Improving Disk Performance under Paging Workloads

15-712: Final Report

David Koes and David McWherter
{dkoes,cache+}@cs.cmu.edu

December 10, 2003

1 Introduction

Page-oriented workloads are commonplace in many real-world applications, ranging from virtual memory subsystems to database bufferpool management. These workloads are characterized by small, fixed-sized, random-access reads and writes. The layout of traditional disks is optimized for workloads with large sequential accesses. For these sequential workloads, cost of seeks and rotational latencies are amortized over the transfer of a large amount of data to/from the disk, and the application sees good data transfer rates. Each access from a page-oriented workload, however, incurs a time-consuming seek and rotational latency that dominate the request response time, resulting in very low data rates. We exploit the characteristics of page-oriented workloads to develop a new disk layout called a Paged Disk. We also develop a write optimization, called bad tenants, which improves write requests for paged disks. Together, the two techniques optimize disks in favor of paged-workloads, and the resulting disk outperforms existing disk technology.

The central idea in paged disks is to replicate application-defined pages of data on the disk. Pages are replicated uniformly around the circumference of a track, so that the average rotational latency decreases directly with the replication factor. The pages are also stored such that they never span track boundaries, so the cost of track switching is never needed in the middle of an access. Bad tenants exploit the replication done within a paged disk to improve write performance. Since most pages have multiple copies on the disk, an incoming page write can overwrite whatever is currently under the disk head, regardless of where it currently is. Of course, this can only be done if there is another copy of the overwritten data. The disk only needs to maintain and consult a mapping of the locations of bad tenant pages to ensure correctness.

We have modified the validated DiskSim disk simulator to implement paged disks with bad tenants on an existing disk model. We find that paged disks are able to drastically improve the performance of randomly generated workloads as well as real world TPC-C transactional database workloads. In fact, the performance of paged disks with one disk outperforms the performance of a mirrored RAID system using 4 disks.

2 Related Work

Numerous researchers have focused on the problem of reducing the costs of reads and writes to the disk subsystem. Typically, the techniques used involve some combination of data replication (mirroring), striping across multiple disks, dynamic data placement, and track alignment. Our paged disk and bad tenant techniques expand upon the ideas of data mirroring, dynamic data placement, and track-based extents (traxtents).

2.1 Mirroring and Striping

Disk Shadowing [1] is a technique that has long been used to improve disk performance, redundancy and availability. Each disk serves as an identical copy of the others. Writes are copied onto each disk, and reads can be serviced from any of the disks. Since only the first write must make it to stable storage to ensure consistency, the disk with the smallest access latency cost determines the performance of writes. Likewise, reads can also be serviced by the disk with the smallest latency. Britton and Gray show that the expected seek distance can drop around 43% using shadowing. Hou and Patt [9] and Dishon and Liu [7] demonstrate that policies which copy data onto many disks improve performance under simulation.

Ivy [10] examines the benefits of storing multiple copies of a file throughout a file system spanning multiple storage nodes. The system dynamically changes the number of copies of files within the system according to the data usage pattern. They demonstrate experimentally the decrease in seek distance and performance increase gained by increasing the number of copies of the files.

Ng [12] examines analytically the benefits of mirroring data within a single disk. In particular, blocks are placed

twice on the disk, 180 degrees out of phase of one another, in the same track. The result is that the average rotational delay is reduced by a factor of two, improving read performance. SR-Array [17] extends this work, also mirroring blocks onto different tracks to help reduce seek latencies. SR-Array concentrates on the effectiveness of such placement policies in disk arrays, rather than single disks.

Mirroring can reduce the storage capacity of a system drastically. HP AutoRAID focuses on minimizing the storage overhead when using mirroring, while retaining its benefits. The key idea is a storage hierarchy with different storage characteristics at each level. Active data is handled by one level where it is stored in a mirrored form for better performance.. When the data becomes inactive and unused, it is moved to a lower level in the hierarchy, where it is stored in a more space-efficient manner. The system automatically determines this placement, and strikes a good trade-off between performance and storage cost.

2.2 Dynamic Data Placement

Dynamic data placement characterizes systems where the mapping of logical blocks to physical blocks changes, usually allowing the system to eagerly write data to the closest free block when a write request is made.

Loge [8] improves write performance by using dynamic data placement in a technique called *eager writing*. It reserves a small percentage of the disk for free space, and attempts to issue writes to the closest available free block. As the number of free blocks is kept constant by remapping logical to physical block mappings, the seek and rotational latencies needed to perform a write will be relatively low. Loge makes no attempts to optimize read performance, although it is possible that the implicit reorganization of the data through the greedy write placement policy clusters commonly accessed data close together. The Mime [3] system improves upon Loge by adding support for transactional capabilities and crash recovery.

The Virtual Log File System (VLFS) [16] builds upon the ideas of Loge and Mime. The system uses a non-contiguous log file that traces through the free blocks of the disk (similar to Trail [4] with the same device for log and data). One enhancement of VLFS over Loge is to "clean" the disk during idle time, moving data and free blocks around to improve the distribution of free blocks on the disk. As a result, unlike Loge, VLFS ensures that writes will be cheap by preventing regions of the disk from running out of free blocks.

