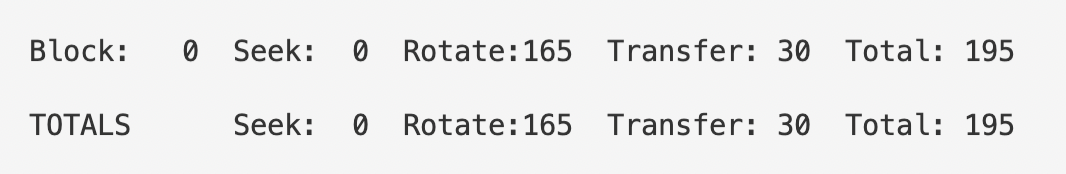
**Cpt 37**

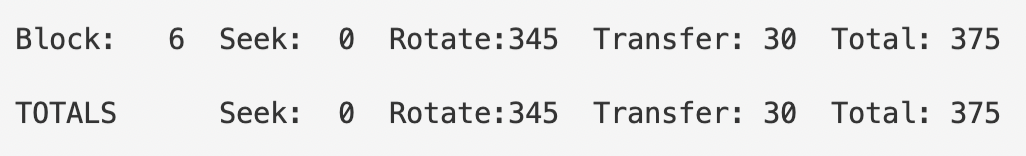
**Homework (Simulation)**

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.**

python3 disk.py -a 0 -c



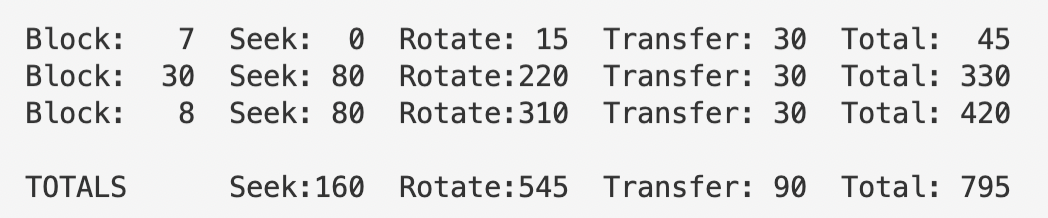
python3 disk.py -a 6 -c



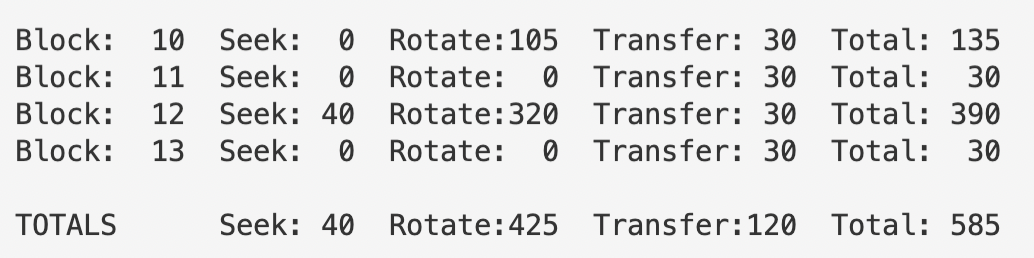
python3 disk.py -a 30 -c



python3 disk.py -a 7,30,8 -c



python3 disk.py -a 10,11,12,13 -c



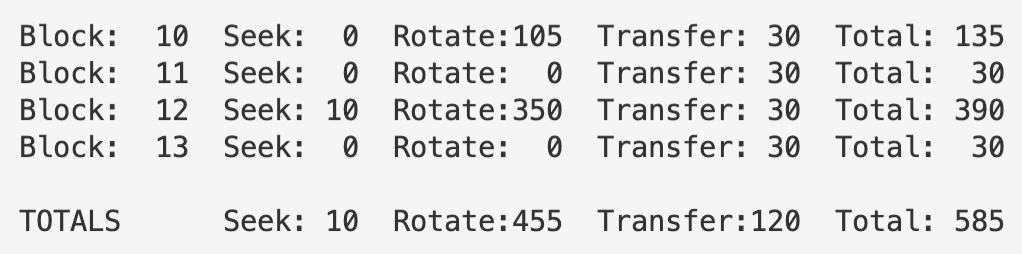
1. **Do the same requests above, but change the seek rate to different values: -S 2, -S 4, -S 8, -S 10, -S 40, -S 0.1. How do the times change?**

Take python3 disk.py -a 10,11,12,13 -c as an example:

