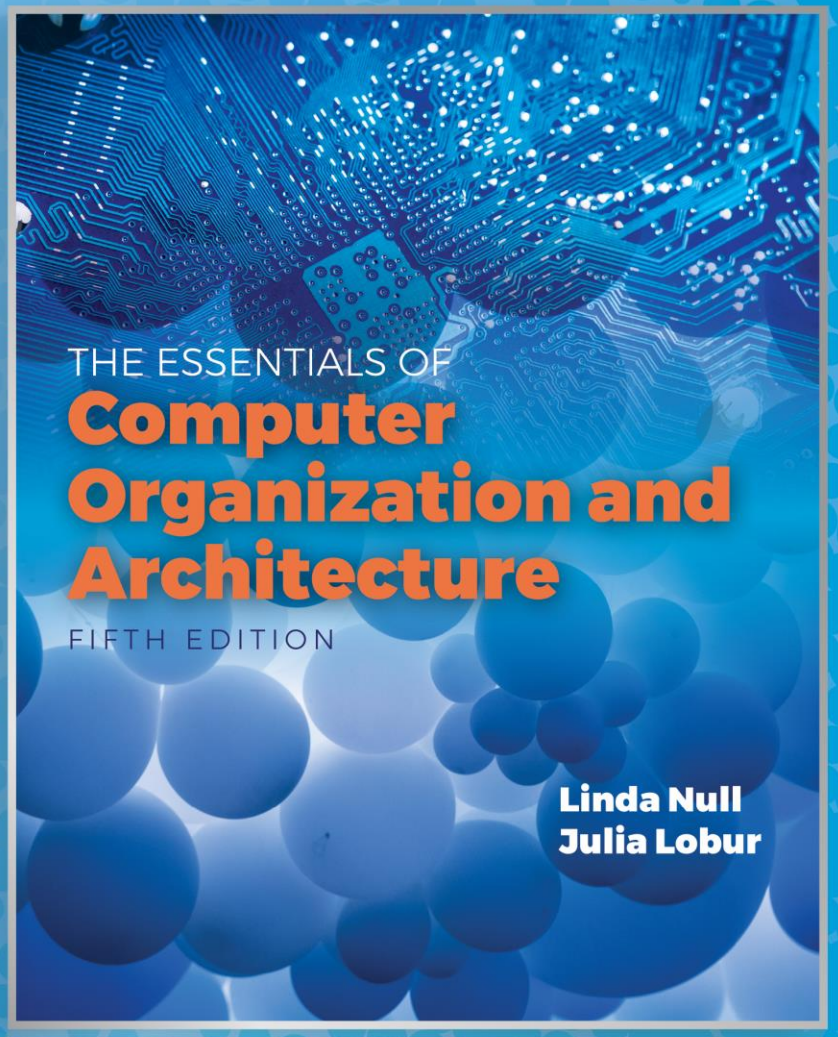
This is the fifth lecture of Chapter 11

Chapter 11

Performance Measurement and Analysis (E)



Quick review of last lecture

- CPU Performance Optimization
  - Branch optimization
    - Delayed branching
    - Branch prediction
      - Fixed predictions
      - Dynamic prediction
      - Static prediction
  - User code optimization
    - Operation counting
    - Loop optimization
      - Loop unrolling
      - Loop fusion
      - Loop fission

11.6 Disk Performance (1 of 23)
1. Understanding the problem

- Optimal disk performance is critical to system throughput.
- Disk drives are the slowest memory component, with the fastest access times one million times longer than main memory access times.
- A slow disk system can choke transaction processing and drag down the performance of all programs when virtual memory paging is involved.
- Low CPU utilization can actually indicate a problem in the I/O subsystem, because the CPU spends more time waiting than running.