Zhang et al examine the use of eager-writing techniques in disk arrays [18], using mirroring and striping across multiple disks to improve performance. Striping the data increases the throughput of writes and reads by increasing the available disk bandwidth. Mirroring not only improves redundancy, but provides additional scheduling opportunities to improve seek and rotational latency for both reads and writes. Zhang demonstrates that these benefits can be improved upon by adding eager-writing.

The Doubly Distorted Mirror (DDM) [5] system replicates data across two disks and within each disk. Each disk is partitioned into two regions, a master and slave, with the slave 20% larger than the master. Data is allocated dynamically into the slave partition and then copied into the master partition where it is placed statically. Since the slave is larger, there will be a large percentage of free blocks guaranteed to be available. Writes are serviced synchronously by the slave partition, where they are eagerly written to the nearest free block. Written blocks are cached until the cache fills up or they can be opportunistically copied into the master partition. DDMs uses freeblock scheduling [11] to completely hide the cost of copies into the master partition by writing cached copies when a read occurs to the same track.

DDMs improve performance by minimizing the seek and rotational latencies on both reads and writes. Since blocks are available from two separate locations on the disk (the master and the slave partition), the expected distance the head needs to travel to access the data is smaller. Likewise, since written blocks are placed in the free block closest to the head in the slave partition, seek and rotation costs for writes are expected to be small. The system also ensures that sequential reads are efficient as they can be sent to the master partition, eliminating multiple seeks within the randomly organized slave.

2.3 Track Alignment

Trail [4] uses two disks, a logging disk and a data disk, to improve synchronous write performance. The disk head of the log disk is kept over a relatively free track, and log-writes can be issued without seek and minimal rotational latency. Asynchronously, the system propagates writes from the log to the main data disk. Reads are processed either from the Trail buffer cache, which contains all blocks in the log that have not been written through to the data disk, or directly from the data disk. Thus, synchronous writes see almost no latency, and reads are generally unaffected, but multiple disks or multiple disk heads are required.

Track-aligned extents (traxtents) [15] are designed to minimize disk rotational latencies and track crossing overheads. The approach works by encouraging applications to make reads and writes in track-sized units. Smaller writes can be buffered until they grow large enough, making traxtents quite natural for pairing with log-structured

file systems. Since an entire track is being read at a time, the disk can begin reading as soon as it seeks to a track, as all of the data is passed to the application. The result of eliminating these rotational latencies can be up to 50% for mid-sized requests, though reads and writes much smaller than a track size do not benefit from this approach.

2.4 Our Approach

Our approach builds on most of the ideas of the systems reported here, combining them in a novel way, combining data replication, track alignment, eager-writing, and block cleaning. While traditional shadowing, mirroring, and striping techniques require multiple disk drives to improve performance, we improve performance of a single disk system.

Like Ng [12], we replicate data within a single disk track to reduce rotational latencies. While the work of Schindler et al [15] aims to align client application accesses to track boundaries, our goal is to align track boundaries to client application accesses by exploiting natural page-based workloads.

Similar to the eager-writing policies of Loge [8] and VLFS [16], we aim to eliminate write latencies by writing data to a location close to the head. In our case, however, we can improve upon existing eager-writing solutions, eliminating all rotational and seek latencies by overwriting data immediately underneath the disk head (providing another copy will still be available on the disk).

Under DDM writes must take the disk head into the slave partition to issue an eager write. Since our system does not partition the data in this way, our system does not require this seek on a write. Additionally, DDM must keep written blocks in memory until they are copied into the master partition (though an alternative system which keeps directory information and reads blocks from the correct partition is possible). Our system can be lazy and read bad tenants without having to hold them in memory or having to force them into their final location on the disk. Unlike our system, however, DDM manages to keep sequential read performance unhindered.

To our knowledge, our system is the first to use eager-writing principles to overwrite valid replicated data, and the first to tailor the disk track layout to page-based application semantics.

3 Design

3.1 System Architecture

Our overall system design is depicted in Figure 1. The core of the system is a paged disk drive. This disk drive allocates and manages blocks to maximize random read and write performance of page sized blocks of data. User applications, such as databases and virtual memory subsystems, interface with the paged disk drive through a paged file system, which provides a page based, instead of file based, view of storage. It is possible that performance gains will be realized just by simplifying the file system, but the focus of our investigation is the ability of a disk drive to minimize rotational and seek latency through the use of data replication and eager writing. The only requirement we make of the file system is that it allocate pages to aligned sequential logical block numbers. The file system interacts with the disk drive by making read and write requests of logical block numbers.

3.2 Paged Disks

A paged disk is designed to optimize performance of random reads and writes of pages. The disk defines pages to be any page-sized and page-aligned sequence of logical block numbers. For example, if the page size is 8k and blocks are 512 bytes, a page is any sequence of 16 blocks beginning at a block number that is a multiple of 16. A paged disk is optimized for paged-based random workloads in three primary ways:

- **Track-alignment**: Pages do not cross track boundaries.
- **Replication**: Pages are uniformly replicated within a track.
- **bad tenants**: Pages are written over whatever pages the disk head is currently over, if possible.

First, we ensure that pages are track-aligned and do not cross track boundaries. Since page reads and writes are relatively small, incurring a track-switching penalty in the middle of an access would be prohibitive. While this results in some wasted disk space, it is relatively insignificant. We find a mild performance benefit (< 1 %) realized by track aligning pages.

