and WorkRdLive of fitness-tree, respectively, as a function of (DtgSortExp, FscSortExp).

5 The Age-Threshold Algorithm

The age-threshold algorithm was simulated at NAT's from 0.025 to 0.175 in steps of 0.025. We did not consider larger NAT's because the fraction of free space in the system is $1 - \text{ASU} = 0.2$. Most of the optimal NAT's were 0.1 or larger. Recall that the age-threshold algorithm was always simulated with Sorting = Fifo. Figure 10 shows the results. For each entry, GCU, CltdCount, and WorkRdLive are shown. In the left-hand tables, GCU and WorkRdLive are given at the optimal NAT among those tested. Sometimes, the smallest GCU and the smallest WorkRdLive or WorkRdAll occurred at two different NAT's. However, GCU at the WorkRdLive-optimal NAT was within 0.5% of the optimal GCU, and WorkRdLive at the GCU-optimal NAT was within 0.4% of the optimal Work. For WorkRdAll, these percentages were at most 1% with a few exceptions. The CltdCount values at the optimal NAT's differed by at most 1K. In the right-hand tables, NAT is fixed at 0.10. Regarding the percentage increase in WorkRdLive at the fixed NAT = 0.10 compared to that at the WorkRdLive-optimal NAT: for Cache = 1 this percentage is at most 5.01% and is 3.76% at the point $(4, 4)$; for Cache = 320 it is at most 3.65% and is 0% at the point $(4, 4)$. The changes in CltdCount for changing DtgSortExp and FscSortExp are similar to those for the fitness algorithm. Plots of GCU and WorkRdLive using optimal AT's are shown in Figures 11 and 12.

6 Comparison of the Fitness and Age-Threshold Algorithms

To compare the age-threshold algorithm with the fitness algorithm, the tables in Figure 13 show the percentage by which GCU, WorkRdLive, and WorkRdAll of the age-threshold algorithm is larger (positive percentage) or smaller (negative percentage) of that of the fitness-fifo algorithm, both at the optimal AT and at NAT = 0.10. One fact that can be seen clearly in these tables is that, with a few exceptions occurring mostly when DtgSortExp = 0 or FscSortExp = 0, the fitness-fifo algorithm has smaller GCU and WorkRdLive, while the age-threshold algorithm (both optimal AT and NAT = 0.10) has smaller WorkRdAll. To see the effect of increasing SortExp's, we can look at the main diagonal of each table, *i.e.*, the points (i, i) for $0 \leq i \leq 4$. The general trend is that whichever algorithm has the advantage (fitness for GCU and WorkRdLive, age-threshold for WorkRdAll) increases its advantage as i increases. However, there is often a decrease in advantage when going from $(3, 3)$ to $(4, 4)$, and there are a few cases where a decrease in advantage occurs for smaller i.

7 Unwritten Tracks

The algorithms were simulated in a scenario where, in addition to the 36607 tracks that are written in the trace, the tracks include twice as many (73214) tracks that are never written during the simulation on the trace. This gives a total of LsaTrkCount = 109821. Recall that the unwritten tracks are written during the initial warm-up consisting of $10 \times$ LsaTrkCount uniformly random writes. Keeping SegTrkCount fixed at 20, LsaSegCount was tripled to maintain ASU = 0.8. Detailed results for the fitness algorithm are shown in Figure 14, and plots are shown in Figures 15 (GCU) and 16 (WorkRdLive). Results for the age-threshold algorithm are shown in Figure 17, and plots (using optimal AT's) are shown in Figures 18 (GCU) and 19 (WorkRdLive). Comparing these results with those in the case of no unwritten tracks, the absolute GCU and Work values are smaller in the case of unwritten tracks. However, the relationships between the numbers in the case of unwritten tracks are similar to those in the case of no unwritten tracks. For example, performance still tends to increase as the SortExp's increase.

The table in Figure 20 compares the performance of the fitness-fifo algorithm with that of the age-threshold algorithm in the case of unwritten tracks; this is the analogue for unwritten tracks of

Algorithm	Unwritten?	Cache	SortExp's	Percentage Improvement GCU	WRL	WRA	Percentage Above Goal GCU	WRL	WRA
fitness	Y	1	0	34.6	43.6	25.3	19.9	24.9	11.2
age-thr	Y	1	0	17.9	25.0	15.0	55.6	77.3	29.9
fitness	N	1	0	2.4	3.7	2.2			
fitness	Y	1	4	69.6	72.0	52.1	10.7	10.6	6.9
age-thr	Y	1	4	36.9	39.3	22.7	134.2	147.7	42.2
fitness	N	1	4	25.1	26.4	16.7			
fitness	Y	320	0	37.7	51.4	32.3	20.0	12.2	12.2
age-thr	Y	320	0	13.9	22.8	15.2	65.5	50.0	45.0
fitness	N	320	0	3.1	6.0	4.0			
fitness	Y	320	4	78.8	84.7	66.4	23.3	25.0	10.3
age-thr	Y	320	4	56.5	66.5	47.1	180.0	211.4	55.5
fitness	N	320	4	34.7	46.0	30.7			

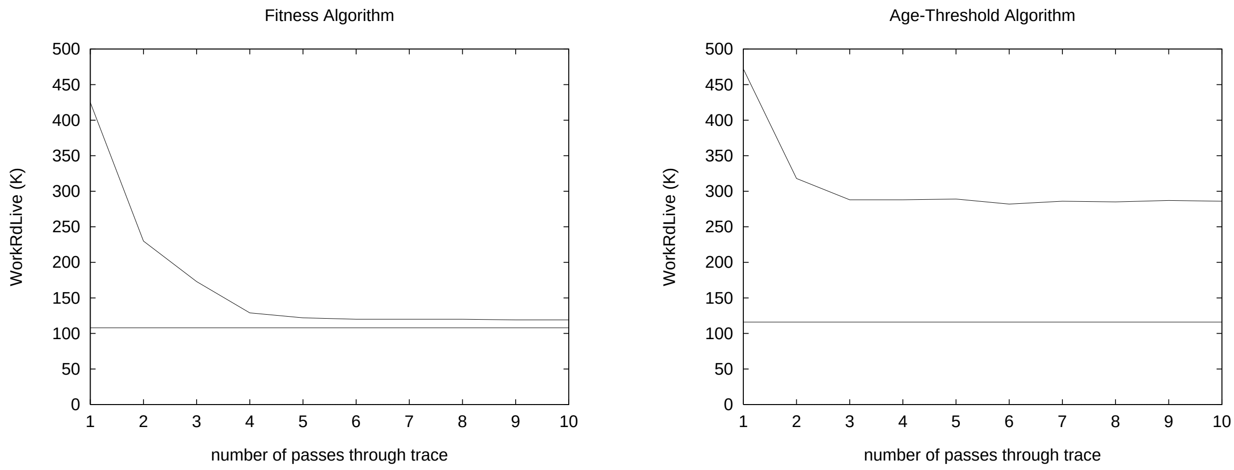
Figure 3: Results from running through the trace ten times.

Figure 13. As in the case of no unwritten tracks, the fitness-fifo algorithm has better performance according to GCU and WorkRdLive, while the age-threshold algorithm does better according to WorkRdAll.

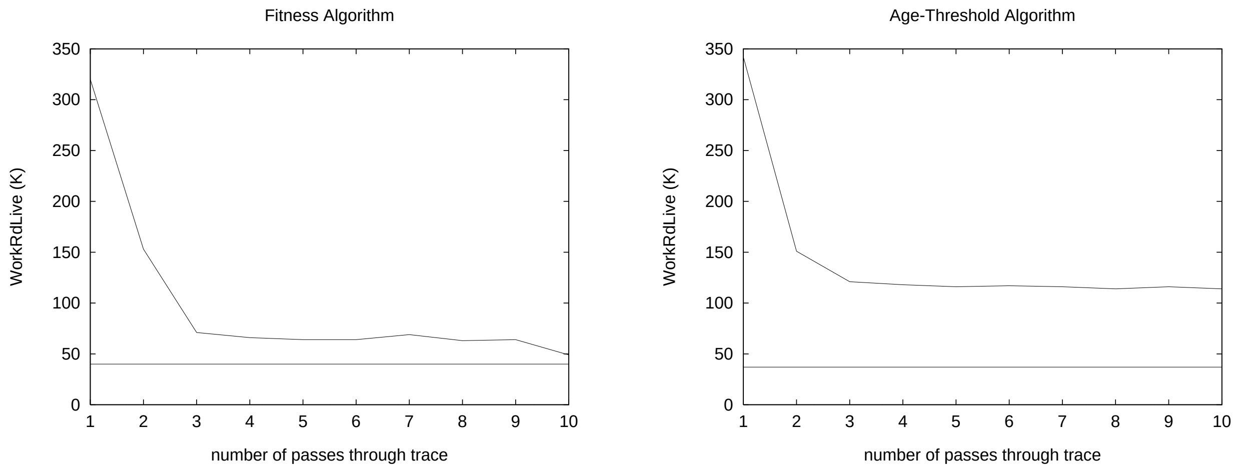
8 Reorganization of Unwritten Tracks

In this section we report results of running the algorithms for ten times in sequence on the full trace, starting with unwritten tracks just as in the previous section. The goal was to see how much improvement in GCU and Work is obtained from the first run on the trace to the tenth run. In summary, the fitness algorithm shows more improvement than the age-threshold algorithm. It is reasonable to conjecture that the fitness achieves better improvement because it does a better job of collecting "frozen" segments and congregating the unwritten tracks into fewer segments. To test this conjecture, we also ran the fitness algorithm ten times over the trace where there were no unwritten tracks. Here the fitness algorithm shows some improvement but it is not as large as that obtained with unwritten tracks. This supports the conjecture. In the case of no unwritten tracks, the improvement could be due to a better congregation of the tracks that are written relatively rarely. Adding a large number of unwritten tracks makes the improvement larger.