11.6 Disk Performance (2 of 23)

- Disk utilization is the measure of the percentage of the time that the disk is busy servicing I/O requests.
- It gives the probability that the disk will be busy when another I/O request arrives in the disk service queue.
- Disk utilization is determined by the speed of the disk and the rate at which requests arrive in the service queue. Stated mathematically:
  - Utilization = Request Arrival Rate ÷ Disk Service Rate.
    - where the arrival rate is given in requests per second, and the disk service rate is given in I/O operations per second (IOPS)

11.6 Disk Performance (3 of 23)

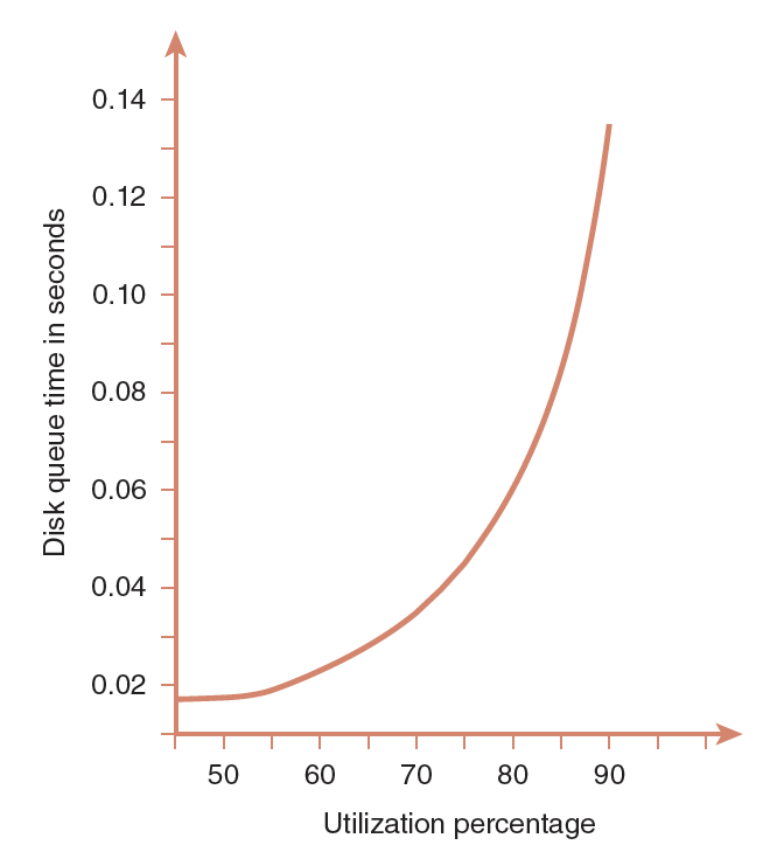
- The amount of time that a request spends in the queue is directly related to the service time and the probability that the disk is busy, and it is indirectly related to the probability that the disk is idle.
- In formula form, we have:

$$\text{Time in Queue} = (\text{Service time} \times \text{Utilization}) \div (1 - \text{Utilization})$$

- The important relationship between queue time and utilization (from the formula above) is shown graphically on the next slide.

11.6 Disk Performance (4 of 23)

- The “knee” of the curve is around 78%. This is why 80% is the rule-of-thumb upper limit for utilization for most disk drives.
- Beyond that, queue time quickly becomes excessive.



11.6 Disk Performance (5 of 23)
2. Physical Considerations

- The manner in which files are organized on a disk greatly affects throughput.
- Disk arm motion is the greatest consumer of service time.
- Disk specifications cite average seek time, which is usually in the range of 5 to 10ms.
- However, a full-stroke seek can take as long as 15 to 20ms.
- Clever disk scheduling algorithms endeavor to minimize seek time.

