Lecture #23

*********************************
Review -- 1 min
*********************************
I/O Motivation
    limits CPU performance
    perf improving slowly
    industry driven by application domains not perf.

Challenge – improve architecture to overcome limitations of underlying technology.

*********************************
Outline - 1 min
*********************************
Disk Performance
Performance Measures
A Little Queuing Theory

*********************************
Preview - 1 min
*********************************
Interface to CPU
Busses

*********************************
Lecture - 20 min
*********************************
disk access time
    response time = seek + rot + xfer
        + controller time
        + queue time
        +/- cache effects

seek time
-----------
    for back of the envelope:
    min seek – 1 sector

max seek – inside to outside
average seek – 1/3 of way across disk (somewhat meaningless)

back of the envelope – real seeks tend to be between min and avg (due to locality)

More details
-------------

- better way to look at it - seek profile

short seeks – dominated by settling time (1-3ms)
long seeks – overhead + proportianal to distance

FIGURE 5 – IEEE computer March 1994
**<get graph, min, max, avg from IEEE computer>**

- Track following

narrower tracks → increased settling time
→ read seek < write seek (speculative reads)
**head switch time** nontrivial
(due to electronics, *head settling time (tracks no longer precisely aligned across different platters*)
→ "cylinders" less important than they once were

- recalibration

use internal table to decide how far to seek
(store distance to subset of track→ sawtooth pattern)
figure: H&P fig 6.42

thermal expansion, arm pivot-bearing stickiness, etc
→ recalibrate drives internal seek tables
triggered by 1) timer
2) temperature changes
→ happen often during start-up, every 15-30 minutes during steady state

recalibration takes 500+ milliseconds → kills multimeda

(solution – firmware to avoid recalibration or to allow host to schedule recalibration “multimedia disk drives”)

Data layout issues
--------------------
zoning – more bytes on outer tracks than inner tracks
(improves capacity, bandwidth)
→ can’t just divide address by bytes per track to get track #

sparing – too expensive to manufacture “perfect” surface
→ remap bad sectors
slip sparing – shift numbering by 1
remapping – map bad sector somewhere else on disk

→ makes seek, rotation models complex

Rot time
today 7200RPM is common
→ 8ms per rotation
→ avg wait (after seek): 4ms

subtleties
track skewing
stagger block zero on different tracks to account for seek, head switch

drift in rot. rate

0.5%-1%
(→ hard to model disk rotational position
100-200 revolutions after last known operation)

Transfer time

time for sector to fly under head

e.g. 32 sectors per track
    4ms per rotation
    → 1/8 ms to read a track

bandwidth increases more quickly than other factors
    **(<graph> data from thesis)**
    reason: capacity drive increases bit per inch
    (so not just relying on mechanical)

→ want to transfer large, contiguous units from
e.g. current disk 10MB/s transfer, 10ms seek + rot
→ need to transfer 10ms * 10MB/s = 100 MB to get 50% efficiency
    (cylinder probably more like 1MB = 32 sectors * 1KB/sector * 20 tracks/cylinder → 640KB/cyl)

Controller time
    **nontrivial (see IEEE computer article)**

    0.3-1ms
        (processors get faster at about the same rate that controller programs get more complicated)

Bus transfer time
    scsi 5MB/s - 40 MB/s
    fibre channel – 100MB/s
    FIGURE 2: IEEE Computer march 1994

Queue time
    later

Caching
    read ahead/read buffer

hard to model (large transfers may bypass cache, replacement policy, etc.)

read ahead really important for sequential reads (otherwise back-to-back sector reads take 1 full revolution)

write buffer

"immediate reporting" on data writes

(optimize for sequential transfer broken up by FS)

OS can disable for metadata writes.