Second, reads are optimized by replicating pages uniformly around a track. The average rotational latency thus should decrease directly proportional with the *replication factor*, the number of copies of each page. On one hand, rotational latency is minimized when only one page is stored on each track (when the replication factor is $tracksize/pagesize$). On the other hand, as the replication factor increases the distance between pages increases, resulting in increasing seek times as well. Determining the optimal trade-off between these two contradictory factors will be essential to get good performance from paged disks.

Third, writes are optimized by the use of bad tenants, and is illustrated in Figure 2. In this scheme, a page P_3

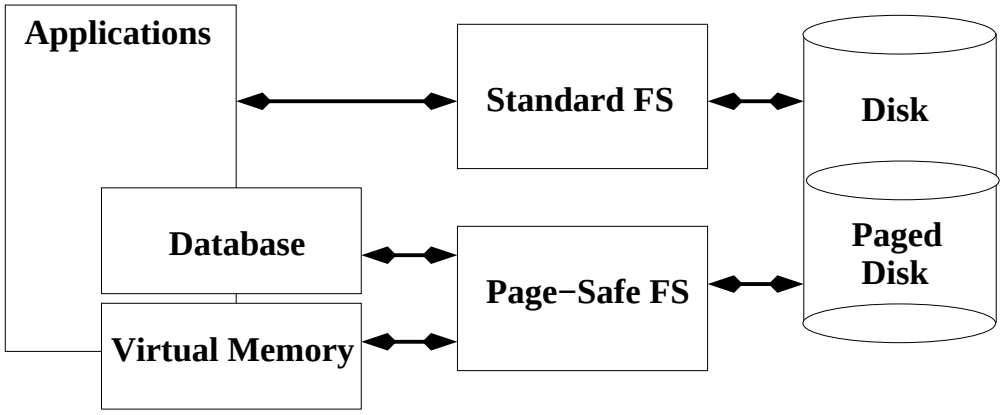


Figure 1: The overall system design.

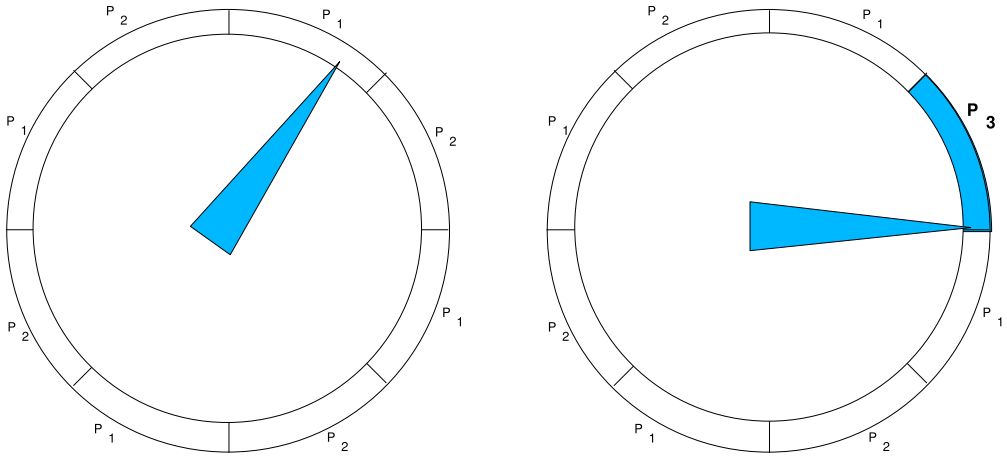


Figure 2: The layout of pages on a track before and after a write. In this diagram, the track contains 8 pages and P_1 and P_2 are replicated throughout the track. When a write of P_3 is made, the disk immediately writes P_2 to the page location closest to the current head location, overwriting valid, but replicated, data. Later a cleaner will move P_3 to its own track where the data will be replicated.

being written may be written overtop another page P_2, as long as another copy of page P_2 exists. This write of P_3 reduces the number of copies of page P_2 resulting in an increase to P_2's expected access time. We say that page P_3 is a "bad tenant" in the home of page P_2. The bad tenant may be placed in a given location as long as there is either (i) at least two copies of the page being overwritten, or (ii) no valid data currently stored in that location of the disk. Note that the disk may have sectors which are unallocated, since any replicas of a bad tenant page become stale when the bad tenant is written. These pages can be reused for any purpose.

In the ideal case, a write incurs nearly zero seek and rotational latency. Bad tenants improve write performance at the cost of decreasing the read performance of both the page being written and the page being overwritten. Writes will benefit even more as the replication factor increases, since more pages will be made available for overwriting.

As the number of bad tenants increases, the read performance will gradually degrade to the read performance of a traditional disk, provided the workload does not contain sequential accesses. Clearly, a workload with sequential requests will not perform well at all, as it may potentially seek randomly on the disk to access bad tenants. In order to limit degradation of performance due to bad tenants, a cleaner process periodically cleans and returns them to their normal replicated state.

To implement bad tenants, the disk must keep track of the number of copies of pages on the disk, the mapping of which page resides at a given physical location on the disk, and finally, the location of each bad tenant.

3.3 Design Parameters

The amount of replication of a page on a track can vary from a fixed small constant to full replication. On a modern disk drive, full replication would result in 25-50 copies of a page on each track depending on the zone, resulting in a corresponding 25-50 times decrease in addressable disk capacity. Although full replication would allow reads to incur zero rotational latencies, the corresponding increase in seek times outweigh the benefits. Therefore, it is necessary to find a "sweet spot" where the benefits of replication outweigh the negative side-effects.