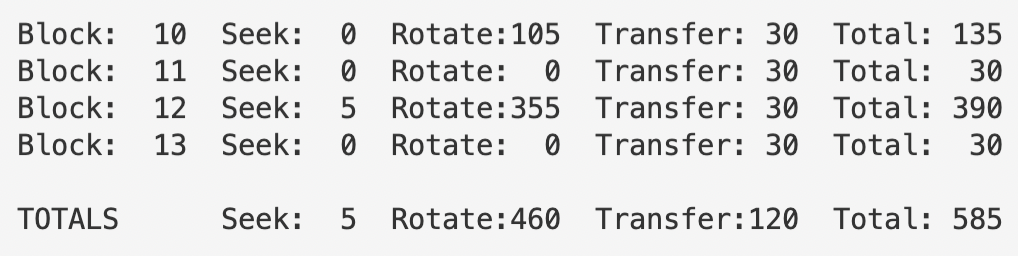
python3 disk.py -a 10,11,12,13 -c -S 2



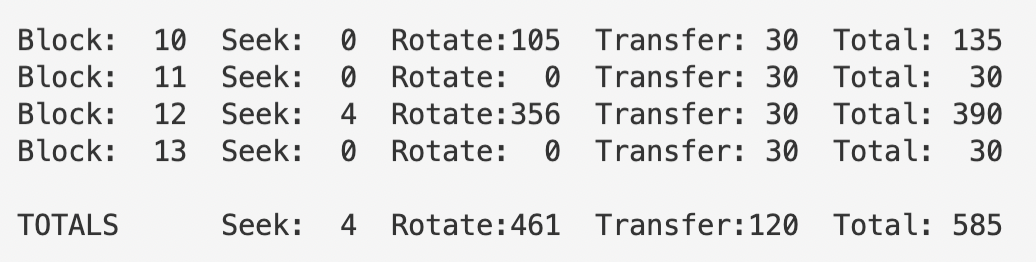
python3 disk.py -a 10,11,12,13 -c -S 4



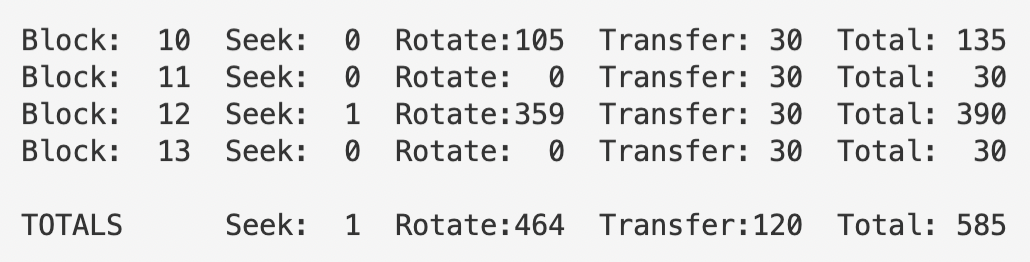
python3 disk.py -a 10,11,12,13 -c -S 8



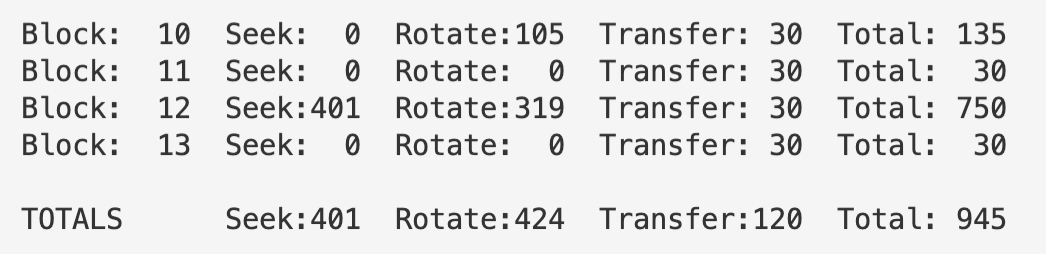
python3 disk.py -a 10,11,12,13 -c -S 10



python3 disk.py -a 10,11,12,13 -c -S 40



python3 disk.py -a 10,11,12,13 -c -S 0.1

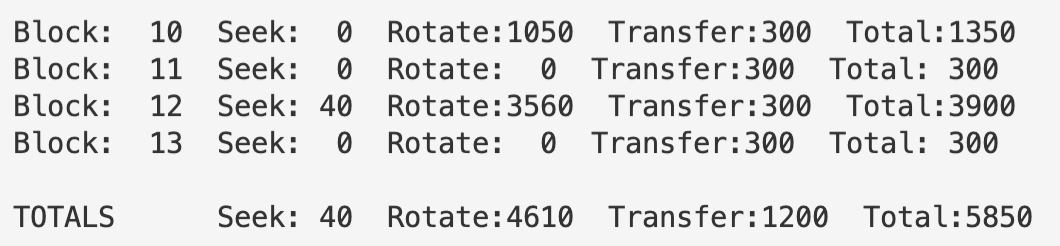


From the screenshots above, we can see that total time spent on IO remains the same when -S flag alters from 1 to 2, and then all the way up to 40, which is because when the disk head is reaching out for block 12, due to a shortage of seeking speed, it’ll take two complete cycles