We also compare the GCU and Work of an algorithm at the tenth iteration with the GCU and Work that would occur if the algorithm was completely successful in congregating all the unwritten tracks into segments by themselves. Although we would not expect an algorithm to reach this perfect state, it is interesting to see how close it comes to this "goal". To find the GCU and Work goals, we compute the ASU of the system when all the unwritten tracks are in segments by themselves. In the scenario with unwritten tracks and ASU $= 0.8$, all the tracks fill a fraction $8/10$



(a) Cache = 1



(b) Cache = 320

Figure 4: Plots of WorkRdLive of the fitness-fifo (left) and age-threshold (right) algorithms by number of passes through the trace, in the case DtgSortExp = FscSortExp = 4. The straight horizontal lines are at the goal values.

of the segments. Of these tracks, 2/3 are unwritten. If all the unwritten tracks were in segments by themselves, the unwritten tracks would fill $(8/10)(2/3) = 8/15$ of the segments, and the written tracks would fill $(8/10)(1/3) = 4/15$ of the segments. Thus, the effective ASU is $(4/15)/(1 - 8/15)$ $= 4/7$. To compute the GCU and Work goals for a given case, the case was simulated ten times on the trace with no unwritten tracks and ASU = 4/7; that is, LsaTrkCount = 36607, LsaSegCount = 3203, and SegTrkCount = 20.

The cases considered were Cache = 1 and 320, Sorting = Fifo and Tree for the fitness algorithm, and DtgSortExp = FscSortExp = i for $0 \leq i \leq 4$. For simplicity we only report results at the extreme points $i = 0$ and $i = 4$, and only for Sorting = Fifo. With other parameters fixed, Sorting = Fifo consistently showed larger improvement than Sorting = Tree. A fixed AT = 1121 was used for the age-threshold algorithm. This value is close to the optimal value for both scenarios: ASU = 0.8 and unwritten tracks; ASU = 4/7 and no unwritten tracks. The NAT's in the two scenarios are 0.163 and 0.35. The heuristic of [3] described above would give 0.10 and 0.21, respectively.

Results are shown in Figure 3 (where WorkRdLive and WorkRdAll are abbreviated WRL and WRA). For each case are given the percentage improvement in GCU and Work from the first iteration to the last, and the percentage that the tenth-iteration value is larger than the goal value for the case.

Most of the improvement in GCU and Work occurred during the earlier iterations. This can be seen in Figure 4 for the fitness-fifo and age-threshold algorithms with unwritten tracks in the cases Cache = $1, 320$ and SortExp = 4.

It is also reasonable to wonder what is happening during the early runs on the trace. To check this, the trace was divided in four parts of equal size. The WorkRdLive (K) values during the first eight quarters (twice through the whole trace) are

141	81	113	90
78	50	57	45

This suggests that certain quarters of the trace need more work than others.

9 The Fitness Algorithm with an Age-Threshold

Detailed results for the fit-age-nat.10 algorithm without and with unwritten tracks are shown in Figures 21 and 22, respectively.

10 Notes

1. The FSC simulation program used in this study simulates the *all-age* version of the age-threshold algorithm, where all closed segments must wait to pass the AT before being collected. In another version, the *TW-age* version, only segments closed from the destage stream must wait. Results of Sections 16 and 17 of [3] show that on a realistic synthetic trace and on the trace used here, the GCU's of the all-age and TW-age versions are virtually identical for all reasonably small values of the AT, in particular, for all values of AT smaller than or in the vicinity of the "optimal" AT where GCU is minimized. Because we consider only such

values of AT in this study, one would expect the results to have been virtually identical if the TW-age version had been simulated.

2. The simulation program used in this study allows multiple destage and FSC streams, and a write to track in a stream causes the track to be removed from the stream and placed in a new destage stream. This means that a closed segment is always full. For the simulation program used in [3], there is only one destage stream and one FSC stream, and a write to a track in the destage stream causes the track to be removed from the stream, but a "hole" remains in the stream where the track was. This means that a destage segment can contain holes, i.e., free space, when closed. In particular, this means that if a sequence of N track writes enters the LSA, the FSC process must create very close to N tracks of free space. (This is not true if the streams act as a cache.) Thus, smaller GCU implies that a smaller number of segments are collected. So in [3], decreasing GCU always means increasing performance, and vice versa.

3. Experiments were done to determine the effect on the performance of the age-threshold algorithm caused by increasing MaxEmpty. Both Cache = 1 and Cache = 320 were considered. In one type of experiment, MaxEmpty was increased slowly from 1 to $0.05 \times$ LsaSegCount, while Threshold was decreased from its optimal (at MaxEmpty = 1) value at each step by the same amount that MaxEmpty increased. (The reason why it is reasonable to decrease AT by the same amount that MaxEmpty increases is explained in [3, §10].) The second type of experiment was the same except that Threshold was held fixed at its optimal value. In all cases, both GCU and work generally increased as MaxEmpty increased, with the exception of a few cases with MaxEmpty $\leq 0.01 \times$ LsaSegCount where the measure decreased by less than 0.6%.

Acknowledgements. I am grateful to Harry Butterworth, not only for supplying the simulation program, but also for many helpful suggestions about the details of the simulations. In particular, he noticed early on that varying the number of streams was producing skewed results, and he suggested the solution of varying the amount of stream sorting instead. I also thank Jody Glider for helpful discussions about work measures for FSC algorithms.

References

[1] H. E. Butterworth, The design of segment filling and selection algorithms for efficient free-space collection in a log structured array, unpublished manuscript, IBM Hursley, 1999.

[2] 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.

[3] J. Menon and L. Stockmeyer, An age-threshold algorithm for garbage collection in log-structured arrays and file systems, IBM Research Report RJ 10120, May 1998; a shorter

version appears in *High Performance Computing Systems and Applications*, J. Schaeffer, ed., Kluwer Academic Publishers, 1998, pp. 119–132.

[4] 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.

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

Appendix: Tables and Plots

Sorting = Fifo

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.397 49 784	0.377 48 721	0.365 47 687	0.358 47 667	0.353 46 653
1	0.338 57 768	0.304 55 674	0.290 55 636	0.280 54 606	0.277 54 598
2	0.299 61 731	0.262 60 634	0.246 60 593	0.238 60 574	0.232 60 555
3	0.272 63 687	0.236 63 590	0.219 63 549	0.211 62 528	0.207 63 517
4	0.245 66 649	0.209 65 548	0.194 65 509	0.187 65 489	0.183 65 480

Sorting = Tree

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.403 50 801	0.384 48 743	0.386 49 751	0.384 49 747	0.381 48 738
1	0.337 56 760	0.308 55 683	0.304 56 679	0.302 56 674	0.298 56 664
2	0.300 61 727	0.262 61 636	0.256 61 627	0.254 62 627	0.250 62 616
3	0.273 63 682	0.236 63 594	0.228 64 581	0.224 64 576	0.221 64 566
4	0.237 65 621	0.206 66 542	0.198 66 523	0.197 66 523	0.193 66 513

(a) Cache = 1

Sorting = Fifo

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.499 28 553	0.484 27 522	0.475 27 504	0.468 26 490	0.464 26 482
1	0.474 26 502	0.455 26 467	0.441 25 441	0.433 25 428	0.430 25 422
2	0.456 26 465	0.434 25 427	0.417 24 400	0.410 24 387	0.404 23 379
3	0.438 25 434	0.415 24 395	0.391 23 358	0.388 23 353	0.382 23 345
4	0.416 24 396	0.394 23 362	0.367 22 324	0.365 22 321	0.359 22 313

Sorting = Tree

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.504 28 563	0.493 27 539	0.493 27 540	0.489 27 532	0.489 27 532
1	0.479 27 510	0.465 26 483	0.455 26 466	0.452 25 461	0.451 25 459
2	0.463 26 476	0.447 25 448	0.436 25 428	0.430 24 419	0.428 24 416
3	0.443 25 438	0.424 24 408	0.412 24 389	0.407 23 382	0.405 23 379
4	0.414 24 392	0.396 23 365	0.386 23 351	0.382 23 345	0.378 22 340

(b) Cache = 320

Figure 5: [GCU, CltdCount (K), WorkRdLive (K)] of the fitness algorithm.