There are many options for the bad tenant placement policy. The optimal choice for a bad tenant placement depends on a number of factors, including the seek and rotational latencies to the bad tenant as well as the future workload read and write requests (for instance, choosing to overwrite a copy of a frequently read data block may prove detrimental). Our choice is a simple policy. A bad tenant will be only be placed somewhere on the track which the disk head is currently over. It will choose the closest page which is either unallocated or holding a page which has more than one copy on the disk. Otherwise, the system falls back on a fully replicated write.

The cleaner is the most configurable component of the system. Various strategies can be used to determine when to clean: only during idle time, when a "dirtiness" threshold has been reached, during a write if it isn't too expensive, after a read if cleaning will improve future reads, or even as a nightly maintenance process. It is necessary to determine which pages will be cleaned and in what order. The location bad tenants are cleaned to is also flexible. Pages could always have home tracks that the cleaner will copy and replicate them to, or the cleaner could eagerly write to the nearest, fully available track. The current version of our system does not implement a cleaner process.

3.4 Recovery

A paged disk must maintain a mapping between logical and physical block locations. In order to recover from crashes, this mapping must in some way be made persistent. Although we have not implemented a recovery mechanism, we enumerate some possible designs here.

The mapping could be kept in a NVRAM buffer. When this NVRAM buffer gets filled, it would be written to a specially mapping log region on the disk. This would cause periodic interruptions in disk activity, but would allow very fast recovery of the mapping information. This is the strategy used by [18].

In addition to maintaining the mapping in a volatile memory, all pages written to the disk could include a small amount of meta data. Writing out the meta data would incur a trivial increase in transfer times and recovery would entail a full disk scan to reconstruct the mapping.

The mapping could be maintained in a volatile memory buffer, but this buffer could periodically be written out to disk. Pages would have a small amount of meta data associated with them which would include a timestamp. During recovery, a mapping could be quickly recovered from the mapping log region that would restore the disk to a consistent (but not most recent) state. Such an implementation would require that invalid pages not be overwritten until their mapping state had been written to the disk, somewhat constraining performance.

4 Implementation

We have implemented paged disks inside of DiskSim 3.0[2], a validated disk subsystem simulator. We have modified both its layout module, to support dynamic data placement mappings, and the disk controller to support the disk position awareness required when servicing requests on a paged disk. As a simplification, we do not

model bad or slipped sectors. This simplification only results in a negligible (less than .1%) change in the average behavior.

The implementation of the paged disk on the read path is straightforward. When the disk state machine makes a read request, it queries the layout manager for a logical-to-physical translation to initiate the seek. Once the seek has completed, it performs another query, this time providing the current disk head location. The layout component returns the copy of the desired page that is closest to the current location. If this page is currently a bad tenant, there will only be one such copy and a normal average rotational latency will be incurred.

The implementation of the paged disk on the write path is a bit more complex. When the seek is initiated, a call is made into the layout component to see if it is possible to remap the page being written to an available page on the current track. A page is available if there is a page on the current track which is either replicated elsewhere or has been overwritten by a bad tenant that is no longer valid. If such a page exists, no seek is incurred. Otherwise, the write does not become a bad tenant.

If the write can not be mapped to a bad tenant, a full replicated write to all of the available replicas takes place, resulting in a seek and rotational latency. The disk seeks to the home location of the page, and then attempts to write to all the replicas of that page on the track. If a replica has been overwritten by a bad tenant that is still valid, it can not be overwritten by this request. We maintain the invariant that at least one replica in the home location of every page is never overwritten by a bad tenant. The request is considered completed after the first replica, the replica closest to the disk head, is written. However, additional internal writes are issued to overwrite all available replicas resulting in additional rotational latency which, although not observable by the first request, will delay pending requests. By implementing this policy, we avoid clustering all the bad tenants in one region of the disk, even if the workload is characterized by long series of writes.

If the write can be mapped to a bad tenant, the closest available location that the disk head can service once the data is available is chosen. Because the disk will be ready to write the request almost immediately, it is necessary to incur some rotational latency at this point in order to wait for the transfer of data from the CPU to disk to finish.

5 Evaluation

It is often difficult to evaluate the performance of modifications to a disk, as the results depend on many factors, ranging from workload parameters to the design of the disk hardware. As a result, we attempt to paint a fair picture of paged disk performance trends by examining the behavior of a wide range of random and real-world workloads.

Throughout the rest of this section, we first describe the random and real-world workloads and simulation parameters considered in our experimentation. Second, we determine the optimal replication factor for improving read performance. Third, we compare the performance of paged disks to a standard disk. Fourth, we evaluate paged disks, RAID arrays, and standard disks and examine their performance as they evolve during their use.

5.1 Workloads

We focus on two types of workloads to evaluate our approach. First, we consider a family of randomly generated paged workloads, parameterized by the fraction of read/write requests. Second, we consider real-world workloads produced by the TPC-C database benchmark [6] running on the Shore Storage Manager [13].