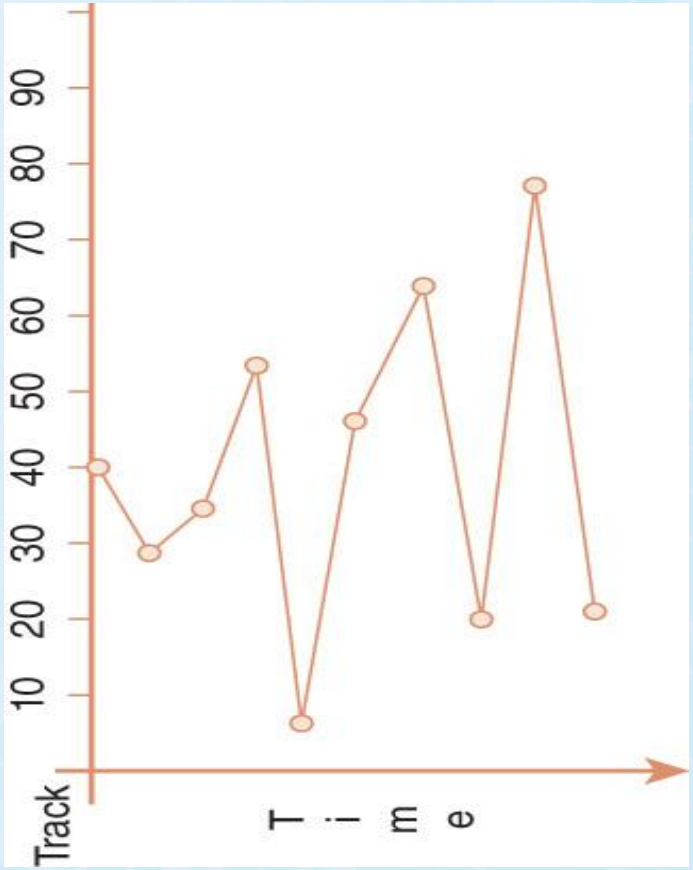
11.6 Disk Performance (6 of 23)
3. Logical Consideration

- The most naïve disk scheduling policy is *first-come, first-served* (*FCFS*).
- As its name implies, FCFS services all I/O requests in the order in which they arrive in the queue.
- With this approach, there is no real control over arm motion, so random, wide sweeps across the disk are possible.
- The next slide illustrates the arm motion of FCFS.

11.6 Disk Performance (7 of 23)

- Using FCFS, performance is unpredictable and widely variable.

- Initially, at track 40
- Requests to read tracks
  - 28, 35, 52, 6, 46, 62, 19, 75, 21