Major factor: FIGURE 6 IEEE Computer march 1994

Bottom line

1. tech trends

avg seek time 1987: 16.7ms

1994: 8.0/9.0 ms (read, write)

1999: 5.7/6.5 ms

9%/year

rotational speed/avg latency 1987: 3600rpm/8.3 ms

1994: 7200rpm/4.2 ms

1999: 10000rpm/2.99ms

9%/year

bandwdith: 1987 2.5 MB/s

1994: 6-9 MB/s (inner-outer tracks)

1999: 18-28MB/s

20%/yr

8KB transfer: 1987: 28ms

1994: 13.1 ms

1999: 8.7ms

10%/year

1MB transfer: 1987: 425 ms

1994: 123 ms

1999: 55 ms
(19%/year)

A modern high-performance disk: (March 1999)
Seagate Cheetah:

| | |
|---|---|
| Capacity | 36GB |
| Internal Transfer rate | 18.3-28.0MB/s |
| External transfer rate | 80-100MB/s (Fibre Channel) |
| Multisegmented cache | 1MB |
| Optional cache | 4MB |
| Track-to-track seek | .6ms(read)/.9ms(write) |
| Average seek | 5.7ms(read)/6.5ms(write) |
| Maximum seek | 12ms(read)/13ms(write) |
| Average rot latency | 2.99ms |
| Spindle speed | 10,000RPM |
| Disks/heads | 12/24 |
| Sectors per drive | 71M |
| Cylinders | 9,772 |

Recoverable read errors per bits read 10 per 10^12
Nonrecoverable read errors per bits read 1 per 10^15

Bottom line 2: disk access timeline (fig 2 IEEE Computer March 1994)

*********************************
Admin - 3 min
*********************************
HW4 available
Projects

*********************************
Lecture - 24 min
*********************************

## Measures of Performance

------------------------------

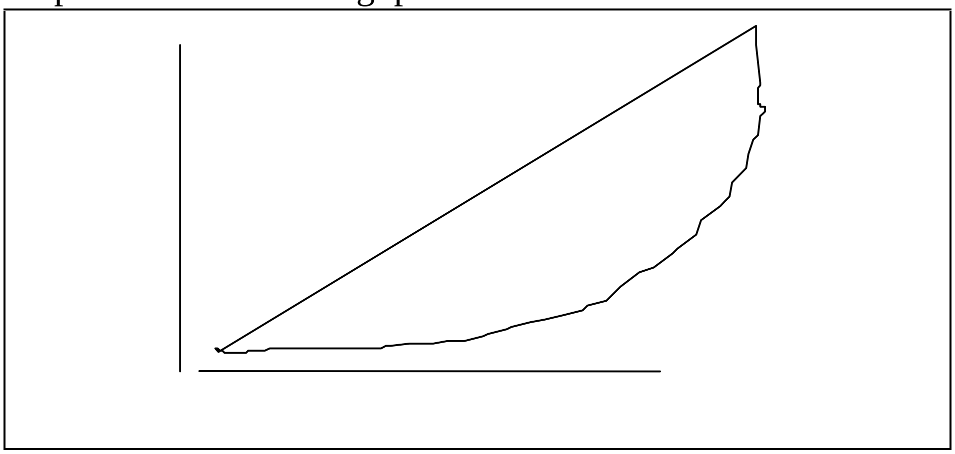
Everyone talks about throughput

(it is easy to measure)

response time is what matters

(for productivity)

Response time v. Throughput



Fundamental trade-off

(picture: queue in front of disk

for best response time: empty queue

for best throughput: full queue)