Random workloads are generated by the built-in DiskSim workload generation module. The workload is configured to generate requests scattered randomly on the disk, each 8192 bytes, or 16 sectors long. For these random workloads, the requests are synchronous, meaning that the next request is generated and submitted to the disk only after the current request completed. According to the outcome of a Bernoulli trial with a fixed parameterized probability, each request is either a read or a write. We will refer to these workloads by the probability that a request is a write (for example, 10% writes). Two particular cases of importance are the read-only (0% writes) and write-only (100% writes) cases.

The real-world workloads considered in this paper are generated from execution traces of the TPC-C transactional database workload running on the Shore Storage Manager. For each read or write request issued by the storage manager, the trace contains a 4-tuple, including the time the request was issued, the offset of the volume, the length of the request, and a boolean indicating whether the request is a read or a write. The trace is generated by instrumenting the Shore `diskrw` process to log each read or write as they are issued. Each request generated by `diskrw` is already aligned and size to the size of a page (8k).

The TPC-C experiments presented in this paper are generated from a single execution run of the TPC-C benchmark. The trace is made up of 60000 read/write transactions, and consists of approximately 490000 read and write requests. Since the disk that the trace was collected on has transfer rates approximately 5 times faster than the simulated disk, we scale all inter-arrival times up by a factor of 5 to compensate (without doing this, the

DiskSim queues grow too large and the simulation halts prematurely). Note that the storage manager does significant buffering within its bufferpool, and most blocks are never read more than once during the trace. Writes, however, are almost always pushed down to the disk immediately to ensure ACID compliance.

Throughout our experimentation, the disk that DiskSim simulates is the `IBM 18ES`. The disk is an 8.7GB hard disk with 7200 RPMS and 5 surfaces. We make only a few changes to the disk for our experimentation. First, we use a "nosparing" block layout for simplicity of implementation. In addition, we disable sequential write combining and immediate reads as these features are inappropriate for paging workloads and nonsensical for a paged disk.

5.2 Replication Factor

Our thesis had been to layout pages on the disk such that each track holds many copies of only one page. The intended result was that rotational latency would be zero, and overall response times would decrease. Unfortunately, our original assumptions were fairly bonkers[1]. As the replication factor increases, data is spread over a wider area on the disk. Thus, as rotational latency is shrinking, seek times are increasing. It is thus important to discover the optimal replication factor to maximize performance.

Figure 3 depicts the breakdown of average transaction access times as a function of replication factor into seek, rotational, transfer, and replication times.

Figure 3(a) depicts the access time breakdown for the read-only random workload (1-way replication is similar to a standard disk, but it ensures pages never span track boundaries). Since each request is a read, there are never online costs for replicating data, and only seek, rotate, and transfer costs are involved. The graph clearly shows that as the replication factor increases, rotation decreases and seek increases. The optimal trade-off for this read-only workload is at 4-way replication.

Figure 3(b) depicts the access time breakdown for the write-only random workload without bad tenants, so each write must replicate multiple copies of the page. The seek, rotate, and transfer times are measured for writing the first copy of the replicated write. The replicas time is the amount of time needed to write the extra copies of the page onto the disk. Clearly, this is the dominating component as replication increases. While this replication time can be devastating for a disk-heavy application, an application that issues infrequent synchronous write requests never has to see the replication time, as control returns to the user immediately after the first copy is placed. Unfortunately, disk-heavy applications can suffer drastically from this extra replication cost.

[1] In the technical sense.

5.3 Bad Tenants

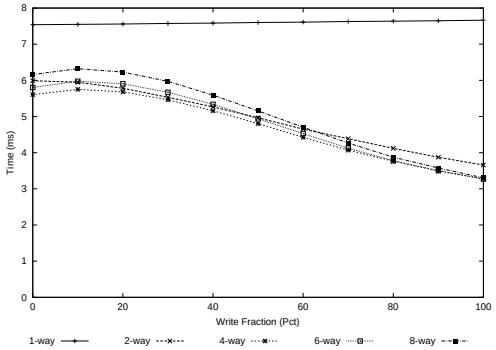


Figure 4: Comparison of access time for a paged disk with replication factors 1-way through 8-way as a function of read/write ratio. The 1-way numbers are similar to that for a standard disk.

As seen in Section 5.2, write performance, and in turn, overall disk performance, degrades significantly if the disk needs to maintain page replicas. The use of bad tenants not only alleviates, but completely eliminates, this problem of write performance.

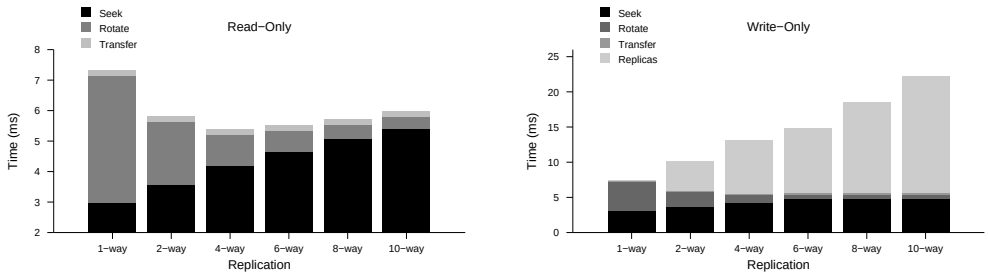
Figure 4 depicts the average access times for a paged disk with 1-way through 8-way replication as a function of the fraction of write requests in a random workload. 1-way replication performs similarly to a standard disk. Starting at a read-only workload, it is seen that 2- through 8-way replication provide a speedup of 1.3 times over the baseline 1-way replication. The average access times for replication levels above 2-way are all fairly similar. For a write-only workload, 4-way replication with bad tenants provides a 4.36 times speedup over the 1-way replicated disk. For a workload with 60% of writes, which is comparable to the TPC-C workload which we use in Section 5.4, the improvement is a speedup of 2.1.