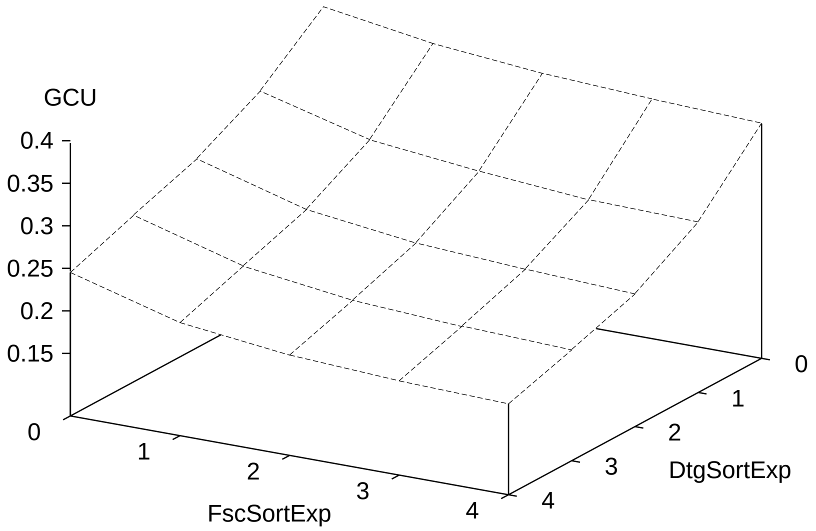
Destage Sort Exponent		FSC Sort Exponent 0	1	2	3	4
	0	+1.46 +2.27 +0.66	+1.79 +3.02 +0.63	+5.70 +9.28 +3.65	+7.34 +11.94 +4.89	+8.07 +13.03 +5.21
	1	−0.32 −1.01 −3.25	+1.21 +1.34 −3.88	+4.65 +6.76 −1.92	+7.71 +11.26 +0.19	+7.55 +11.03 −0.34
	2	+0.56 −0.43 −4.76	−0.00 +0.28 −6.86	+3.99 +5.72 −5.58	+6.46 +9.27 −4.45	+7.72 +10.84 −3.38
	3	+0.16 −0.74 −5.53	+0.29 +0.60 −7.31	+4.40 +5.95 −5.62	+6.13 +8.93 −4.77	+7.11 +9.48 −4.85
	4	−3.12 −4.33 −7.94	−1.56 −1.04 −8.76	+1.90 +2.77 −8.33	+5.27 +6.92 −6.46	+5.36 +6.85 −7.15

(a) Cache = 1

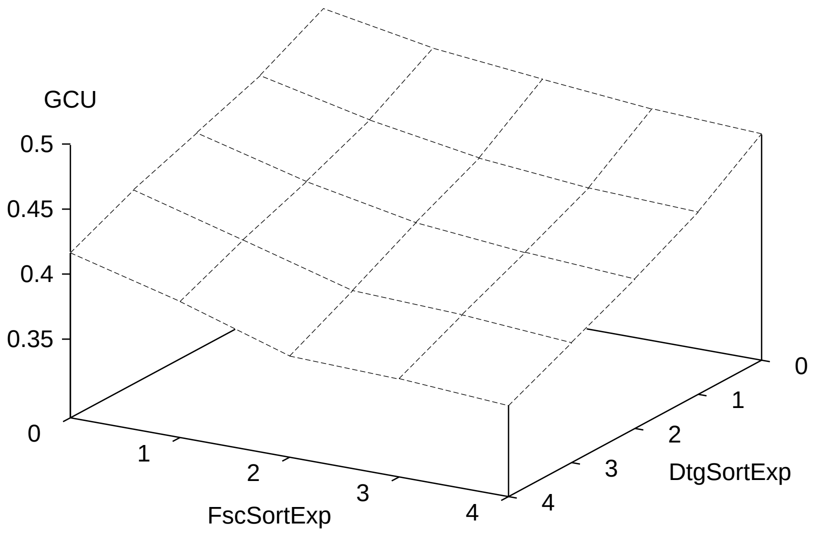
Destage Sort Exponent		FSC Sort Exponent 0	1	2	3	4
	0	+1.05 +1.69 +1.02	+1.97 +3.42 +2.01	+3.79 +6.99 +4.24	+4.56 +8.56 +5.21	+5.48 +10.35 +6.36
	1	+1.07 +1.55 +0.77	+2.12 +3.41 +1.73	+3.30 +5.57 +2.89	+4.46 +7.70 +4.11	+4.89 +8.64 +4.73
	2	+1.50 +2.22 +1.16	+3.02 +5.04 +2.53	+4.36 +7.21 +3.46	+4.92 +8.16 +3.79	+5.77 +9.71 +4.61
	3	+0.93 +1.08 +0.41	+2.07 +3.21 +1.11	+5.23 +8.55 +3.72	+5.05 +8.20 +3.45	+6.10 +9.95 +4.37
	4	−0.46 −1.02 −1.00	+0.57 +0.84 −0.72	+5.22 +8.52 +3.16	+4.65 +7.36 +2.23	+5.30 +8.41 +2.72

(b) Cache = 320

Figure 6: Percentage by which [GCU, WorkRdLive, WorkRdAll] in the case Sorting = Tree is larger (positive percentage) or smaller (negative percentage) than that in the case Sorting = Fifo.

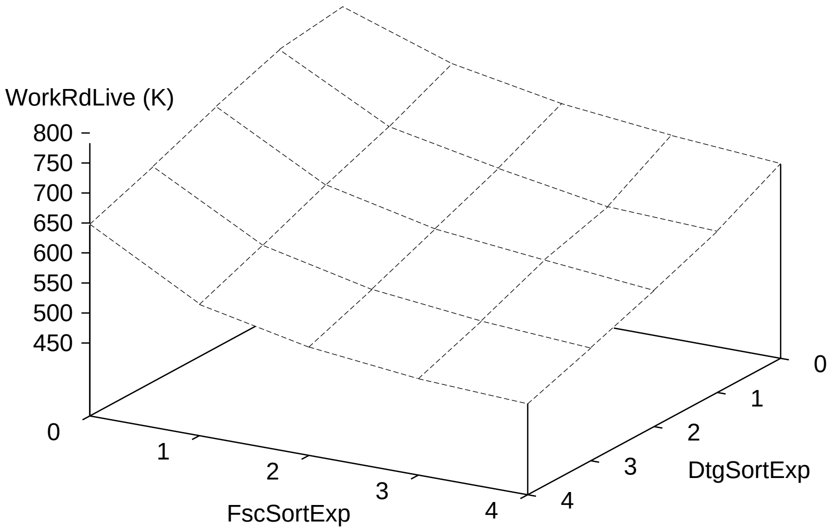


(a) Cache = 1, Sorting = Fifo

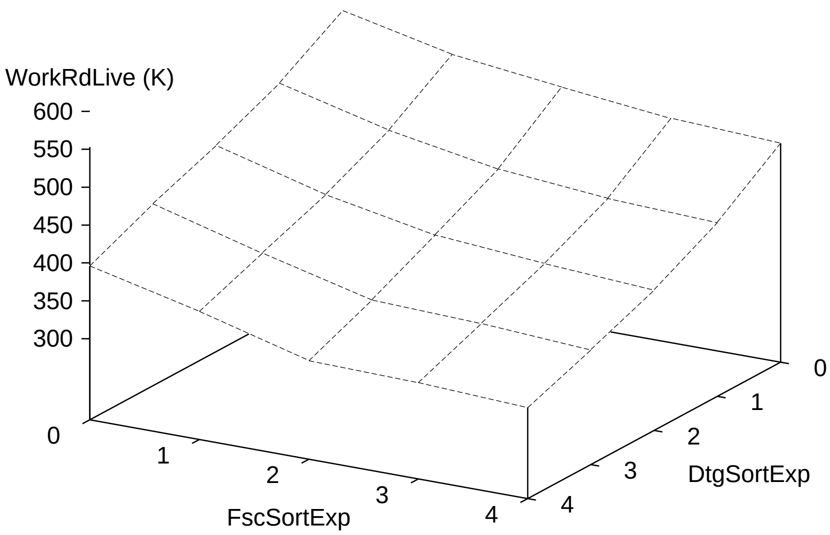


(b) Cache = 320, Sorting = Fifo

Figure 7: Plot of GCU for the fitness-fifo algorithm with Cache = 1, 320.

