Reducing Disk Rotation Delays using Range Writes

Rokas Venckevicius
Mayank Maheshwari
Department of Computer Sciences
University of Wisconsin, Madison

December 20, 2007

Abstract

Disk I/O is generally considered the bottleneck in memory operations. This is generally due to mechanical limitations (limited disk rotation caps transfer bandwidth), and the general architecture of modern disks (rotating platters accessed by a single head on the arm), which causes a delay on the positioning of the head. Operating systems are generally able to fight the seek delays by writing to blocks not far away from each other, but tackling the rotational delay problem is much more complicated. We propose a method to fight the disk rotational delays, dubbed Range Writes, *by allowing for the operating system to give a wider flexibility to disk to perform a write, potentially saving significant rotational delays. Instead of providing just one logical block, in Range Writes the file system can provide multiple blocks for disk to write data to, thus allowing the disk to make a smart decision and save some rotational delay. We provide an implementation of Range Writes on Disksim disk simulator, and show the savings in rotational delays and total positioning time. Our experiments show up to 32% improvement in rotational delays for write-intensive workloads.*

1 Introduction

The time overhead of disk accesses is one of the significant performance bottlenecks in the storage systems. Traditionally, secondary storage has been accepted to be at least two orders of magnitude slower than memory and systems have been built around this premise. Caching of frequently accessed data and lazy write back policy are some examples where policy is dictated by the constraint of high disk access time. Over the years, the access time in modern drives has reduced considerably due to architectural improvements in disk technology and implementation of scheduling algorithms which reduce seek time. The file system also writes files to the disk in such a way as to reduce the disk I/O time. The UNIX Fast File System is one example of this spatial locality [1].

While a lot of improvements have been made to reduce the disk seek time, little has been done to improve rotational latency in disks. Since in earlier disks, rotational latency was only a small percentage of the total disk access time, sufficient attention was not given to reduce it. But in modern disks, with seek time reduced considerably, the rotational delay has become a significant portion of the delay in disk access time. In this paper, we have tried to address the issue of rotational delay incurred by disks, by providing more options to the controller in terms of a range of blocks it could write the data to, instead of a single logical-to-physical mapped block.

The improvisation we propose is called *"Range Writes"* in which the file system provides a range of logical blocks to the disk controller to write to which maps to contiguous physical blocks on a track on the disk. Based on the current disk head position, the controller estimates which block among the range of blocks provided will incur the least rotational delay and picks that block to start writing data. In case the disk head points to a block in the range, there is no rotational delay incurred, otherwise the controller picks the block closest to the disk head in the range. We have implemented the idea for small random writes but it can also be extended to allow flexible reads (called *"Flex-reads"*) where given information about the replicated copies of file, the controller can decide which copy to read from depending on the least positioning time.

We used the disk simulator *Disksim* [2] for our experiments and observe an improvement of up to 32% in the rotational latency incurred and a savings of 16% in the disk positioning time. Disksim simulates some old disk models and so the positioning time improvement is not as significant but we show that modern disks with lesser average seek time will get much better improvements in positioning time with Range Writes.

The rest of the paper is organized as follows. Section 2 discusses the performance problems with disks and the limitations to improving them. Section 3 talks about the idea of Range Writes in more detail and how it can be extended for reads. Section 4 outlines our implementation using Disksim. Section 5 gives details of our experiments and results. Related work is discussed in Section 6, further work in Section 7 and we will conclude in Section 8.

2 Disk Performance Problems

On a rather coarse level, each disk operation consists of three major parts: seek delay, rotational delay and the actual read/write operation. To improve disk performance, we would need either to reduce one of the delays or speed up the actual read/write operation.

The actual read/write operation is closely coupled with the rotation speed of the disk and the density of segments in each track. While platter density has been constantly increasing in the previous years, the rotation speed has been fairly steady throughout. Furthermore, the file system generally has very little to do with improving the bandwidth (except for not adding additional overhead and providing a contiguous sequence of blocks to write data to). Improving the actual disk bandwidth is generally more related to improvements in hardware.

What the file system can do, however, is try to fight the delays introduced by disks. Traditionally the delays are simply *dead weight*, that is, they do not add in any way to the speed of disk transfer, yet every I/O operation has to incur them. While this is not a problem for large, contiguous I/O, it quickly becomes a problem for small I/O operations, since the relative time spent on delays becomes a high proportion of the total time spent to perform the I/O. File systems have traditionally attempted to fight seek delays by issuing write requests to blocks that have some locality [1]. Under this lies an assumption that blocks are mapped into disk's sectors, tracks and cylinders in a linear manner, but it has proved quite effective in real systems aware of this structure [1].