As expected, as the write fraction increases, and bad tenant writes dominate the system performance. Overall access times improve dramatically since bad tenant writes incur almost no latency via seeks or rotates.

5.4 Real-World Comparison

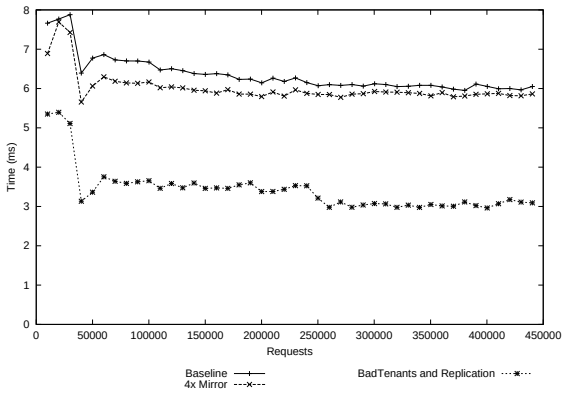
One of the concerns of a paged disk is that bad tenants can degrade performance. A bad tenant written over (a copy of) another page will increase the access times for that page, since there are now fewer copies of it. Likewise, since only one copy of the bad tenant is written, future reads of the bad tenant page will also be expected to be slow.

At the same time, however, a replicated page has n copies on the disk, and writing a bad tenant reduces that

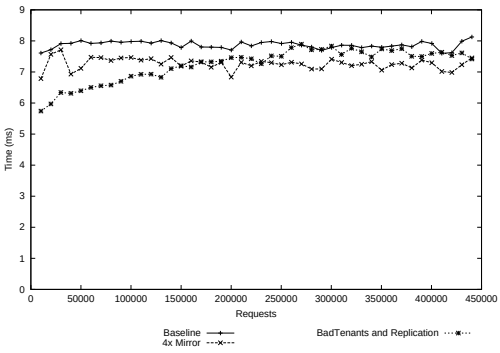


(a) Read-Only (b) Write-Only

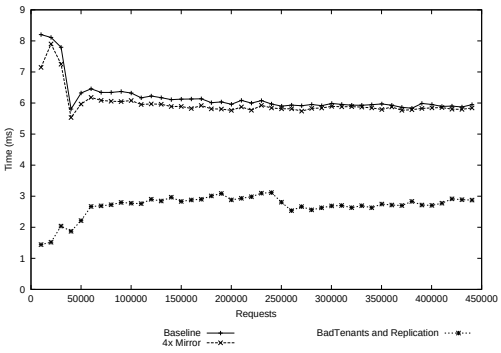
Figure 3: Breakdown of disksim average access time for read-only and write-only workloads as a function of replication factor.



(a) Read-Write

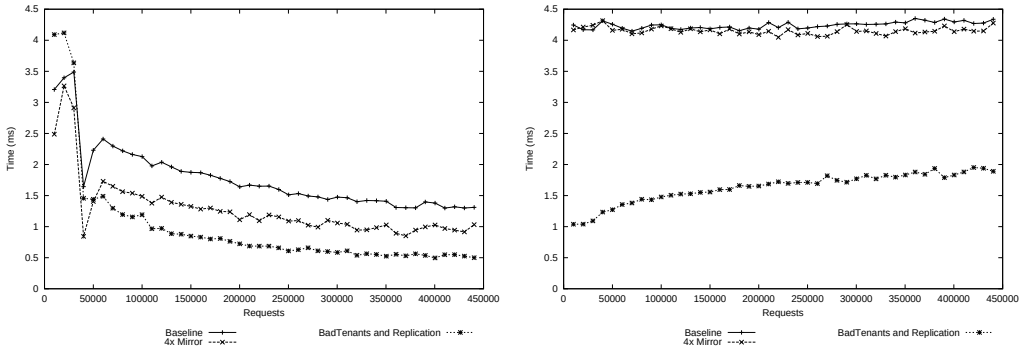


(b) Reads



(c) Writes

Figure 5: Average access times for a paged disk with bad tenants, 4x RAID Mirror, and a standard disk, as a function of the number of requests issued during a TPC-C trace.



(a) Seek (b) Reads

Figure 6: Average seek and rotational times for a paged disk compared to that of a 4x RAID Mirror and a standard disk, as a function of the number of requests issued during a TPC-C trace.

to 1 copy. Thus, there are $n-1$ pages that are unallocated by the first bad tenant write of a page. Each unallocated page can subsequently be used for holding more bad tenants, thus possibly reducing the performance degradation caused by bad tenants.

Figure 5 compares the average access time of a paged disk with bad tenants, a 4x RAID Mirror, and a standard disk, as a function of the number of requests issued for the TPC-C workload. Each sequential set of 10000 requests are averaged together to produce a data point in the figures.