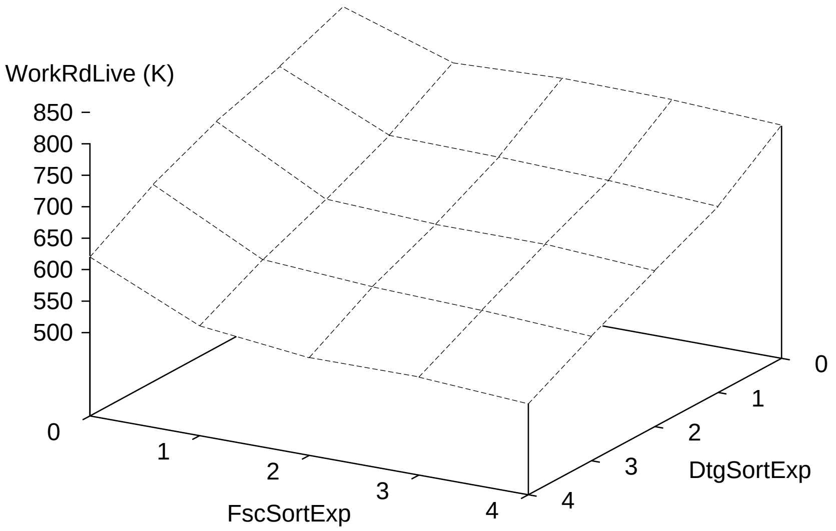


(a) Cache = 1, Sorting = Fifo

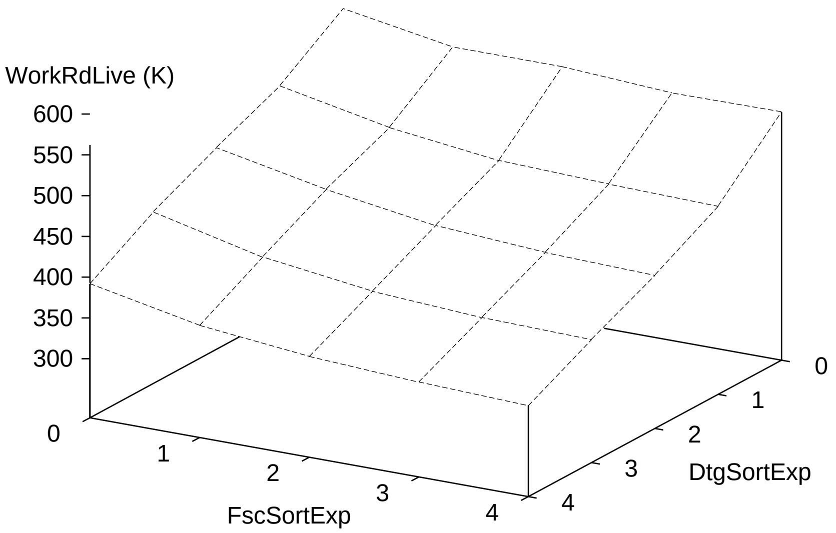


(b) Cache = 320, Sorting = Fifo

Figure 8: Plot of WorkRdLive for the fitness-fifo algorithm with Cache = 1, 320.



(a) Cache = 1, Sorting = Tree



(b) Cache = 320, Sorting = Tree

Figure 9: Plot of WorkRdLive for the fitness-tree algorithm with Cache = 1, 320.

Destage Sort Exponent	FSC Sort Exponent				
	0	1	2	3	4
0	0.397	0.378	0.368	0.360	0.357
	50	48	48	47	47
	790	730	699	674	664
1	0.334	0.313	0.302	0.296	0.291
	57	56	56	56	55
	758	705	674	659	646
2	0.292	0.270	0.258	0.250	0.247
	61	62	62	61	61
	714	664	639	616	606
3	0.265	0.244	0.231	0.224	0.219
	63	64	64	64	64
	671	626	591	575	561
4	0.236	0.216	0.205	0.198	0.194
	66	67	67	67	66
	622	575	545	527	517

At optimal AT

Destage Sort Exponent	FSC Sort Exponent				
	0	1	2	3	4
0	0.397	0.378	0.368	0.360	0.357
	50	48	48	47	47
	790	730	699	674	664
1	0.335	0.315	0.302	0.296	0.292
	57	56	56	56	55
	761	709	674	659	647
2	0.298	0.273	0.262	0.255	0.252
	61	62	62	62	62
	732	674	647	630	623
3	0.273	0.248	0.237	0.231	0.227
	64	64	64	65	65
	693	636	610	595	585
4	0.243	0.220	0.210	0.207	0.200
	67	67	67	67	67
	649	591	565	553	536

At NAT = 0.10

(a) Cache = 1

Destage Sort Exponent	FSC Sort Exponent				
	0	1	2	3	4
0	0.501	0.488	0.479	0.473	0.469
	28	27	27	27	26
	559	532	513	502	494
1	0.477	0.462	0.452	0.445	0.441
	27	26	26	25	25
	510	481	462	452	444
2	0.457	0.442	0.432	0.424	0.420
	26	25	25	24	24
	470	442	425	412	406
3	0.435	0.418	0.407	0.401	0.397
	25	24	24	23	23
	428	402	383	374	369
4	0.407	0.393	0.380	0.373	0.370
	24	23	23	22	22
	383	362	343	333	328

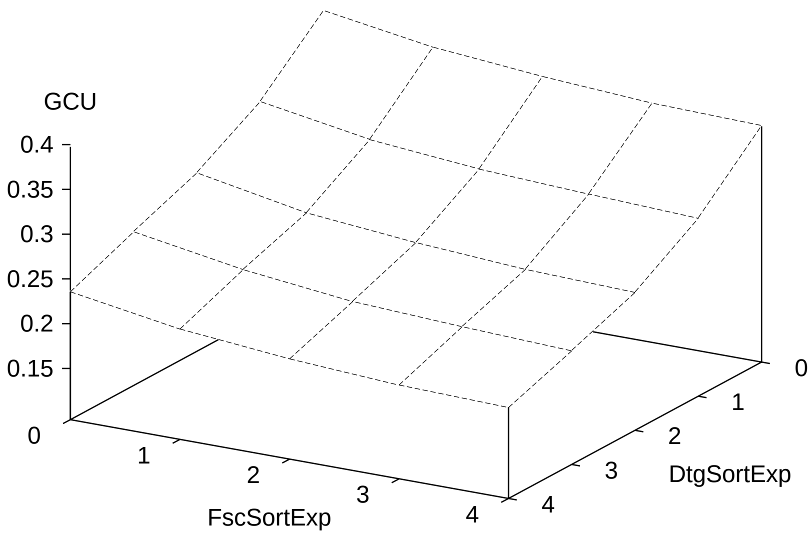
At optimal AT

Destage Sort Exponent	FSC Sort Exponent				
	0	1	2	3	4
0	0.501	0.489	0.482	0.476	0.472
	28	27	27	27	26
	559	534	518	507	499
1	0.479	0.462	0.454	0.446	0.445
	27	26	26	25	25
	514	481	467	452	450
2	0.462	0.443	0.432	0.426	0.423
	26	25	25	24	24
	478	444	426	414	409
3	0.441	0.422	0.407	0.404	0.398
	25	24	24	23	23
	438	407	383	377	369
4	0.416	0.397	0.383	0.375	0.370
	24	23	23	22	22
	397	367	347	335	328

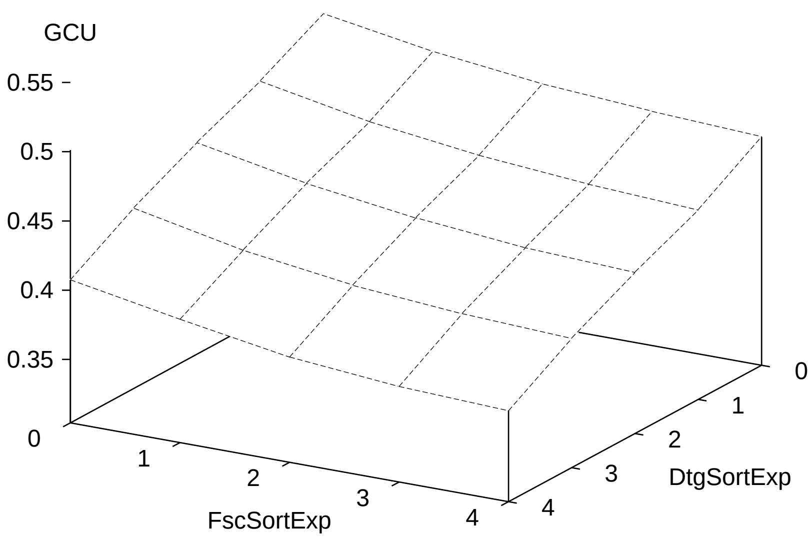
At NAT = 0.10

(b) Cache = 320

Figure 10: [GCU, CltdCount (K), WorkRdLive (K)] of the age-threshold algorithm.