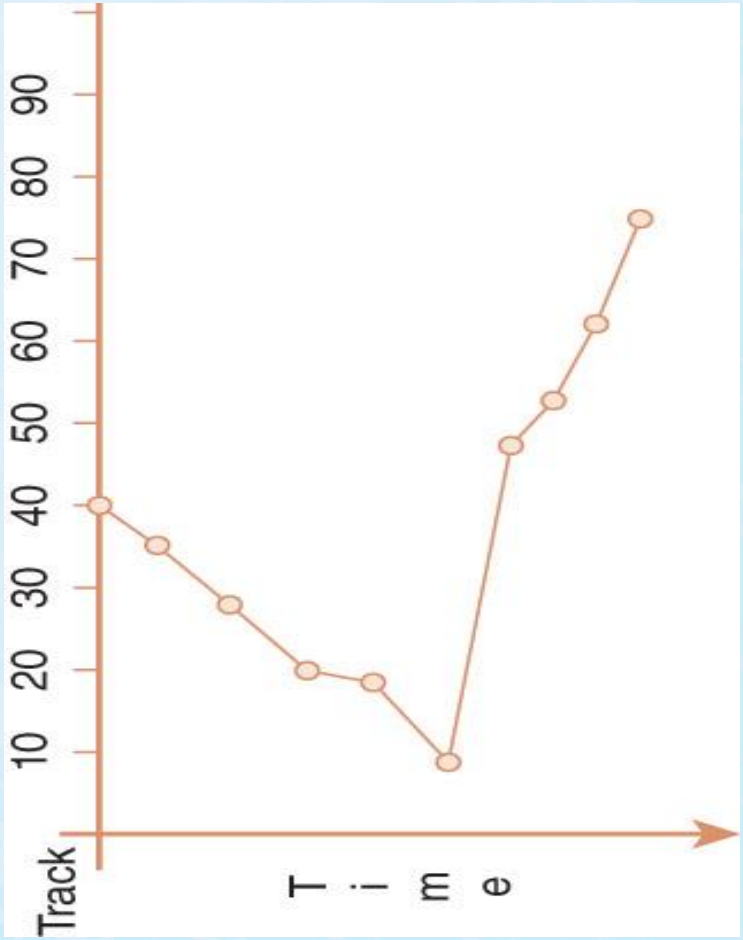


11.6 Disk Performance (8 of 23)

- Arm motion is reduced when requests are ordered so that the disk arm moves only to the track nearest its current location.
- This is the idea employed by the *shortest seek time first* (*SSTF*) scheduling algorithm.
- Disk track requests are queued and selected so that the minimum arm motion is involved in servicing the request.
- The next slide illustrates the arm motion of SSTF.

11.6 Disk Performance (9 of 23)

- Shortest Seek Time First (SSTF)
- Disk schedule using SSTF
  - Initially, at track 40
  - 35, 28, 21, 19, 6, 46, 52, 62, 75



11.6 Disk Performance (10 of 23)

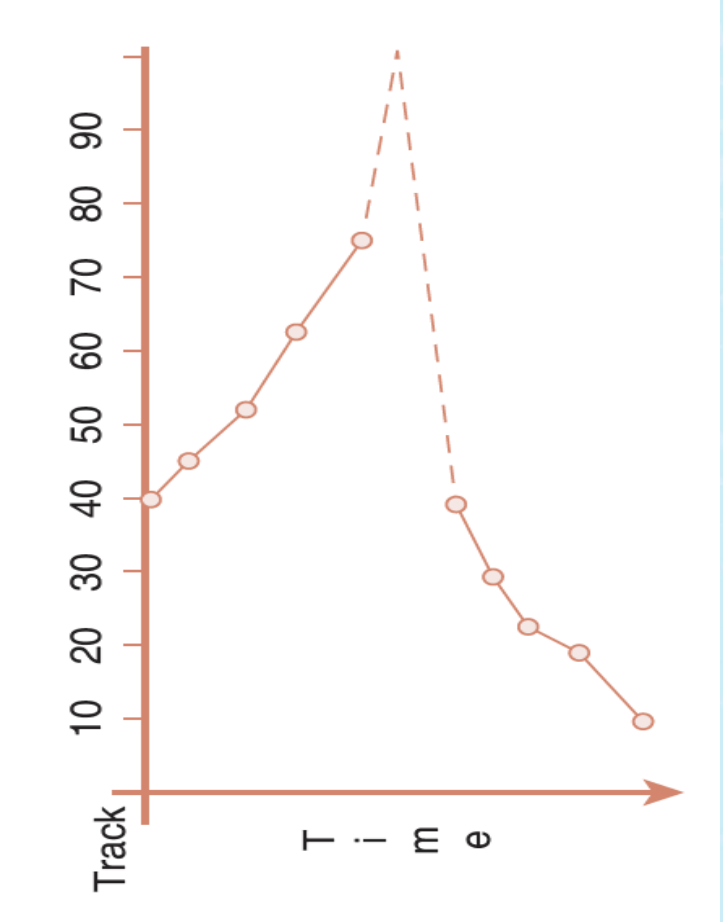
- With SSTF, starvation is possible: A track request for a "remote" track could keep getting shoved to the back of the queue, and nearer requests are serviced.
  - Interestingly, this problem is at its worst with low disk utilization rates.
- To avoid starvation, fairness can be enforced by having the disk arm continually sweep over the surface of the disk, stopping when it reaches a track for which it has a request.
  - This approach is called an *elevator algorithm*.