of disk spin for the disk head to reach the targeted track, which significantly aggravates the total time cost by 360.

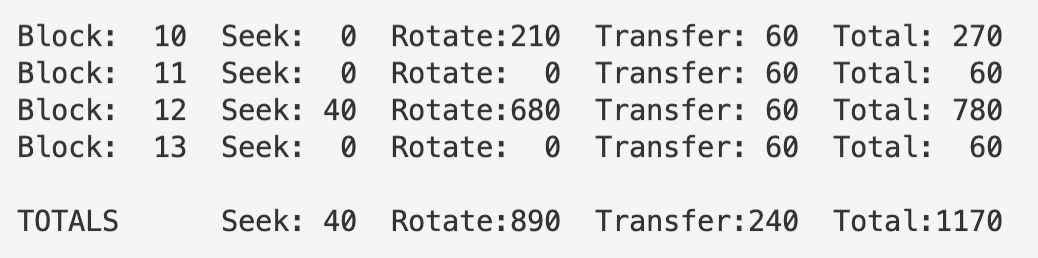
1. **Do the same requests above, but change the rotation rate: -R 0.1, -R 0.5, -R 0.01. How do the times change?**

Also take python3 disk.py -a 10,11,12,13 -c as an example:

python3 disk.py -a 10,11,12,13 -c -R 0.1



python3 disk.py -a 10,11,12,13 -c -R 0.5

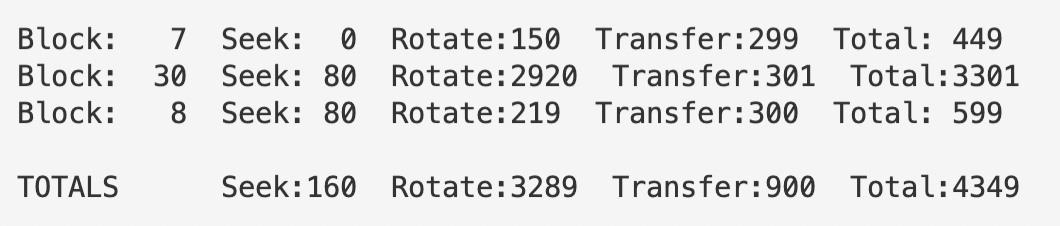


python3 disk.py -a 10,11,12,13 -c -R 0.01

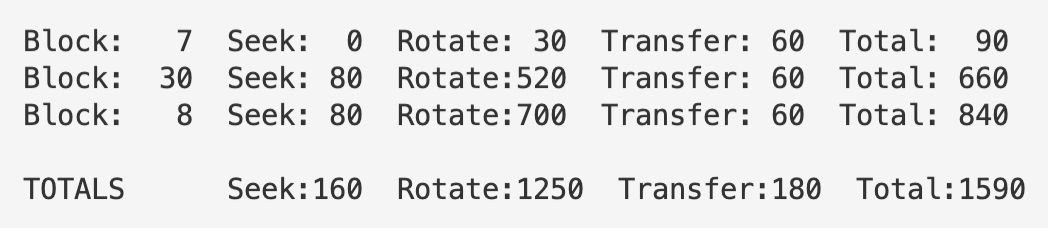


Clearly the total amount of time spent is in inverse proportion with the rotation rate in the case above. However, if we change the request set to -a 7, 30, 8, the situation changes.