On the other hand, fighting rotational delays is much harder for the file system. A rotational delay is defined to be the time that passes after the arm has finished seeking to the right track, but before the write (or read) can actually begin. The reason why it occurs is that the head may be at a different position on the track than the block that needs to be written to. Therefore the platter may have to spin a whole revolution (in the worst case) until the head actually reaches the block that it needs to start writing to. How long is one revolution exactly? That depends on the disk and the rpm's of its platters. A typical desktop system disk in 2007 rotates at 7200 rpm [3], which translates to approximately 8.3 ms for a single revolution. If the workload is many small writes, this may add a substantial delay and slow-down of the already slow I/O operations. Furthermore, file systems have no way of performing optimization in providing the block to which the disk should write to, since they do not have access to the logical to physical block number mappings, nor are they able to get the current position of the disk head. Therefore in the best case any FS can simply guess a block for writing to, without knowing how much the rotational delay will be (except for the upper bound of 8.3 ms). We believe this could be solved to some extent with Range Writes.

3 Range Writes

The idea of Range Writes inherently is to provide more options to the disk controller for writes so that it can optimize the operation based on the information it has of the disk state. Since the controller has information about the current head position, it can estimate the positioning time and transfer time of the write. By providing a range of blocks or more than one discrete blocks equidistant from each other, the controller can estimate the rotational delay (if the disk head does not currently lie in the range) for the starting block of the range and select the block with the least delay for the write. After writing, the controller returns the block number of the data to the file system.

In our implementation of Range Writes, we have provided a fixed number of discrete equidistant blocks to the controller for writes. We have taken this approach for two reasons. One is for the purpose of simplification and the other is, since we are optimizing for a workload consisting primarily of small random writes, providing a range of blocks would require that the physical contiguous blocks on a track should be free. It would entail keeping contiguous blocks empty on the track to accommodate range writes while the actual data size is small. This would limit the size of the range that can be provided and as the start data block may not be the starting block of the range (if the head position lies in the range), this could cause fragmentation of the track.

With discrete equidistant blocks, the controller can write data starting from any of these blocks based on least rotational delay and return the block number to the file system where the data is written. We could optimize for the number of discrete blocks which reduce the rotational delay while keeping track fragmentation at bay. With large modern drives, the effects of fragmentation might not be apparent immediately but if the workload consists of a large number of small writes, it will manifest itself in the form of increased latency.

Range Writes requires modifications to the interface between the disk and the file system. It propagates the control to the disk to make a decision on the position of data block out of the options provided to it by the file system. So, in a way, the idea is to make the disk smarter and file system more disk aware. The disk controller is the lowest level abstraction and by providing it with decision-making capabilities, it can make the most informed decision about read/write based on the state of the disk. One crucial point about our approach to Range Writes is that it itself is not the scheduler but can be thought of as the *scheduler's friend*. The requests arriving to the disk are still scheduled according to the existing scheduling algorithm implemented in the disk. Range Writes module decides the final block from among the block options provided for the scheduled request to complete the write. A benefit of our approach is that the scheduler need not be modified to implement range writes and it can be done independent of the scheduling algorithm underneath.

The concept of Range Writes can also be extended to reads *i.e.*Flex-reads in which the file system provides information about all the replicated copies of the file to be read to the disk controller. The controller using the Flex-reads module and the information about the current disk head position estimates the access time for each replica and decides on the one with the least access time. The approach has overtones of the Shortest Positioning Time First (SPTF) algorithm and we used some of its components in our implementation but our main focus is on reducing the rotational latency and not the overall positioning time.

One of the possible limitations of this idea is that it works well for writes where the file system provides the potential blocks on the same track and there is no extra seek time incurred. In case of reads however, the replicated copies of the file will with high probability, not be on the same track of the cylinder and so the positioning time might have a bearing on the optimality of the operation. In our opinion, existing modern drives have been optimized for low seek times and the commonly used scheduling algorithms are devised to reduce seek. So even for reads, rotational latency will be the bottleneck for disk access and Flex-reads should be able to provide a significant improvement. However, this conjecture requires further analysis and experimentation and we do not concentrate on reads in this paper.

We analyze our implementation for different number of discrete optional blocks provided and observe that the savings in rotational delay is as expected with the disk head being in any position on the track uniformly at random. It is discussed with experiments and results in detail in Section 5. The other question worth pondering is whether the idea of Range Writes is really beneficial? The next section discusses this and our implementation in further depth.