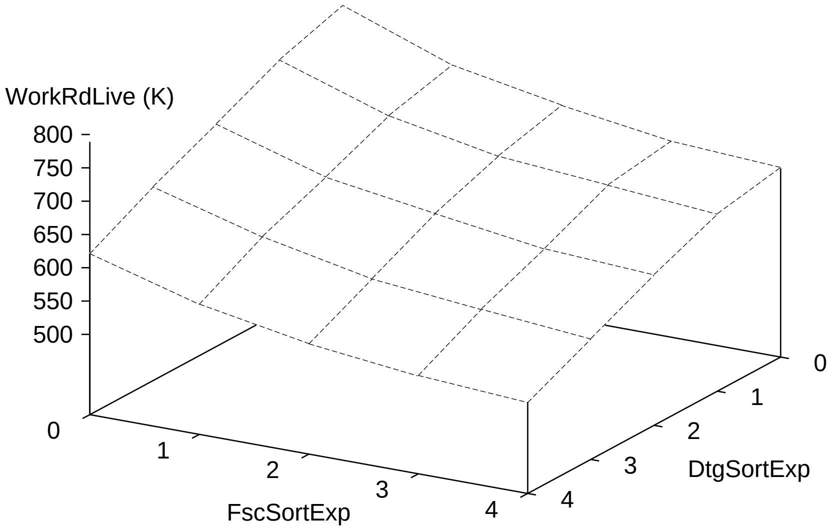


(a) Cache = 1

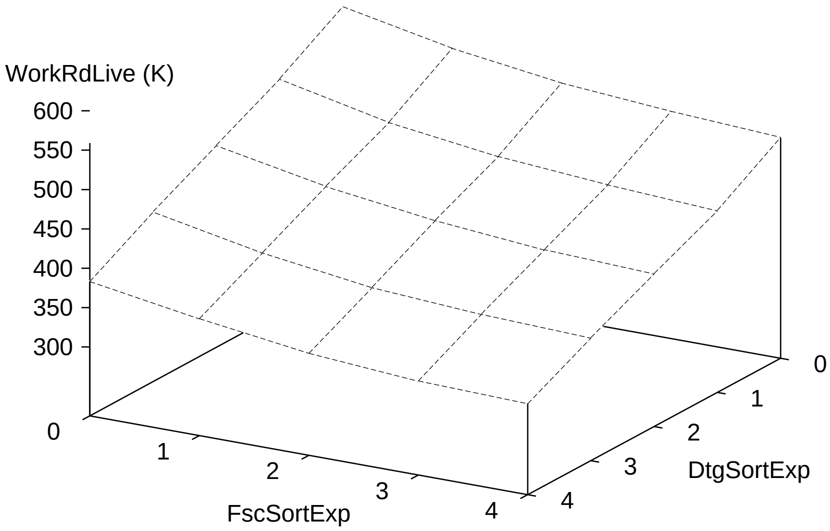


(b) Cache = 320

Figure 11: Plot of GCU for the age-threshold algorithm (at optimal AT).



(a) Cache = 1



(b) Cache = 320

Figure 12: Plot of WorkRdLive for the age-threshold algorithm (at optimal AT).

Destage Sort Exponent	FSC Sort Exponent				
	0	1	2	3	4
0	+0.08 +0.82 −3.96	+0.37 +1.16 −3.97	+0.68 +1.67 −3.69	+0.50 +0.98 −4.12	+1.16 +1.69 −3.95
1	−1.24 −1.31 −14.99	+3.08 +4.61 −11.36	+4.15 +5.92 −10.15	+5.89 +8.80 −9.44	+5.19 +8.06 −9.52
2	−2.12 −2.32 −17.96	+2.85 +4.71 −14.90	+5.07 +7.71 −14.11	+5.03 +7.34 −13.65	+6.22 +9.00 −12.95
3	−2.84 −2.39 −18.58	+3.68 +6.04 −14.98	+5.48 +7.79 −13.87	+6.12 +8.92 −13.41	+6.25 +8.50 −13.48
4	−3.88 −4.15 −19.80	+3.18 +4.98 −15.27	+5.23 +7.08 −13.81	+5.55 +7.71 −13.38	+5.97 +7.63 −13.52

At optimal AT

Destage Sort Exponent	FSC Sort Exponent				
	0	1	2	3	4
0	+0.08 +0.82 −3.96	+0.37 +1.16 −3.97	+0.68 +1.67 −3.69	+0.50 +0.98 −4.12	+1.16 +1.69 −3.95
1	−0.82 −0.92 −12.38	+3.55 +5.23 −9.27	+4.15 +5.92 −8.78	+5.89 +8.80 −6.95	+5.59 +8.29 −7.18
2	−0.17 +0.15 −14.31	+4.12 +6.39 −10.93	+6.39 +9.13 −9.23	+6.91 +9.90 −9.19	+8.40 +12.08 −7.05
3	+0.08 +0.92 −14.16	+5.14 +7.86 −9.90	+8.31 +11.29 −7.93	+9.03 +12.70 −7.04	+9.79 +13.30 −6.44
4	−0.70 −0.01 −13.34	+5.20 +7.84 −9.16	+8.05 +10.98 −7.30	+10.40 +13.11 −6.27	+9.17 +11.67 −6.83

At NAT = 0.10

(a) Cache = 1

Destage Sort Exponent	FSC Sort Exponent				
	0	1	2	3	4
0	+0.39 +1.10 −0.75	+0.86 +1.96 −0.20	+0.79 +1.68 −0.67	+1.21 +2.42 −0.55	+1.26 +2.42 −0.32
1	+0.65 +1.64 −1.98	+1.55 +2.96 −1.62	+2.54 +4.77 −0.69	+2.82 +5.47 −0.69	+2.60 +5.00 −0.35
2	+0.26 +0.94 −4.77	+1.90 +3.62 −3.68	+3.40 +6.24 −2.52	+3.49 +6.43 −2.14	+3.98 +7.28 −1.66
3	−0.88 −1.29 −7.64	+0.79 +1.73 −5.71	+4.07 +6.86 −3.32	+3.43 +5.92 −3.92	+4.12 +6.97 −3.81
4	−2.11 −3.34 −9.19	−0.20 −0.11 −8.16	+3.52 +6.00 −4.60	+2.22 +3.66 −5.58	+2.90 +4.68 −5.32

At optimal AT

Destage Sort Exponent	FSC Sort Exponent				
	0	1	2	3	4
0	+0.42 +1.11 −0.27	+1.11 +2.36 +0.36	+1.36 +2.71 +0.45	+1.80 +3.57 +1.02	+1.76 +3.42 +0.90
1	+1.10 +2.43 −0.74	+1.55 +2.96 −0.52	+3.10 +5.81 +1.05	+2.96 +5.51 +0.74	+3.49 +6.52 +1.33
2	+1.28 +2.68 −1.99	+2.14 +4.06 −1.48	+3.63 +6.49 −0.11	+3.98 +6.97 +0.02	+4.54 +8.00 +0.68
3	+0.55 +1.00 −3.84	+1.67 +3.04 −2.80	+4.07 +6.86 −0.86	+4.07 +6.88 −0.87	+4.32 +7.15 −0.77
4	+0.04 +0.19 −4.55	+0.70 +1.32 −4.40	+4.31 +7.10 −1.03	+2.74 +4.51 −2.73	+2.90 +4.68 −2.68

At NAT = 0.10

(b) Cache = 320

Figure 13: Percentage by which [GCU, WorkRdLive, WorkRdAll] of the age-threshold algorithm exceeds that of the fitness-fifo algorithm.