Figure 5(a) shows the average access times for read and write requests together, whereas Figure 5(b) and Figure 5(c) show the access times for just reads and writes, respectively. The times for first 50000 requests or so are somewhat distorted, as at this point the database bufferpool is still being populated with data from the disk, and many new reads dominate the workload. After this point, however, a steady state is reached, as the bufferpool is filled and disk activity becomes more write-oriented. As noted earlier, the read-to-write ratio for our trace is approximately 40% to 60%. Clearly, the performance of the paged disk outstrips performance of both the standard disk as well as the 4x RAID Mirror, which uses 4 times as many disk drives as the paged disk solution.

Most interestingly, Figure 5(a) shows that access times for the paged disk are relatively constant, and in particular, Figure 5(c) shows that the performance degradation for paged disk writes is essentially negligible. Overall, access time has a speedup of about 1.35 times relative to standard, mostly due to the improvement in write performance. Read performance, on the other hand, depicted in Figure 5(b) is not as stellar. After around 200000 requests, the performance of paged disk reads is similar to that of the 4x RAID mirror, and after 300000 requests, it converges to the standard disk.

As a result, a bad tenant cleaner process may be needed to ensure that read-performance is improved. The cleaner would have to find enough idle time every 100000 requests to clean a significant portion of the 40000 pages during that interval. Since the total amount of work needed to clean a page for a given replication factor is similar to the write performance as seen in Figure 3(b), the idle time of the disk may have to be significant. In fact, it may need to be idle 70% of the time if the replication factor is 4 (If the disk is at 30% utilization, 15% utilization is writes, and 4 times the 15% is used to issue the replicas). It is not yet clear whether implementation of a cleaner will buy significant performance gains, since read performance was only a 1.33 times speedup to begin with.

Figure 6 depicts the average seek and rotational times for TPC-C disk requests as a function of the number of requests issued. As above, a paged disk with bad tenants (4 replicas) is compared with a 4x RAID Mirror and a standard disk. Figure 6(a) shows that the seek time for the paged disk decreases as a function of the number of requests, which suggests that the data is being clustered closer together on the disk due to bad tenants. The result is faster read seeks. The fact that the seek times are so much lower than that for the standard disk is due to the fact that write seeks on the paged disk are almost always zero. Figure 6(b) shows that rotational latency increases as time passes on the paged disk, which is expected, as the disk has to pass over more and more bad tenants on each track.

6 Future Work

Future work includes implementing a cleaner and exploring the trade-offs involved in its design and implementation. In particular, different workloads may show substantially different benefits from different cleaning strategies and the best strategy will vary from system to system.

In order for paged disks to become a reality, it is likely that the implementation will have to be removed from the disk controller and transferred to the file system. This way a single partition (such as the swap partition) of the disk could be used for paged accesses. In order for the file system to implement a paged disk, it would have to derive the disk properties[14], essentially reconstructing the physical layout of the disk.

An additional possible research direction would be to look at incorporating paged disks into a disk array. As it stands now, a paged disk should fit seamlessly into a traditional disk array. A more interesting system would be one in which the mapping is maintained by the disk array controller.

7 Summary

We have described our implementation of a paged disk, a single disk drive optimized for workloads characterized by random small accesses. Our page disk:

- prevents pages from spanning track boundaries to eliminate intra-transfer seeks
- replicates pages within a track to reduce the rotational latency incurred by reads
- eagerly writes pages to eliminate seek and rotational latency on writes

When compared against a traditional disk and a 4-disk array our paged disk outperforms a standard disk by a factor of about 4.5, and outperforms a 4x RAID Mirror (with 4 sets of disk hardware) by a factor of 2. Although read performance gradually decays with time, performance is still quite good for writes even after long sequences of operations, and we appear to do strictly better than a standard disk under all workloads.

8 Concluding Remarks (For Ganger's Eyes Only)

We're pretty darn pleased with the results we got, even though we never got around to implementing the cleaner. We are quite interested in advancing this work to a publishable state, as the results look particularly promising. The most essential failure, we believe, is that we fail to compare against an existing "Eager Writing" disk, which may perform nearly as well as bad tenants.

One of us (David) would like to profusely apologize for confusing your youngest offspring with the kid from the Life cereal commercials.

References

[1] Dina Bitton and Jim Gray. Disk shadowing. In *Proceedings of the Fourteenth International Conference on Very Large Databases*, pages 331–338, 1988.

[2] John S. Bucy, Gregory R. Ganger, and et al. The disksim simulation environment version 3.0 reference manual. Technical Report CMU-CS-02-180, Carnegie Mellon University, September 2002.

[3] C. Choa, R. English, D. Jacobson, A. Stepanov, and J. Wilkes. Mime: a high performance parallel storage device with strong recovery guarantees, 1992.

[4] Tzi cker Chiueh and Lan Huang. Track-based disk logging. In *International Conference on Dependable Systems and Networks (DSN'02)*, pages 429–438, 2002.

[5] J.Solworth C.Orji. Doubly-distorted mirrors. In *Proceedings of ACM SIGMOD*, 1993.

[6] Transaction Processing Performance Council. TPC benchmark C. Number Revision 5.1.0, December 2002.

[7] Y. Dishon and T. S. Lui. Disk dual copy methods and their performance. In *Proceedings of Eighteenth International Symposium on Fault-Tolerant Computing (FTCS-18)*, pages 314–318, 1988.