4 Implementation

In the implementation of Range Writes, we have used Disksim to simulate reads and writes on a disk and get operation statistics in the form of number of events simulated, average seek and rotational latency, response times, inter-arrival request statistics etc. For our purposes, we concentrate on the average seek and rotational time and transfer time across all disks in the system. Our implementation includes modifying Disksim to allow for the Range Writes interface and accept inputs with more than one discrete block numbers specified. We also introduced modules for Range Writes in Disksim alongside the scheduler to pick the most efficient block in terms of low rotational latency to write to.

Our implementation with only two discrete blocks shows an improvement of up to 32% in reducing rotational delay and a 16% reduction in positioning time. This gives an indication of the possible improvements in disk access time we could achieve by providing more number of discrete blocks as options. Mathematically, the expected savings would be of the order of $3/8$ *i.e.*37.5% of a revolution and so 75% improvement over average rotational delay without range writes if we provide four discrete blocks. Before we go into the details of our implementation, let us give a brief background of Disksim.

4.1 Disksim

Disksim is one of the most widely used disk simulators in the systems community. It provides disk specifications and simulates some old disk models like Cheetah and Barracuda variants of Seagate disks and some versions of HP disks [2]. It takes as input trace files which abstract the read/write requests sent by the file system. Disksim identifies certain trace formats based on the disk model the simulation uses and outputs the statistics in terms of events simulated, transfer time, access time, seek distance and seek time, rotational latency and other parameters across all devices used in the simulation.

The components of Disksim can be broadly classified into three different sets of files. One is the disk model specifications and parameter files which decide various disk parameters like the scheduling algorithm, cylinder mapping strategy etc. of the simulation. The second set consists of disk sub-system files which simulate the actions of the scheduler, disk controller, bus and I/O driver modules. The third is the set of input trace files which abstract the file system read/write requests and consist of fields like the time at which the request was issued, logical block numbers and device number to which the request is to be directed.

4.2 Our Approach

In simulation on Disksim without range writes, the trace files provide the logical block numbers where the data is to be written or read from. The disk controller maps these logical block numbers uniquely to physical addresses. All the requests are added into the scheduler queue and the scheduler breaks it into sub-queues and orders requests in each sub-queue independently. It schedules the requests using the scheduling algorithm as specified in the parameter file of the disk model and calculates the operation statistics to output.

In our version of Disksim, we modified the interface to allow Disksim to accept a new trace scheme which provides additional block numbers as options to the controller along with request time and device number. The controller performs the same operations as before and the scheduler orders the requests in the sub-queue according to the underlying algorithm. After the requests have been scheduled, the Range Writes module estimates the positioning time of each of the optional blocks provided with the request structure in the trace file. Since our main focus in this paper is to reduce rotational latency in writes, the optional blocks provided are restricted to be on the same track. So the seek time for all the blocks in the request is essentially the same. We would be using positioning time and rotational latency interchangeably since in our case, rotational latency decides the block with the least positioning time. It should be noted here that our design goal for Range Writes is to cater to workloads comprising mainly of small random writes. So the idea of all the optional blocks in the request being on the same track is feasible and bodes well for our implementation. Range Writes picks the block with the least rotational latency and proceeds to calculate the access time and transfer time statistics for the request. A large number of requests (of the order of 10000) are simulated and the average seek time and rotational latency calculated to reflect the performance of the disk.

The estimation of positioning time for the blocks provided with the request is done using the *device_get_servtime()* function in Disksim which is the difference of access time and transfer time. The positioning time in Disksim comprises of the seek time and the initial rotational latency. It also includes the time required for head switch and additional write settling time in some cases to incorporate the real-world disk latency.

In the Range Writes scheduler, we have restricted ourselves to using First Come First Served (FCFS) algorithm for ordering the requests in the sub-queue. This was partly to ensure simplicity and partly because our main goal was to reduce rotational latency with existing scheduling algorithm and almost all disk models in Disksim surprisingly used FCFS for scheduling their requests. We had also considered more commonly used scheduling algorithms in modern drives like C-Scan and Elevator for Range Writes but both these algorithms order requests to optimize for seeks and in our case, all the additional blocks provided are on the same track and hence, have same seek times. So these algorithms provide us with the same level of improvement for range writes as FCFS. Some other algorithms like Shortest Positioning Time First (SPTF) though, would require global ordering in the sub-queue based on the rotational latency of all the additional blocks provided with the request structure and would require a slightly different implementation. We have not considered SPTF in the existing version but investigating the effect of some other scheduling algorithms will form part of our future work on range writes.