11.6 Disk Performance (11 of 23)

- In the context of disk scheduling, the elevator algorithm is known as *SCAN* (which is not an acronym).
- While SCAN entails a lot of arm motion, the motion is constant and predictable.
- Moreover, the arm changes direction only twice: At the center and at the outermost edges of the disk.
- The next slide illustrates the arm motion of SCAN.

11.6 Disk Performance (12 of 23)

- SCAN Disk Scheduling
- Disk schedule using SCAN
  - Initially, at track 40
  - 46, 52, 62, 75, 35, 28, 21, 19, 6



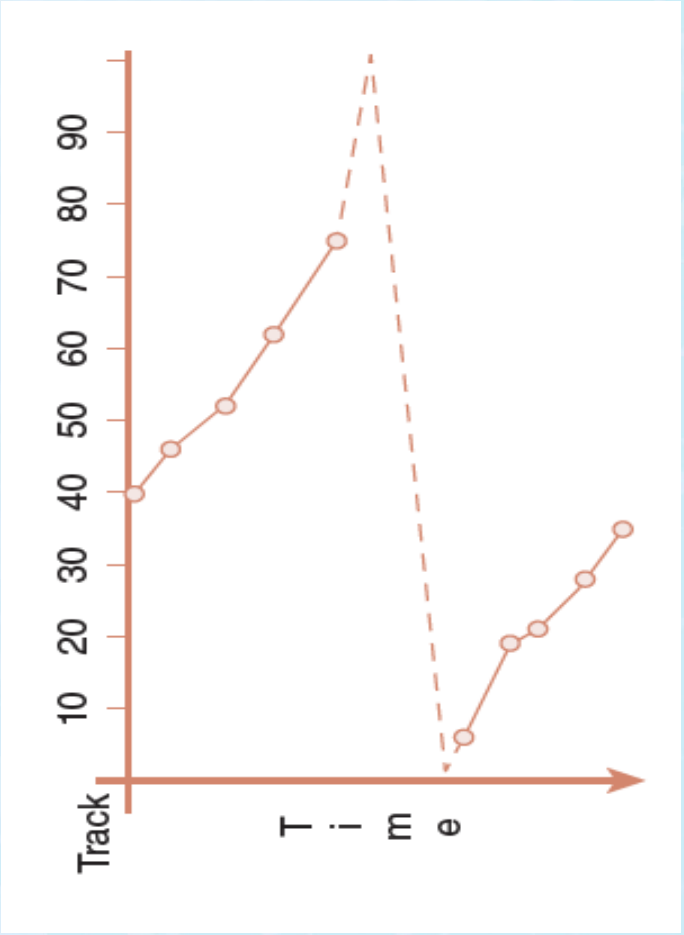
11.6 Disk Performance (13 of 23)

- A SCAN variant, called *C-SCAN* for circular SCAN, treats track zero as if it is adjacent to the highest-numbered track on the disk.
- The arm moves in one direction only, providing a simpler SCAN implementation.
- The following slide illustrates a series of read requests where after track 75 is read, the arm passes to track 99, and then to track 0 from which it starts reading the lowest numbered tracks starting with track 6.

11.6 Disk Performance (14 of 23)

- C-SCAN Disk Scheduling

- Disk schedule using C-SCAN
  - Initially, at track 40
  - 46, 52, 62, 75, 6, 19, 21, 28, 35



11.6 Disk Performance (15 of 23)

- The disk arm motion of SCAN and C-SCAN can be reduced through the use of the *LOOK* and *C-LOOK* algorithms.
- Instead of sweeping the entire disk, the disk arm travels only to the highest- and lowest-numbered tracks for which access requests are pending.
- Although the circuitry is more complex, LOOK and C-LOOK provide the best theoretical throughput, although the circuitry is the most complex.