[8] R.M. English and A.A. Stepanov. Loge: a self-organizing storage device. In *In Proceedings of USENIX Winter'92 Technical Conference*, pages 237–51. USENIX, January 1992.

[9] R. Hou and Y. N. Patt. Trading disk capacity for performance. In *Proceedings of the Second International Symposium on High Performance Distributed Computing*, pages 263–270, 1993.

[10] S.L Lo. Ivy: A study on replicating data for performance improvement. Technical report, Hewlett-Packard Company, 1990.

[11] Christopher Lumb, Jiri Schindler, Gregory R. Ganger, Erik Riedel, and David F. Nagle. Towards higher disk head utilization: Extracting "free" bandwidth from busy disk drives. In *Proceedings of*

the 4th Symposium on Operating Systems Design and Implementation (OSDI), pages 87–102, October 2000.

[12] S. W. Ng. Improving disk performance via latency reduction. *IEEE Transactions on Computers*, pages 307–316, January 1993.

[13] University of Wisconsin. Shore - a high-performance, scalable, persistent object repository. `http://www.cs.wisc.edu/shore/`.

[14] J. Schindler and G.R. Ganger. Automated disk drive characterization. Technical Report CMU-CS-99-176, Carnegie Mellon University, December 1999.

[15] J. Schindler, J. Griffin, C. Lumb, and G. Ganger. Track-aligned extents: matching access patterns to disk drive characteristics, 2002.

[16] Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. Virtual log based file systems for a programmable disk. In *Operating Systems Design and Implementation*, pages 29–43, 1999.

[17] Xiang Yu, Benjamin Gum, Yuqun Chen, Randolph Y. Wang, Kai Li, Arvind Krishnamurthy, and Thomas E. Anderson. Trading capacity for performance in a disk array. In *Proceedings of the 2000 Symposium on Operating Systems Design and Implementation*, pages 243–258, San Diego, 2000. USENIX Association.

[18] Chi Zhang, Xiang Yu, Arvind Krishnamurthy, and Randolph Y. Wang. Configuring and scheduling an eager-writing disk array for a transaction processing workload. In *Proceedings First Conference on File and Storage Technologies*, January 2002.

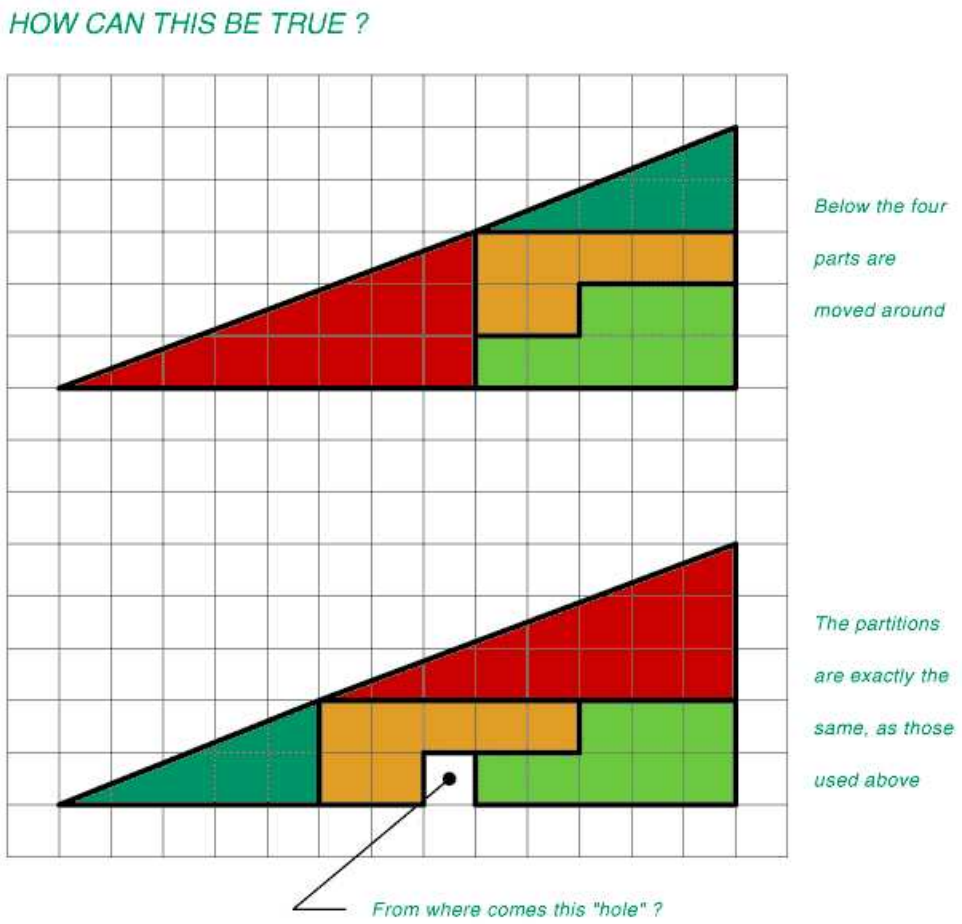


Figure 7: You might think that layout is completely meaningless, but we have conclusively shown that layout is one of the most important factors in disk performance. Through our channeling of the ancient, but cool, daddios of systems research, we have managed to squeeze the extra 1 unit block out of paged disk performance.