Our implementation also ensures that the scheduler for disk model need not be changed. In our existing version, we have made Range Writes independent of the scheduler and it decides on the optimal block only after the request order has been determined by the scheduling algorithm. This provides us some ease of implementation in terms of not requiring to understand the disk model and how it does scheduling? The other benefit would be to implement it rather easily on real disks without having to change the firmware code for scheduling on the disk.

5 Experiments and Results

To show the actual benefits of Range Writes, we conducted a set of experiments to reveal certain features about the savings we achieve in rotational delay. The purpose of these experiments is to illustrate in what conditions Range Writes is beneficial, and what kind of setup we need to use to achieve maximum reduction of rotational delay. We attempted to conduct these experiments using scientific principles, we therefore provide a hypothesis and run an experiment to validate or invalidate the hypothesis.

For all experiments we used Disksim to simulate a disk, and an *HP C2249A* disk conveniently provided to us with the distribution of disksim.

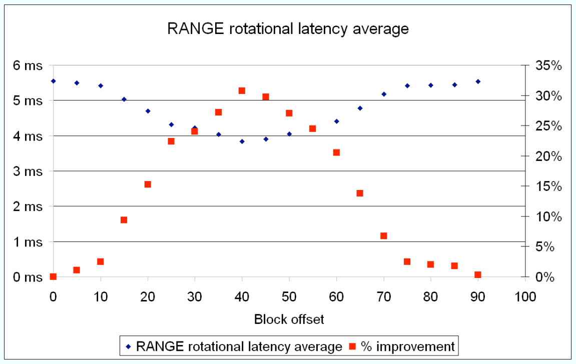

Figure 1: **Range Rotational Latency Average** *Percentage Improvement in Average Rotational Latency using Range Writes*

5.1 Experiment 1

In this experiment, we assume that there is another alternative given for a range write. Our trace consisted of $10,000$ write requests each of 10 blocks in size. The request were arriving at a constant rate. The disk was using an FCFS scheduler. All the requests arriving were to a single zone of the disk (the first one), that has 96 blocks per track and a total of around $50,000$ blocks. The purpose of the experiment is to figure out the offset that an alternative block should get, given an original block. We hypothesize that the offset should be around $1/2$ of the blocks per track, so around 48. This makes sense intuitively: if we provide an alternative block to write to that is on the other side of the track than an existing block, we would on average get the highest savings in rotational delay. Figure 1 shows the results of Experiment 1.

From the results in Figure 1, we see that the average savings in rotational delay reach about 32%. We would think the savings would be

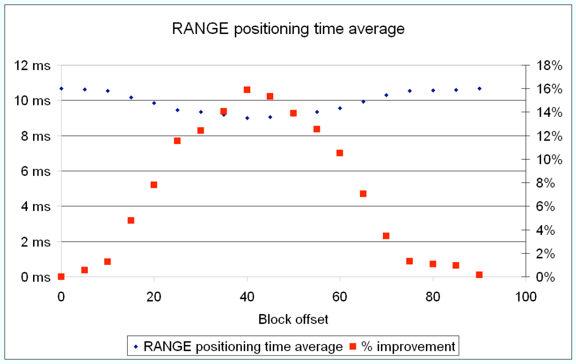

Figure 2: **Range Positioning Time Average** *Percentage Improvement in Average Positioning Time using Range Writes*

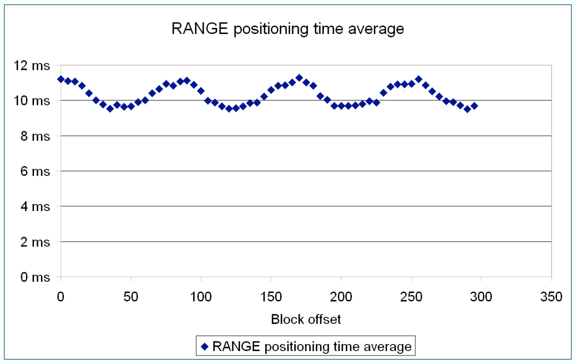

Figure 3: **Range Positioning Time Average** *Average Positioning time with Range Writes with respect to Block Offsets*

around 50%, however in our experiments we are not achieving this figure. We hypothesize this is due to the way Disksim is set up and our incomplete knowledge of the inners of its use of determining the expected rotational and seek times. At the same time, the highest savings come at offsets $40 - 45$, which is close to half of the zone's number of blocks per track. Thus our intuition is verified.

Furthermore, in Figure 2, we see the savings in total positioning time. These savings are more modest, as they include the seek time, which we are not optimizing for. From the results we see, that with one alternative block, we get savings up to 16%. We hypothesize that on contemporary disks the savings would be higher, as the seek time tends to improve faster in current disks than does rotation speed, which directly affects the efficiency of range writes.