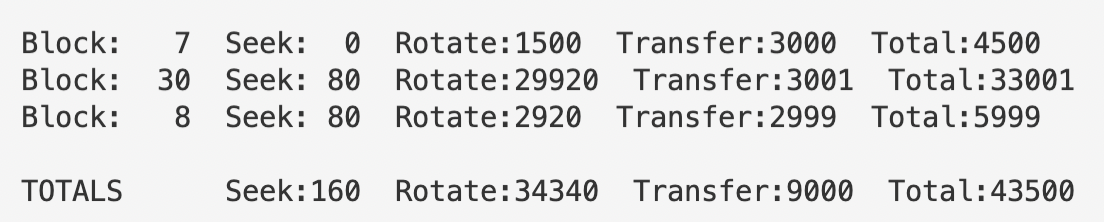
python3 disk.py -a 7,30,8 -c -R 0.1



python3 disk.py -a 7,30,8 -c -R 0.5



python3 disk.py -a 7,30,8 -c -R 0.01

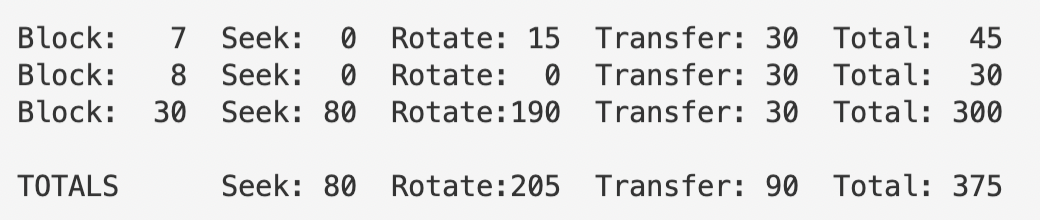


This is because as the rotation speed drops, the disk head would be able to reach the track outside as it’s trying to get block 8 after getting block 30 within a complete cycle of disk spin, which contributes to the reduction in rotating time.

1. **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?**

It would be reasonable to process the requests in the order of 7, 8 and finally 30.

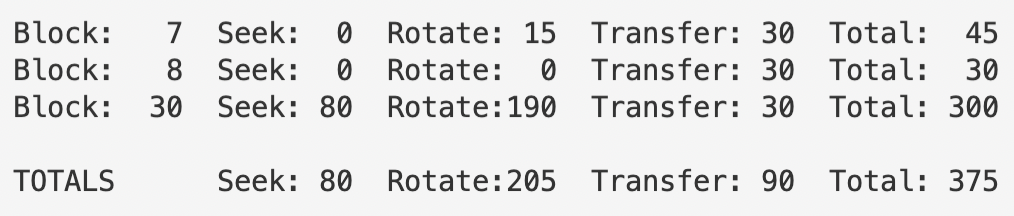
python3 disk.py -a 7,30,8 -c -p SSTF



This takes less than half of the time spent using policy FIFO.

1. **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?**

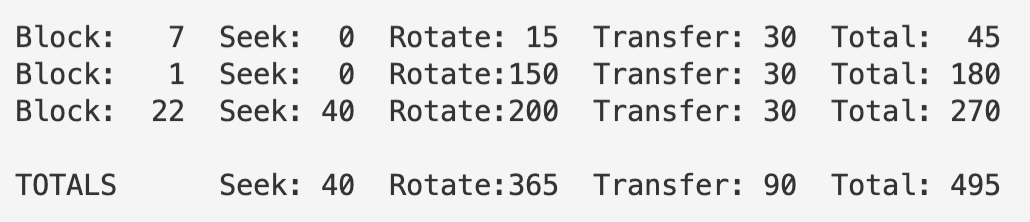
python3 disk.py -a 7,30,8 -c -p SATF



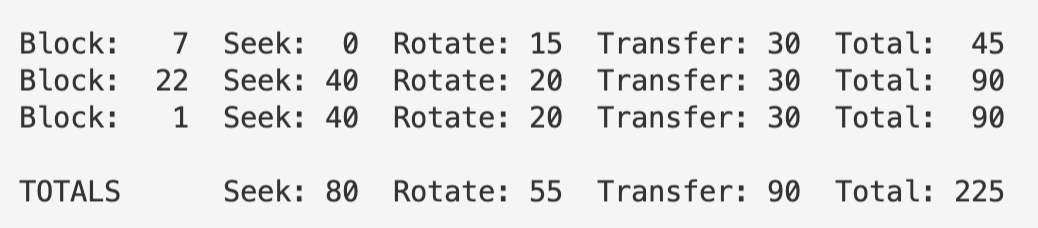
Total time cost is the same for SSTF and SATF in this case.