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.312 44 545	0.286 42 481	0.282 42 474	0.281 42 469	0.276 42 459
1	0.246 52 508	0.217 50 433	0.210 50 417	0.209 50 414	0.206 50 409
2	0.205 58 472	0.180 56 407	0.173 56 391	0.171 56 384	0.170 56 382
3	0.185 61 448	0.160 59 380	0.155 60 369	0.153 60 363	0.150 59 356
4	0.166 64 425	0.142 63 355	0.137 62 342	0.136 62 339	0.134 63 336

Sorting = Fifo

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.322 44 569	0.287 42 484	0.281 42 468	0.278 42 463	0.276 41 458
1	0.252 52 523	0.219 50 438	0.212 50 421	0.210 50 417	0.207 50 411
2	0.215 58 497	0.181 57 409	0.177 56 400	0.174 56 392	0.172 56 389
3	0.192 61 468	0.161 59 382	0.157 59 372	0.155 59 369	0.154 60 367
4	0.170 64 433	0.143 63 357	0.138 62 345	0.136 63 341	0.135 63 339

Sorting = Tree

(a) Cache = 1

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.422 24 408	0.403 23 377	0.395 23 366	0.390 23 358	0.388 23 354
1	0.403 24 379	0.377 23 341	0.365 22 324	0.366 22 326	0.364 22 323
2	0.386 23 353	0.358 22 314	0.345 21 297	0.346 21 298	0.341 21 290
3	0.368 22 326	0.339 21 288	0.328 21 274	0.326 21 271	0.325 21 270
4	0.352 22 305	0.326 21 271	0.316 21 259	0.313 20 256	0.309 20 251

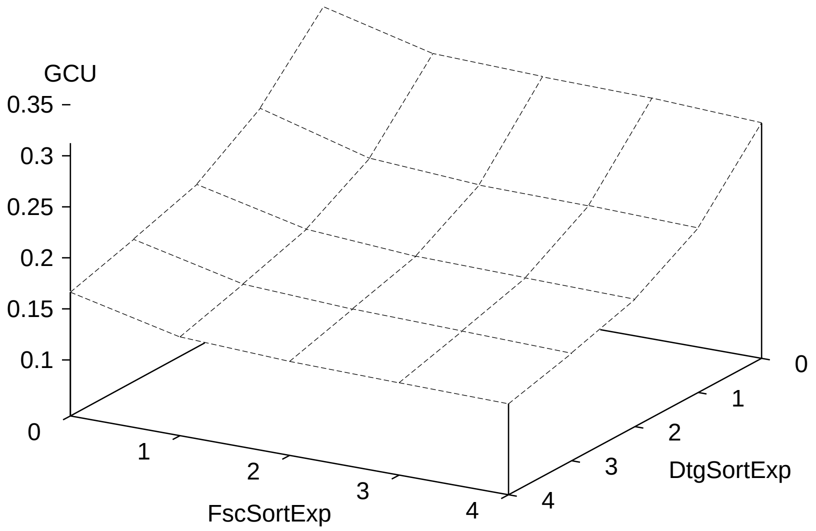
Sorting = Fifo

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.430 24 420	0.400 23 371	0.393 23 360	0.391 23 358	0.389 23 355
1	0.411 24 390	0.374 22 335	0.368 22 327	0.364 22 322	0.361 22 318
2	0.395 23 364	0.356 22 309	0.348 21 298	0.345 21 294	0.341 21 290
3	0.379 22 341	0.335 21 282	0.328 21 272	0.324 21 268	0.321 21 264
4	0.359 22 313	0.315 20 258	0.307 20 248	0.304 20 245	0.302 20 243

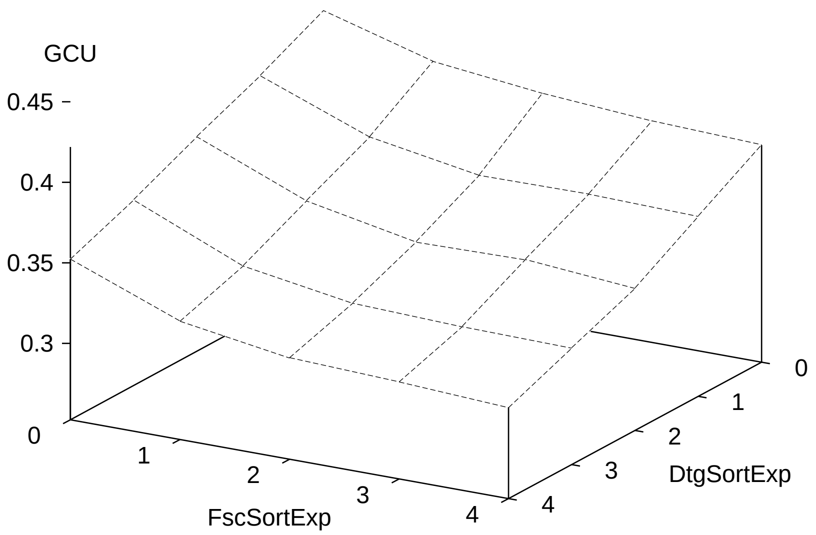
Sorting = Tree

(b) Cache = 320

Figure 14: [GCU, CltdCount (K), WorkRdLive (K)] of the fitness algorithm with unwritten tracks.

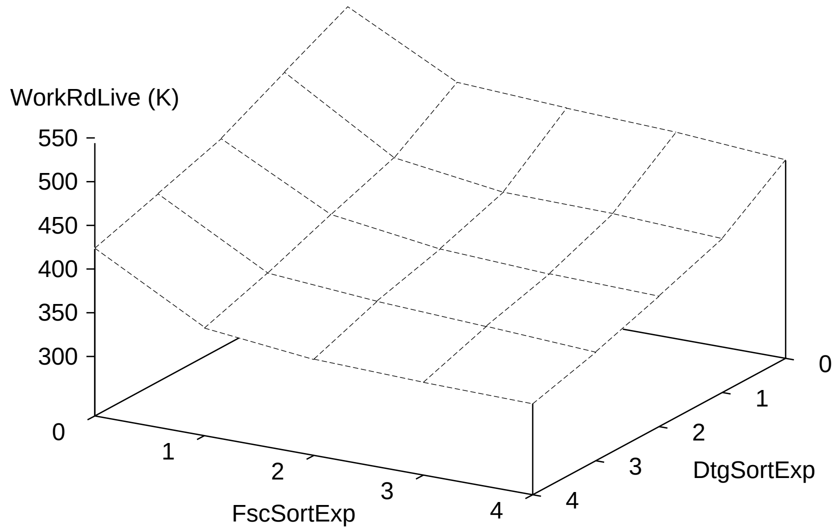


(a) Cache = 1, Sorting = Fifo

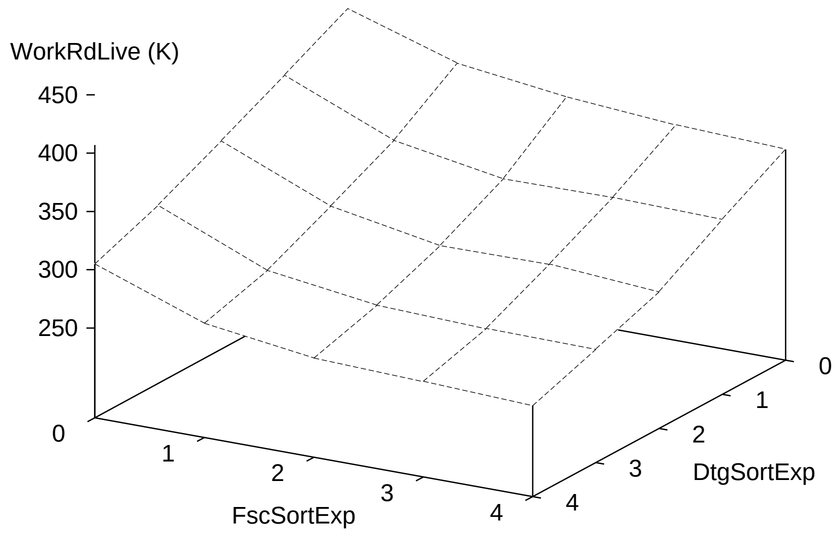


(b) Cache = 320, Sorting = Fifo

Figure 15: Plot of GCU for the fitness-fifo algorithm with unwritten tracks.



(a) Cache = 1, Sorting = Fifo



(b) Cache = 320, Sorting = Fifo

Figure 16: Plot of WorkRdLive for the fitness-fifo algorithm with unwritten tracks.