## Response Time v. Productivity

-------------------------------------

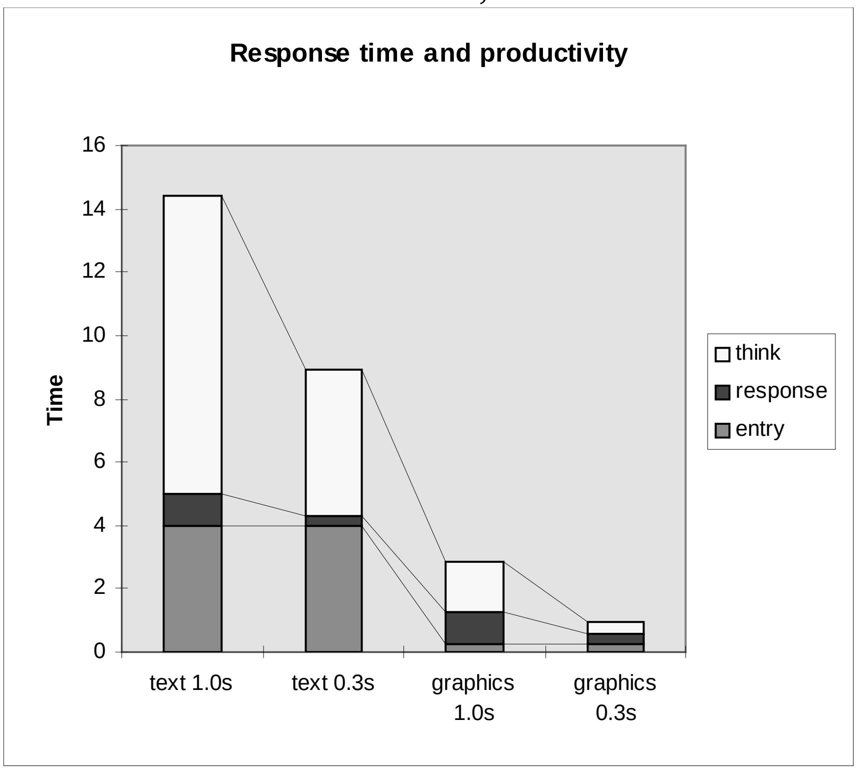
- Interactive environments

Each action has 3 parts

1. entry time
2. system response time
3. think time

QUESTION: What happens as computer gets faster?
A: system time falls?

Actual answer: Think time falls, too!




Morals

1. faster computers → more productive people
   "get on a roll" – better concentration

   another study – found a novice with a fast machine could be as productive as an expert with a slow machine (for some graphical tasks)

2. < 1sec is a threshold – big wins when response time < 1 sec

3. variability – predictable performance helps productivity as much/more fast, but unpredictable perf

This is all an argument for workstations
- not efficient wrt throughput (idle most of the time)
- great for response time

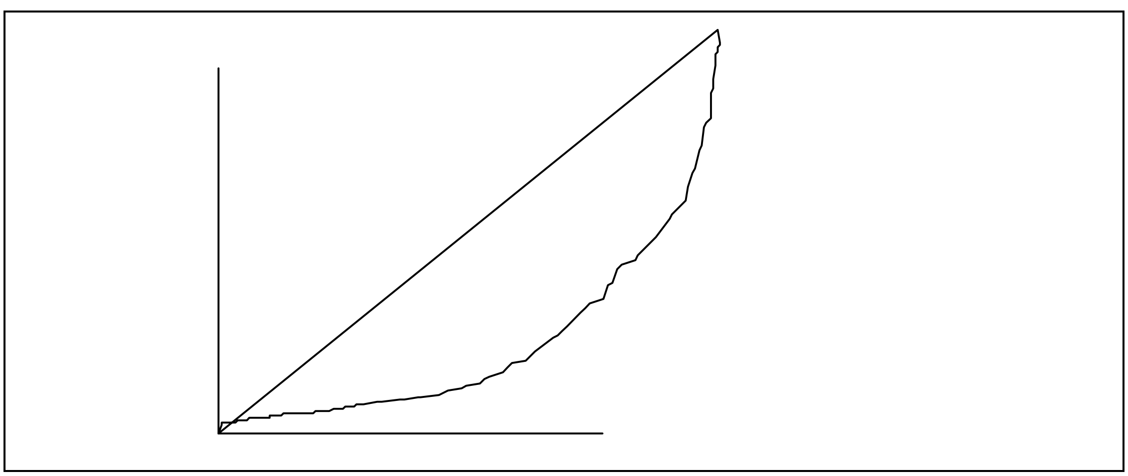
Introduction to Queuing Theory
------------------------------------