Let the request set be -a 7, 22, 1. The outcome for SSTF and SATF are shown in the screenshots below:

python3 disk.py -a 7,22,1 -p SSTF -c



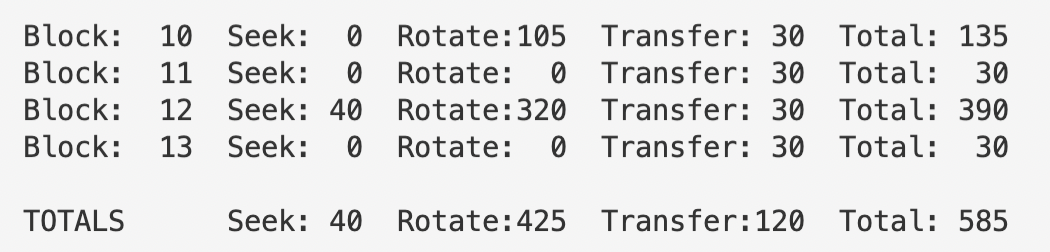
python3 disk.py -a 7,22,1 -p SATF -c



The time cost for SATF is less than a half of that of SSTF, which is because our disk reached for block 22 before 1 in the case of SATF and this makes it possible for the disk head to find all the desired blocks within a single cycle of disk spin, thus significantly reducing the time spent on rotation, which makes up for the extra time cost for two seeking actions.

1. **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?**

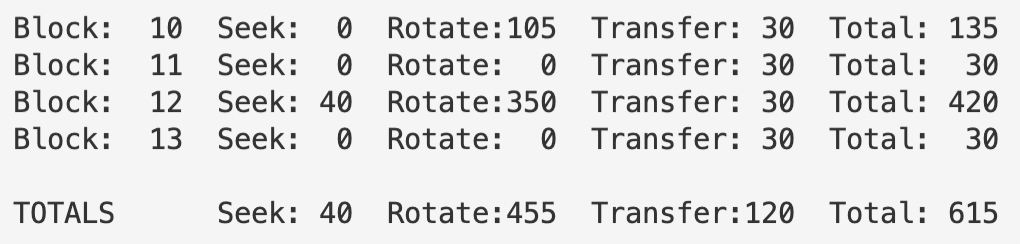
python3 disk.py -a 10,11,12,13 -c



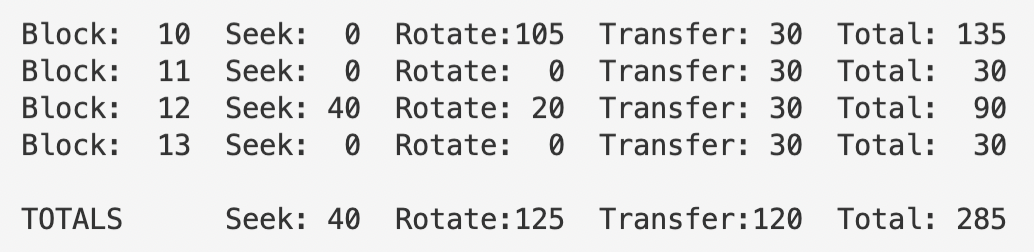
From the screenshot above we observe that rotation time makes up a large proportion of the total amount of time cost.

Now gradually increase the value of -o flag:

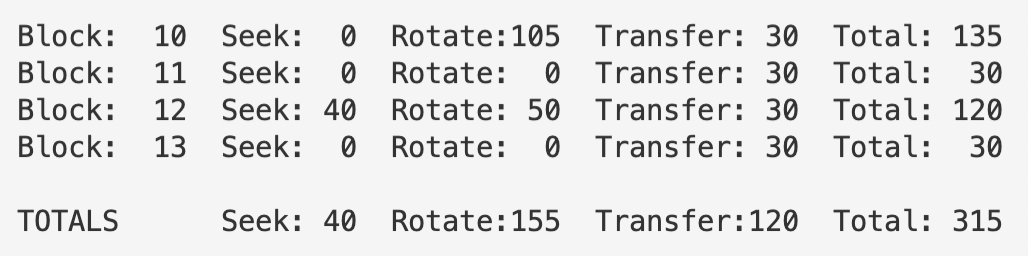
python3 disk.py -a 10,11,12,13 -c -o 1



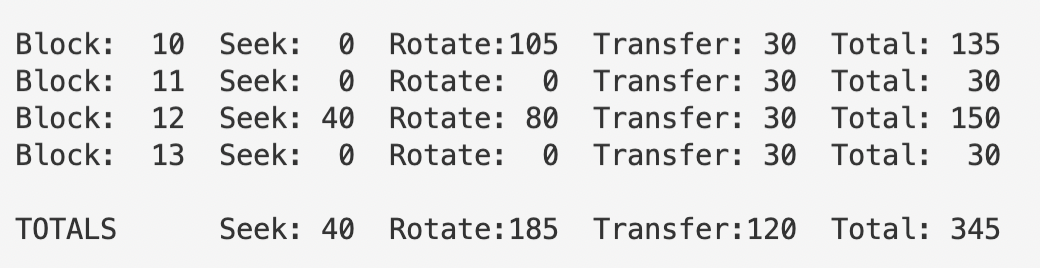
python3 disk.py -a 10,11,12,13 -c -o 2



python3 disk.py -a 10,11,12,13 -c -o 3



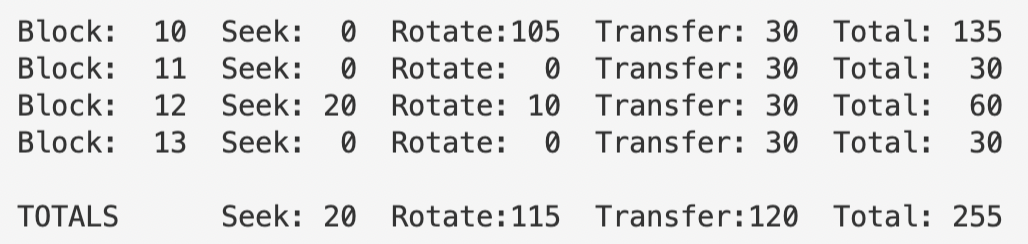
python3 disk.py -a 10,11,12,13 -c -o 4



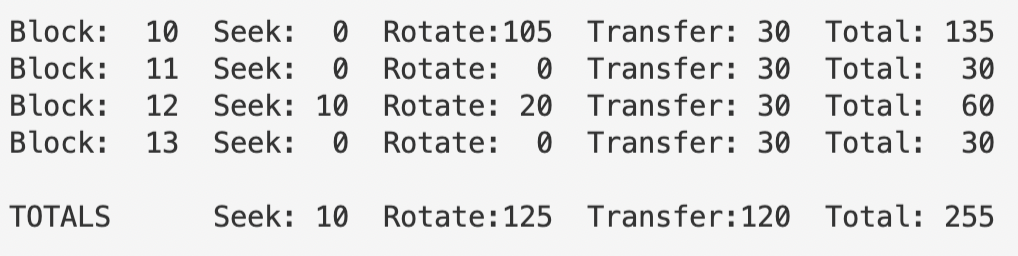
We can see that disk performance is best when -o has a value of 2. In fact, the total time cost plunges suddenly when -o increases from 1 to 2, and then rises gradually again as -o increases. This is because when -o equals 2, the disk head has just the right amount of time to enter the track inside before it misses block 12. When -o increases again, however, the disk performance will have to suffer from extra time cost brought by skew offset.

-S 2 and -S 4 perform the best in the following cases:

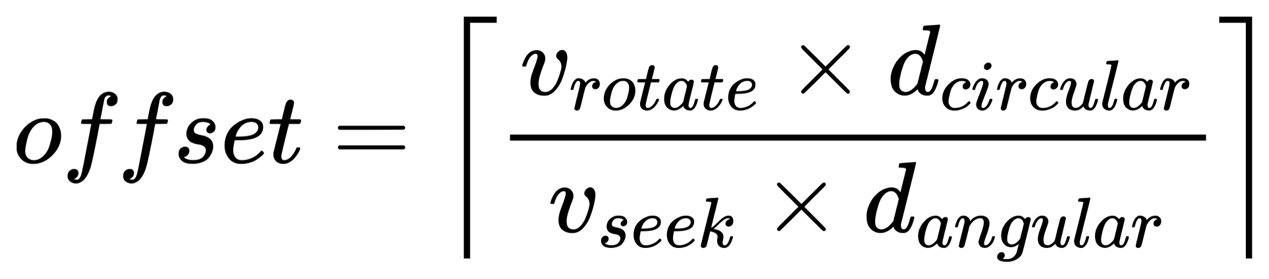
python3 disk.py -a 10,11,12,13 -c -o 1 -S 2