11.6 Disk Performance (16 of 23)

- At high utilization rates, SSTF performs slightly better than SCAN or LOOK. But the risk of starvation persists.
- Under very low utilization (under 20%), the performance of any of these algorithms will be acceptable.
- No matter which scheduling algorithm is used, file placement greatly influences performance.
- When possible, the most frequently-used files should reside in the center tracks of the disk, and the disk should be periodically defragmented.

11.6 Disk Performance (17 of 23)

- The best way to reduce disk arm motion is to avoid using the disk as much as possible.
- To this end, many disk drives, or disk drive controllers, are provided with cache memory or a number of main memory pages set aside for the exclusive use of the I/O subsystem.
- Disk cache memory is usually associative.
  - Because associative cache searches are time-consuming, performance can actually be better with smaller disk caches because hit rates are usually low.

11.6 Disk Performance (18 of 23)

- Many disk drive-based caches use prefetching techniques to reduce disk accesses.
- When using prefetching, a disk will read a number of sectors subsequent to the one requested with the expectation that one or more of the subsequent sectors will be needed “soon.”
- Empirical studies have shown that over 50% of disk accesses are sequential in nature, and that prefetching increases performance by 40%, on average.

11.6 Disk Performance (19 of 23)

- Prefetching is subject to cache pollution, which occurs when the cache is filled with data that no process needs, leaving less room for useful data.
- Various replacement algorithms, LRU, FIFO and random, are employed to help keep the cache clean.
- Additionally, because disk caches serve as a staging area for data to be written to the disk, some disk cache management schemes evict all bytes after they have been written to the disk.

11.6 Disk Performance (20 of 23)

- With cached disk writes, we are faced with the problem that cache is volatile memory.
- In the event of a massive system failure, data in the cache will be lost.
- An application believes that the data has been committed to the disk, when it really is in the cache. If the cache fails, the data just disappears.
- To defend against power loss to the cache, some disk controller-based caches are mirrored and supplied with a battery backup.

11.6 Disk Performance (21 of 23)

- Another approach to combating cache failure is to employ a write-through cache where a copy of the data is retained in the cache in case it is needed again “soon,” but it is simultaneously written to the disk.
- The operating system is signaled that the I/O is complete only after the data has actually been placed on the disk.
- With a write-through cache, performance is somewhat compromised to provide reliability.

11.6 Disk Performance (22 of 23)

- When throughput is more important than reliability, a system may employ the write back cache policy.
- Some disk drives employ opportunistic writes.
- With this approach, dirty blocks wait in the cache until the arrival of a read request for the same cylinder.
- The write operation is then “piggybacked” onto the read operation.

11.6 Disk Performance (23 of 23)

- Opportunistic writes have the effect of reducing performance on reads, but of improving it for writes.
- The tradeoffs involved in optimizing disk performance can present difficult choices.
- Our first responsibility is to assure data reliability and consistency.
- No matter what its price, upgrading a disk subsystem is always cheaper than replacing lost data.

Conclusion (1 of 3)

- Computer performance assessment relies upon measures of central tendency that include the arithmetic mean, weighted arithmetic mean, the geometric mean, and the harmonic mean.
- Each of these is applicable under different circumstances.
- Benchmark suites have been designed to provide objective performance assessment. The most well respected of these are the SPEC and TPC benchmarks.

Conclusion (2 of 3)

- CPU performance depends upon many factors.
- These include pipelining, parallel execution units, integrated floating-point units, and effective branch prediction.
- User code optimization affords the greatest opportunity for performance improvement.
- Code optimization methods include loop manipulation and good algorithm design.

Conclusion (3 of 3)

- Most systems are heavily dependent upon I/O subsystems.
- Disk performance can be improved through good scheduling algorithms, appropriate file placement, and caching.
- Caching provides speed, but involves some risk.
- Keeping disks defragmented reduces arm motion and results in faster service time.