5.2 Experiment 2

Based on the results from our previous experiment, we set up another experiment to see if we can determine the pattern of savings in total delay if with different offsets. We thus run the experiment with a similar set-up as Experiment 1, except we vary the offset not only on a single track (around $90 blocks$), but through more than one track. What we end up seeing is something similar to a sine curve:

Does the result in Figure 3 make sense? It certainly does, since the dips in the curve appear in regular intervals, and that is on average when we are on the opposite side of each track on a platter. The peaks appear when we have two blocks sitting on adjacent tracks, but essentially in the same radial position. In this situation, only extra seek time can be saved, but not rotational delay.

6 Related Work

The idea of reducing rotational latency in disks has been around for a while but it was thought of in terms of scheduling algorithms like variants of Shortest Positioning Time First and Shortest Access Time First. Our approach to improving upon rotational latency with Range Writes is however, different in respect that it does not try to change the scheduling algorithm underneath. It aids the scheduler to decide the block which would incur the least rotational delay once the requests have been scheduled. Algorithms like SPTF have not been implemented commonly in disks because of the issues of starvation and time overheads. Range Writes does not suffer from these limitations and can be modeled on top of any scheduling algorithm.

There are also some new approaches for scheduling suggested that perform online simulation of the underlying disk. Disk Mimic [4] is another simulator which performs scheduling using Shortest Mimicked Time First algorithm to schedule based on the information of the state of the underlying disk. Range Writes is a rather different approach to optimizing disk accesses by providing options, though it does use the information about the current head position of the disk.

7 Further Work

Further work provides multiple possibilities to improve our implementation of range writes, given what we've learnt and given more time. Our current implementation of disksim is scheduler independent *i.e.*we provide alternative block numbers to write to, and the disk pretends that it can only see the first block for scheduling purposes. Once the block is actually scheduled, it gets expanded into multiple alternative blocks, and the one with the lowest rotational delay get picked. This, however, is not optimal. We might do the block expansion before even scheduling ever happens. We believe implementing this would be more complicated and require more time and effort. This algorithm would expand the request into three different request before even scheduling ever happens, then goes through the queue and picks the request with the smallest rotational delay and writes to it. This would effectively guarantee that we're writing into a global minimum rotational delay block as opposed to the ”local” one that we have in our implementation.

Another area for future work would be to instead of giving a fixed number of blocks to the disk, we could provide with an actual range of blocks (indicating the first and last elements). We realized that this route could be more efficient already toward the end of our modifications to disksim, therefore we never had the chance to run it. We do believe, however, that it could simplify a great deal of our calcluations and communication. We would also like to investigate the effect of other scheduling algorithms like SPTF etc. with Range Writes and observe the improvements they provide as compared to FCSF or C-Scan.

8 Conclusions

Disks are the slowest part of a computer memory sub-system, and rotational delays is one of the causes of their slowness. It tends to delay the disk's total head positioning time, and opearting systems are generally unable to fight these delays as effectively as the seek delays. this is especially a problem in workloads with many small writes. To solve this problem, we have proposed an implementation of Range Writes, which allows for the operating system to provide multiple blocks to disk, and let the disk choose which one to write to, based on the smallest rotational delay. This provides improvements in rotational delay up to 30% in the simplest case, and more in the more complicated set ups. Are these results meaningful? They certainly apply for mainly a specific narrow workload - many smal new file writes, where the rotational delay takes as much time as the actual write to disk. However even in general we can conclude that by making the OS more disk aware, we can reduce the rotational delays. And while the results of the total positioning time delay seem not so effective for older disks, we conjecture that modern disks, where the seek time on average is lower than rotational delay, Range Writes would provide even further improved performance.

References

[1] Marshall K. McKusick, William N. Joy, Samuel J. Leffler, Robert S. Fabry A Fast File System for UNIX. *ACM Trans. Comput. Syst. (TOCS) 2(3):181-197 (1984).*

[2] Disksim Simulation Environment V3 manual *http://www.pdl.cmu.edu/DiskSim/ (in Dec. 2007).*

[3] Bruce L. Worthington, Gregory R. Ganger, Yale N. Patt Scheduling Algorithms for Modern Disk Drives. *SIGMETRICS 1994: 241-252.*

[4] Florentina I. Popovici , Andrea C. Arpaci-Dusseau , Remzi H. Arpaci-Dusseau Robust, Portable I/O Scheduling with the Disk Mimic. *Proceedings of the USENIX Annual Technical Conference (USENIX '03) pages 297–310 June 2003 San Antonio, Texas .*