python3 disk.py -a 10,11,12,13 -c -o 1 -S 4

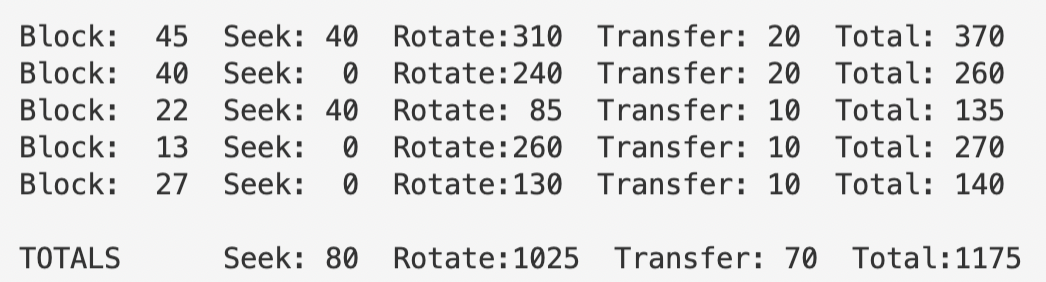


The formula is shown below:



1. **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?**

python3 disk.py -a -1 -A 5,-1,0 -z 10,20,30 -c -s 0



Outer\_bandwidth = 3 / (135 + 270 + 140)

Middle\_bandwidth = 2 / (370 + 260)

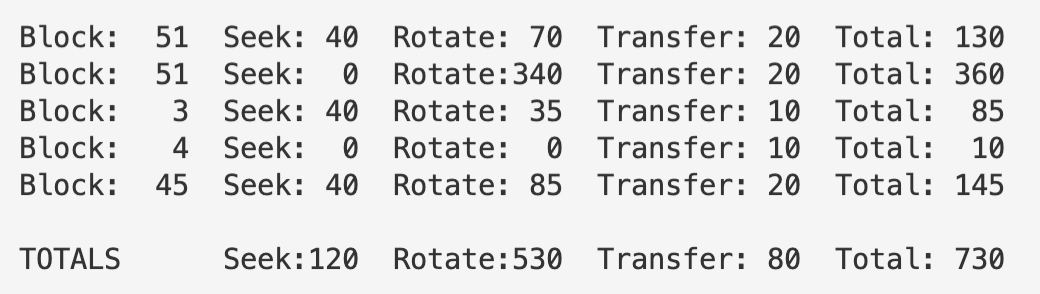
python3 disk.py -a -1 -A 5,-1,0 -z 10,20,30 -c -s 1



Outer\_bandwidth = 3 / (255 + 385 + 130)

Middle\_bandwidth = 2 / (115 + 280)

python3 disk.py -a -1 -A 5,-1,0 -z 10,20,30 -c -s 2

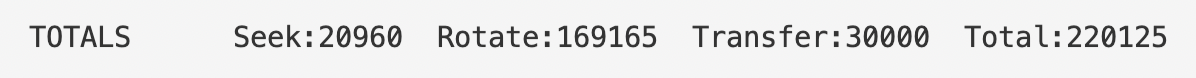


Outer\_bandwidth = 2 / (85 + 10)

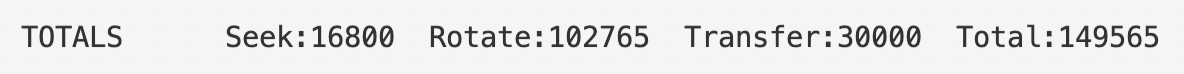
Middle\_bandwidth = 3 / (130 + 360 + 145)

1. **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?**

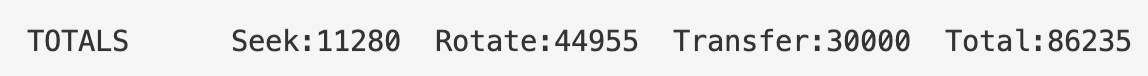
python3 disk.py -A 1000,-1,0 -s 0 -w 1 -c -p SATF



python3 disk.py -A 1000,-1,0 -s 0 -w 2 -c -p SATF



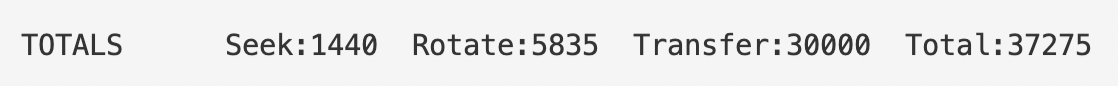
python3 disk.py -A 1000,-1,0 -s 0 -w 5 -c -p SATF



python3 disk.py -A 1000,-1,0 -s 0 -w 10 -c -p SATF



python3 disk.py -A 1000,-1,0 -s 0 -w 100 -c -p SATF

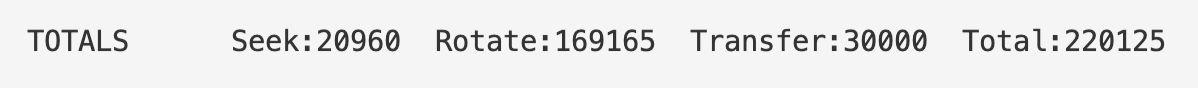


If we set -w 1, policy won’t matter since the disk can only examine next one to come. This could be backed up by the following trials.

