CS304 Assignment-2

This assignment focuses on two topics Hard disk drives and Redundant Array of Inexpensive Disks (RAID)

You need to use the disk.py and get familiarized with how a modern hard drive works and answer the first 10 questions.

You need to use raid.py which models a simple RAID simulator to answer the set of next 8 questions.

Deadline : 11th April 11.59PM

1. Compute the seek, rotation, and transfer times for the following sets of requests: `-a 0`, `-a 6`, `-a 30`, `-a 7,30,8`, and finally `-a 10,11,12,13`.
2. Do the same requests above, but change the seek rate to different values: `-S 2`, `-S 4`, `-S 8`, `-S 10`, `-S 40`, `-S 0.1`. How do the times change?
3. Do the same requests above, but change the rotation rate: `-R 0.1`, `-R 0.5`, `-R 0.01`. How do the times change?
4. FIFO is not always best, e.g., with the request stream `-a 7,30,8`, what order should the requests be processed in? Run the shortest seek-time first (SSTF) scheduler (`-p SSTF`) on this workload; how long should it take (seek, rotation, transfer) for each request to be served?
5. Now use the shortest access-time first (SATF) scheduler (`-p SATF`). Does it make any difference for `-a 7,30,8` workload? Find a set of requests where SATF outperforms SSTF; more generally, when is SATF better than SSTF?
6. Here is a request stream to try: `-a 10,11,12,13`. What goes poorly when it runs? Try adding track skew to address this problem (`-o skew`). Given the default seek rate, what should the skew be to maximize performance? What about for different seek rates (e.g., `-S 2`, `-S 4`)? In general, could you write a formula to figure out the skew?
7. Specify a disk with different density per zone, e.g., `-z 10,20,30`, which specifies the angular difference between blocks on the outer, middle, and inner tracks. Run some random requests (e.g., `-a -1 -A 5,-1,0`, which specifies that random requests should be used via the `-a -1` flag and that five requests ranging from 0 to the max be generated), and compute the seek, rotation, and transfer times. Use different random seeds. What is the bandwidth (in sectors per unit time) on the outer, middle, and inner tracks?
8. A scheduling window determines how many requests the disk can examine at once. Generate random workloads (e.g., `-A 1000,-1,0`, with different seeds) and see how long the SATF scheduler takes when the scheduling window is changed from 1 up to the number of requests. How big of a window is needed to maximize performance? Hint: use the `-c` flag and don't turn on graphics (`-G`) to run these quickly. When the scheduling window is set to 1, does it matter which policy you are using?
9. Create a series of requests to starve a particular request, assuming an SATF policy. Given that sequence, how does it perform if you use a **bounded SATF** (**BSATF**) scheduling approach? In this approach, you specify the scheduling window (e.g., `-w 4`); the scheduler only moves onto the next window of requests when *all* requests in the current window have been serviced. Does this solve starvation? How does it perform, as compared to SATF? In general, how should a disk make this trade-off between performance and starvation avoidance?
10. All the scheduling policies we have looked at thus far are **greedy**; they pick the next best option instead of looking for an optimal schedule. Can you find a set of requests in which greedy is not optimal?

1. Use the simulator to perform some basic RAID mapping tests. Run with different levels (0, 1, 4, 5) and see if you can figure out the mappings of a set of requests. For RAID-5, see if you can figure out the difference between left-symmetric and left-asymmetric layouts. Use some different random seeds to generate different problems than above.
2. Do the same as the first problem, but this time vary the chunk size with `-C`. How does chunk size change the mappings?
3. Do the same as above, but use the `-r` flag to reverse the nature of each problem.
4. Now use the reverse flag but increase the size of each request with the `-S` flag. Try specifying sizes of 8k, 12k, and 16k, while varying the RAID level. What happens to the underlying I/O pattern when the size of the request increases? Make sure to try this with the sequential workload too (`-W sequential`); for what request sizes are RAID-4 and RAID-5 much more I/O efficient?
5. Use the timing mode of the simulator (`-t`) to estimate the performance of 100 random reads to the RAID, while varying the RAID levels, using 4 disks.
6. Do the same as above, but increase the number of disks. How does the performance of each RAID level scale as the number of disks increases?
7. Do the same as above, but use all writes (`-w 100`) instead of reads. How does the performance of each RAID level scale now? Can you do a rough estimate of the time it will take to complete the workload of 100 random writes?
8. Run the timing mode one last time, but this time with a sequential workload (`-W sequential`). How does the performance vary with RAID level, and when doing reads versus writes? How about when varying the size of each request? What size should you write to a RAID when using RAID-4 or RAID-5?