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.320 44 564	0.299 43 514	0.294 43 501	0.289 42 489	0.286 42 483
1	0.244 52 504	0.229 51 466	0.225 51 457	0.222 51 448	0.218 50 440
2	0.203 58 468	0.189 57 433	0.184 57 420	0.181 57 414	0.180 57 409
3	0.183 61 444	0.171 60 411	0.166 60 400	0.164 60 396	0.161 60 387
4	0.165 64 421	0.152 63 384	0.150 63 380	0.146 63 370	0.144 63 366

At optimal AT

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.322 44 570	0.302 43 519	0.294 43 502	0.291 42 495	0.289 42 488
1	0.258 52 540	0.241 52 498	0.235 51 482	0.231 51 473	0.229 51 467
2	0.223 59 526	0.206 58 480	0.198 58 460	0.196 58 453	0.194 58 451
3	0.202 62 501	0.186 61 456	0.180 61 442	0.177 61 435	0.177 61 433
4	0.185 65 483	0.167 64 431	0.161 64 412	0.158 64 405	0.157 64 404

At NAT = 0.10

(a) Cache = 1

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.421 24 408	0.403 23 379	0.399 23 373	0.393 23 363	0.392 23 363
1	0.396 23 370	0.382 23 348	0.374 23 337	0.371 22 334	0.369 22 331
2	0.372 22 333	0.358 22 314	0.351 22 305	0.351 22 304	0.350 22 303
3	0.353 22 306	0.340 21 290	0.333 21 281	0.330 21 278	0.330 21 277
4	0.333 21 281	0.323 21 268	0.317 21 262	0.315 21 258	0.310 20 253

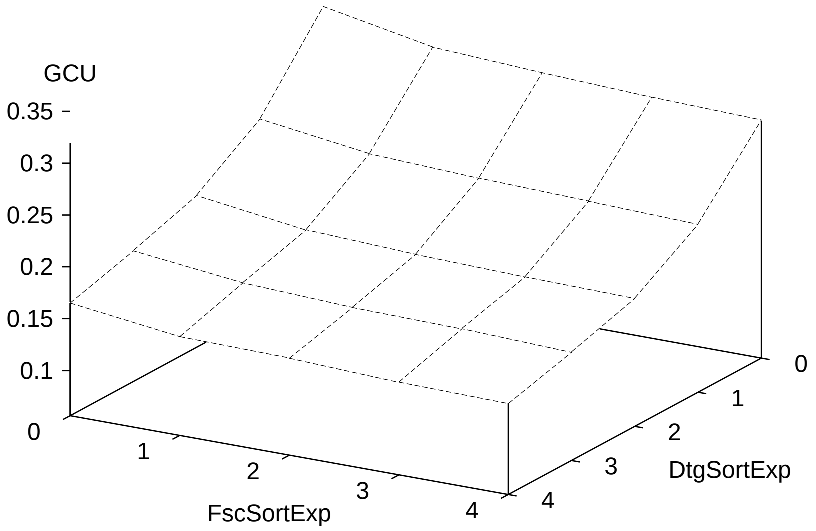
At optimal AT

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.429 24 420	0.410 24 389	0.404 23 379	0.400 23 374	0.398 23 371
1	0.408 24 388	0.390 23 360	0.383 23 351	0.380 23 345	0.378 23 343
2	0.389 23 357	0.373 22 334	0.366 22 325	0.364 22 323	0.362 22 319
3	0.372 22 332	0.355 22 309	0.348 22 300	0.345 21 296	0.343 21 293
4	0.354 22 307	0.335 21 284	0.330 21 277	0.326 21 272	0.324 21 270

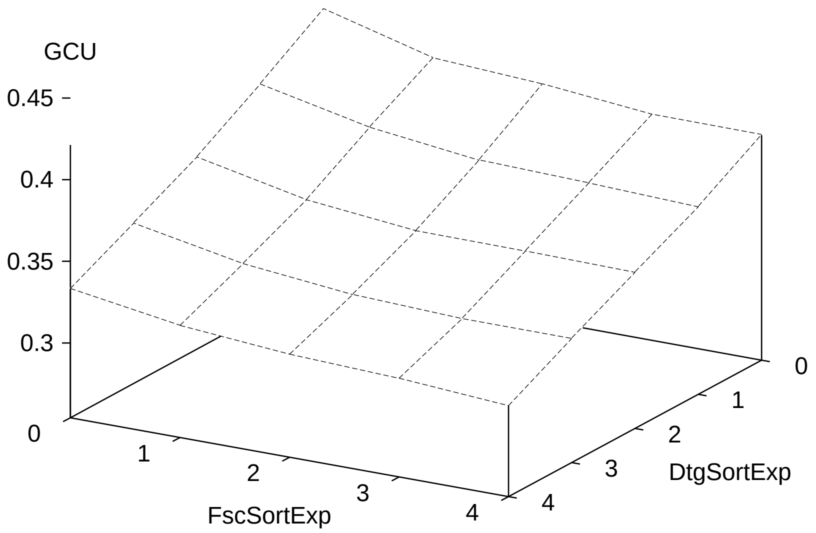
At NAT = 0.10

(b) Cache = 320

Figure 17: [GCU, CltdCount (K), WorkRdLive (K)] of the age-threshold algorithm with unwritten tracks.

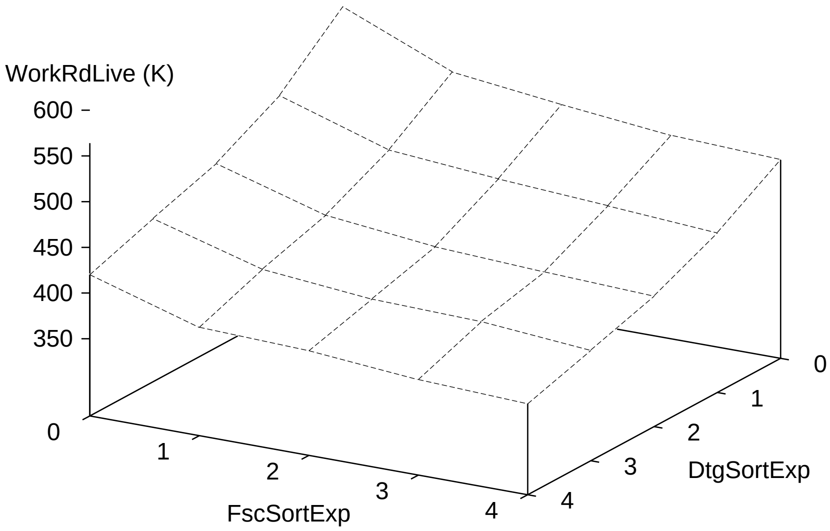


(a) Cache = 1

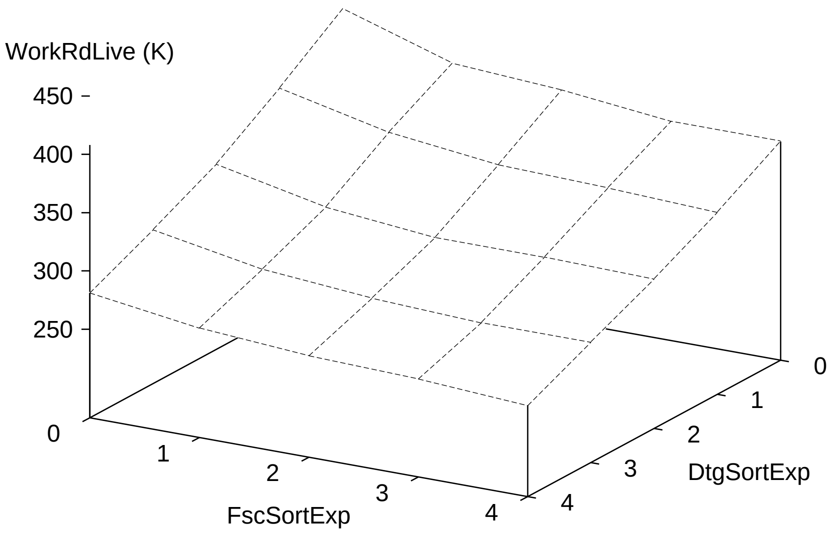


(b) Cache = 320

Figure 18: Plot of GCU for the age-threshold algorithm with unwritten tracks.



(a) Cache = 1



(b) Cache = 320

Figure 19: Plot of WorkRdLive for the age-threshold algorithm with unwritten tracks.