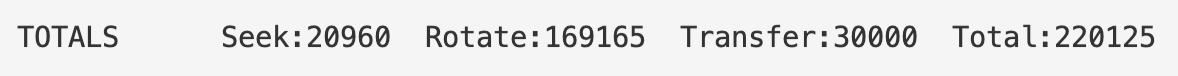
python3 disk.py -A 1000,-1,0 -s 0 -w 1 -c -p FIFO



python3 disk.py -A 1000,-1,0 -s 0 -w 1 -c -p SSTF

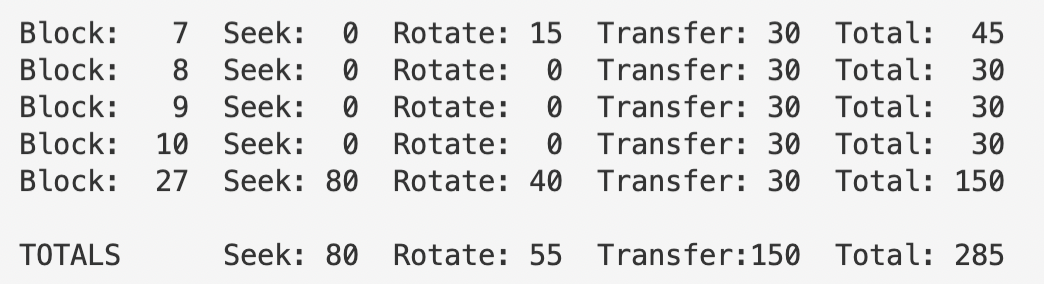


python3 disk.py -A 1000,-1,0 -s 0 -w 1 -c -p BSATF

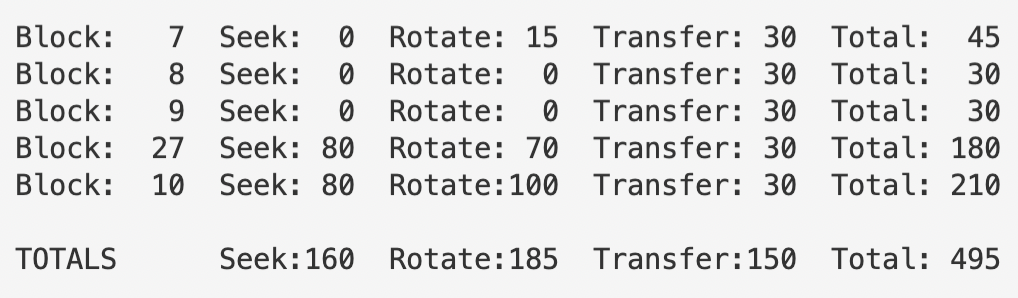


1. **Create a series of requests to starve a particular request, assuming an SATF policy. Given that sequence, how does it perform if you use a bounded SATF (BSATF) scheduling approach? In this approach, you specify the scheduling window (e.g., -w 4); the scheduler only moves onto the next window of 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?**

python3 disk.py -a 27,7,8,9,10 -p SATF -c

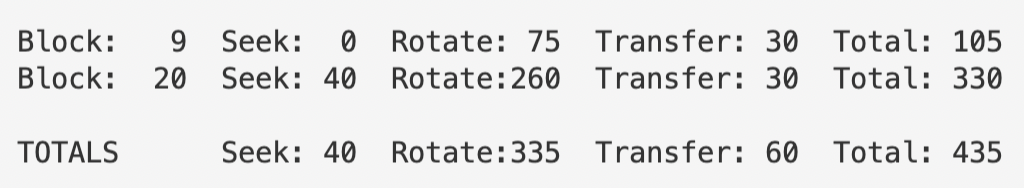


python3 disk.py -a 27,7,8,9,10 -p BSATF -c -w 4



1. **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?**

python3 disk.py -a 9,20 -c



python3 disk.py -a 9,20 -c -p SATF