Disclaimer – just a few very simple ideas
- these simple ideas have a lot of power,
- but you could take a whole class on queuing theory

2 results
1. Little's law
   Lsys = rT
   
   #jobs in a system = arrival rate * residence time

   what's powerful here: applies to any closed system in equilibrium
   - closed == black box does not create/destroy jobs
   - equilibrium == arrival rate = departure rate

2. Response time v. throughput graph



Tqueue = Tservice * u/(1-u)   /* special case */

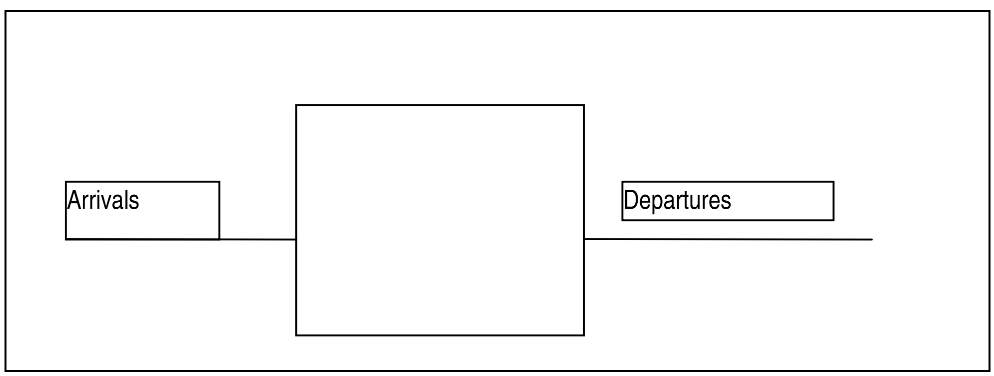
What's powerful here: trade-off between throughput and response time is fundamental. If I drive throughput to 100%, response time goes to infinity.

Also note: variability assumptions are crucial. If arrivals were evenly spaced, could run at 100%.

## Little's Law

Well known for a long time
Little proved it, so it gets his name

$$L_{sys} = r\,T$$




What is interesing – very weak assumptions for this to hold

- system in equilibrium – arrival rate = departure rate
- nothing in black box creates/destroys jobs

e.g.

we have 150 grad students in program
grad student stays in program for 3 years on average
how many students must we accept per year?

The colorado river brings 8M gallons of water into lake travis per hr.
The lake contains 8000M gallons
how long does the average molecule stay in the lake?

I send 50 requests per second to a disk

each request takes 0.1 second to come back
how many requests queued at disk?

L = 50 req/sec * 0.1 sec = 5 req (4 in queue, 1 in disk)

Latency/Throughput Tradeoff
------------------------------------

The above disk calculation took response time as a given
How do you predict response time for a system you want to build?
(little's law requires you to know either response time OR how many items in queue

- these are the things you want to find out

Note that disk access time alone doesn't help –
even if you know that avg disk access time is
1ms (controller) + 3 ms (typical seek) + 4 ms (rot) + 0.5 (xfer)
Tdisk= 8.5 ms

If I'm sending 50 requests/second on average then
Ldisk = .0085 * 50 = .425

QUESTION: How do you interpret that?
A: disk utilization
But that doesn't tell use response time to user.

Central question – burstinesss of requests
Consider two extremes

1. requests arrive every 20ms ( = 50 req/sec)
   → no queuing → Tsystem = 8.5 ms

2. 1000 requests arrive at once, then no requests arrive for several seconds
   → first requests takes 8.5 ms, second takes 17, third takes 25.5…
   Tsystem = ?? some big number??