Destage Sort Exponent		FSC Sort Exponent 0	1	2	3	4
	0	+2.30 +3.63 −8.04	+4.74 +6.98 −6.48	+4.03 +5.71 −6.99	+2.94 +4.27 −7.58	+3.67 +5.22 −7.11
	1	−0.97 −0.84 −20.02	+5.85 +7.57 −15.67	+7.29 +9.54 −14.69	+6.09 +8.21 −15.14	+5.87 +7.74 −15.35
	2	−1.01 −0.88 −18.91	+4.73 +6.24 −14.97	+6.18 +7.45 −14.67	+5.95 +7.70 −14.32	+5.90 +7.07 −14.10
	3	−1.18 −0.81 −18.68	+6.68 +8.06 −14.01	+6.93 +8.29 −13.27	+7.75 +9.09 −13.03	+6.89 +8.46 −13.74
	4	−0.74 −0.88 −17.59	+6.96 +8.15 −12.97	+9.28 +11.02 −11.39	+7.63 +9.30 −11.78	+7.45 +8.71 −12.52

At optimal AT

Destage Sort Exponent		FSC Sort Exponent 0	1	2	3	4
	0	+3.00 +4.59 −6.06	+5.53 +7.96 −4.10	+4.19 +5.92 −5.12	+3.77 +5.55 −5.02	+4.61 +6.31 −4.65
	1	+4.72 +6.32 −11.29	+11.39 +14.87 −6.68	+12.21 +15.73 −5.96	+10.72 +14.23 −6.56	+10.94 +14.38 −6.57
	2	+8.69 +11.43 −7.62	+13.95 +17.97 −4.11	+14.40 +17.74 −4.19	+14.42 +17.93 −3.86	+14.46 +18.02 −2.90
	3	+9.37 +11.80 −6.99	+16.35 +19.94 −2.58	+16.32 +19.83 −1.92	+16.16 +19.60 −2.24	+17.59 +21.58 −1.49
	4	+11.36 +13.65 −5.10	+17.82 +21.32 −0.55	+17.31 +20.42 −0.84	+16.56 +19.57 −0.83	+17.22 +20.22 −0.75

At NAT = 0.10

(a) Cache = 1

Destage Sort Exponent		FSC Sort Exponent 0	1	2	3	4
	0	−0.16 +0.09 −5.77	+0.14 +0.39 −5.70	+1.07 +1.88 −4.79	+0.66 +1.27 −5.54	+1.25 +2.34 −4.72
	1	−1.66 −2.44 −11.58	+1.25 +2.23 −9.13	+2.27 +3.89 −8.49	+1.55 +2.67 −9.41	+1.31 +2.37 −8.73
	2	−3.55 −5.56 −15.86	−0.10 +0.00 −13.11	+1.77 +2.79 −11.89	+1.27 +2.25 −11.92	+2.66 +4.37 −10.21
	3	−4.15 −6.16 −17.13	+0.20 +0.53 −13.62	+1.44 +2.60 −12.91	+1.35 +2.27 −13.10	+1.64 +2.65 −12.78
	4	−5.36 −7.81 −18.43	−0.97 −1.16 −14.82	+0.54 +1.08 −13.74	+0.58 +1.07 −14.06	+0.27 +0.58 −14.25

At optimal AT

Destage Sort Exponent		FSC Sort Exponent 0	1	2	3	4
	0	+1.65 +3.06 −2.12	+1.83 +3.21 −2.16	+2.14 +3.68 −1.91	+2.61 +4.49 −1.82	+2.73 +4.69 −1.47
	1	+1.38 +2.46 −5.48	+3.42 +5.68 −3.82	+4.98 +8.21 −2.30	+3.76 +6.01 −3.92	+3.67 +6.11 −3.86
	2	+0.85 +1.38 −8.18	+3.97 +6.41 −5.35	+6.12 +9.45 −3.81	+5.12 +8.36 −4.46	+6.23 +9.85 −3.60
	3	+1.09 +1.77 −8.61	+4.75 +7.45 −5.39	+6.09 +9.53 −4.57	+5.91 +9.09 −4.84	+5.48 +8.47 −5.12
	4	+0.36 +0.56 −9.42	+2.91 +4.49 −7.27	+4.61 +7.02 −5.93	+4.25 +6.40 −6.30	+4.94 +7.43 −6.00

At NAT = 0.10

(b) Cache = 320

Figure 20: Percentage by which [GCU, WorkRdLive, WorkRdAll] of the age-threshold algorithm exceeds that of the fitness-fifo algorithm in the case of unwritten tracks.

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.393 49 778	0.375 48 721	0.361 47 680	0.353 46 656	0.348 46 640
1	0.324 56 725	0.302 55 667	0.287 55 629	0.281 54 613	0.276 54 598
2	0.286 60 688	0.257 60 620	0.242 60 585	0.235 60 566	0.230 60 554
3	0.259 62 646	0.231 63 580	0.216 63 543	0.208 63 523	0.203 63 510
4	0.230 65 600	0.204 66 536	0.191 66 503	0.184 65 479	0.179 65 467

Sorting = Fifo

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.400 50 798	0.389 49 761	0.389 49 765	0.385 49 748	0.378 48 730
1	0.330 56 741	0.312 56 698	0.311 56 701	0.306 56 690	0.300 56 673
2	0.290 60 699	0.267 61 652	0.259 62 640	0.256 62 633	0.252 62 623
3	0.267 62 663	0.237 63 598	0.230 64 587	0.228 64 584	0.224 64 573
4	0.232 65 602	0.208 65 545	0.202 66 536	0.198 67 528	0.194 67 517

Sorting = Tree

(a) Cache = 1

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.493 28 543	0.480 27 517	0.472 26 499	0.462 26 479	0.457 26 470
1	0.467 26 492	0.450 26 460	0.438 25 437	0.431 25 426	0.426 25 418
2	0.448 25 453	0.428 24 419	0.416 24 400	0.409 24 388	0.401 23 375
3	0.427 24 415	0.408 24 385	0.391 23 359	0.384 23 349	0.380 23 343
4	0.403 23 378	0.383 23 348	0.366 22 323	0.362 22 317	0.355 22 308

Sorting = Fifo

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.500 28 556	0.495 27 544	0.493 27 541	0.488 27 531	0.488 27 532
1	0.475 27 504	0.463 26 483	0.457 26 471	0.454 26 467	0.449 25 457
2	0.456 26 466	0.443 25 443	0.433 25 426	0.429 24 419	0.425 24 413
3	0.435 25 427	0.421 24 404	0.410 24 387	0.406 23 381	0.402 23 375
4	0.408 24 384	0.393 23 362	0.385 23 351	0.380 23 342	0.377 22 339

Sorting = Tree

(b) Cache = 320

Figure 21: [GCU, CltdCount (K), WorkRdLive (K)] of the fit-age-nat.10 algorithm.

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.314 44 552	0.289 42 488	0.280 42 470	0.279 42 466	0.273 41 452
1	0.241 51 493	0.216 50 430	0.210 50 418	0.208 50 413	0.206 50 409
2	0.200 57 458	0.177 56 399	0.172 56 385	0.169 56 378	0.167 56 375
3	0.181 60 437	0.156 59 370	0.153 59 363	0.150 59 356	0.151 59 357
4	0.162 64 411	0.139 62 348	0.136 62 339	0.133 62 332	0.131 62 326

Sorting = Fifo

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.315 44 553	0.285 42 478	0.277 42 461	0.274 41 455	0.272 41 450
1	0.239 51 489	0.214 50 427	0.208 50 413	0.206 49 407	0.203 50 403
2	0.202 57 463	0.178 56 402	0.173 56 388	0.170 56 383	0.168 56 378
3	0.181 60 435	0.160 59 379	0.154 59 365	0.152 59 361	0.150 60 358
4	0.160 63 403	0.141 62 351	0.136 63 341	0.133 62 332	0.131 63 329

Sorting = Tree

(a) Cache = 1

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.416 24 400	0.397 23 368	0.390 23 358	0.386 23 352	0.383 23 348
1	0.393 23 366	0.369 22 331	0.365 22 324	0.362 22 321	0.358 22 315
2	0.375 22 337	0.350 22 303	0.346 22 297	0.343 21 293	0.339 21 289
3	0.357 22 312	0.335 21 284	0.327 21 273	0.323 21 269	0.324 21 270
4	0.339 21 288	0.316 21 260	0.313 20 256	0.308 20 250	0.307 20 249

Sorting = Fifo

Destage Sort Exponent	FSC Sort Exponent 0	1	2	3	4
0	0.416 24 398	0.394 23 363	0.388 23 355	0.385 23 351	0.383 23 348
1	0.392 23 363	0.369 22 330	0.365 22 325	0.361 22 319	0.358 22 314
2	0.376 22 338	0.349 22 301	0.344 21 294	0.340 21 290	0.338 21 287
3	0.355 22 309	0.332 21 279	0.324 21 269	0.320 21 264	0.320 21 265
4	0.337 21 285	0.310 20 252	0.305 20 247	0.301 20 242	0.299 20 240

Sorting = Tree

(b) Cache = 320

Figure 22: [GCU, CltdCount (K), WorkRdLive (K)] of the fit-age-nat.10 algorithm with unwritten tracks.