The Queuing part of Queuing theory (finally)
-----------------------------------------------------
Assume FIFO queue

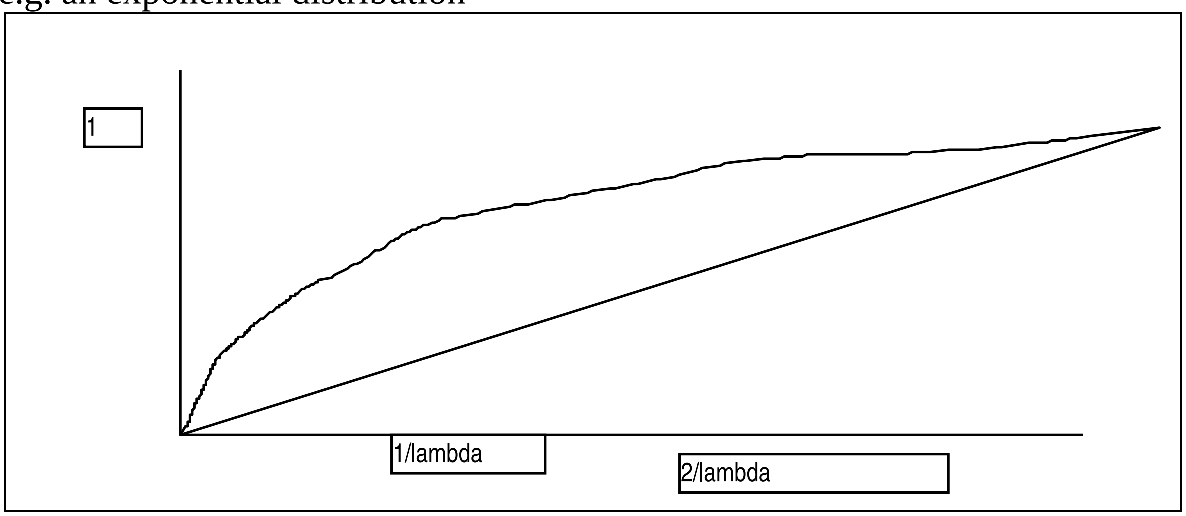
Tsys = Length_queue * Time_server + residual service time

Length_queue * Time_server → how long do I wait for those jobs

last term a bit tricky
RST = mean time to complete tasks being serviced when new task arrives

RST
-----

Assume Memoryless, random arrivals:
e.g. an exponential distribution




P(T<t) = 1 - e^(lambda*t)

What does this mean?
Exponential distribution says that past behavior does not effect future expectations
e.g. if each instruction in stream is equally likely to cause an I/O
e.g. a bunch of different processes are all independently causing I/O to occur

For these distributions squared Coefficient of variation

C = variance / weighted mean time ^2

For exponential distribution → C == 1.0

Hypoexponential → C < 1 – more clustered around average

Hyperexponential → C > 1 – less clustered around average

In such a system

RST = ½ Weighted mean time * (1+C)

Intuition: no variation → C = 0 → makes sense

huge variation → C = infinity → <the burst example above>

For exponential distribution:

Tqueue = Length_queue * Time_server + residual service time
= Length_queue * T_server
+ server utilization * (1/2 * Tserv * (1+C))

= (Arrival Rate * Tqueue) * T_serv
+ Util_serv * Tserv
= Tqueue * Util_serv + Tserv * Util_serv

Solving for Tqueue

→

Tqueue = Tserv * Util_serv/(1-Util_serv) (for C == 1)

Draw Util v. response time curve

Caveats

---------

Queuing system assumptions

1. System is in equilibrium
2. Exponential interrarrival times
3. infinite queue
4. FIFO queue
5. All tasks in line must be completed

These assumptions should make you uncomfortable!

In particular – exponential interrarival times
Used for 2 reasons

    1. some workloads kinda look like this (random arrivals)
    2. makes math easy

#2 is often the main reason, I fear
What's bad is many workloads don't look at all like this

    e.g. network traffic – self-similar
    e.g. disk traffic – 30 second write buffer
    …

*********************************
Summary - 1 min
*********************************

Disks – more complicated than normally taught
    and getting more complicated all the time
Importance of low, predictable response time
Little's law – L = rT
Response time v. throughput