Queueing Basics

P. J. Denning
For CS471/571

Performance Questions

- Performance questions always part of a systems discussion
  - throughput (jobs per second)
  - response time (seconds)
  - congestion and bottlenecks
  - capacity planning
- How to measure and forecast?

Airline Reservations Example

- 1000 reservation agents around the USA.
- “Disk farm” in New York City.
- Each agent issues new transactions against the database every 60 seconds.
- Every transaction accesses the directory disk an average of 10 times.
- The directory disk takes an average of 5 milliseconds to serve a request and is in use 80% of the time.

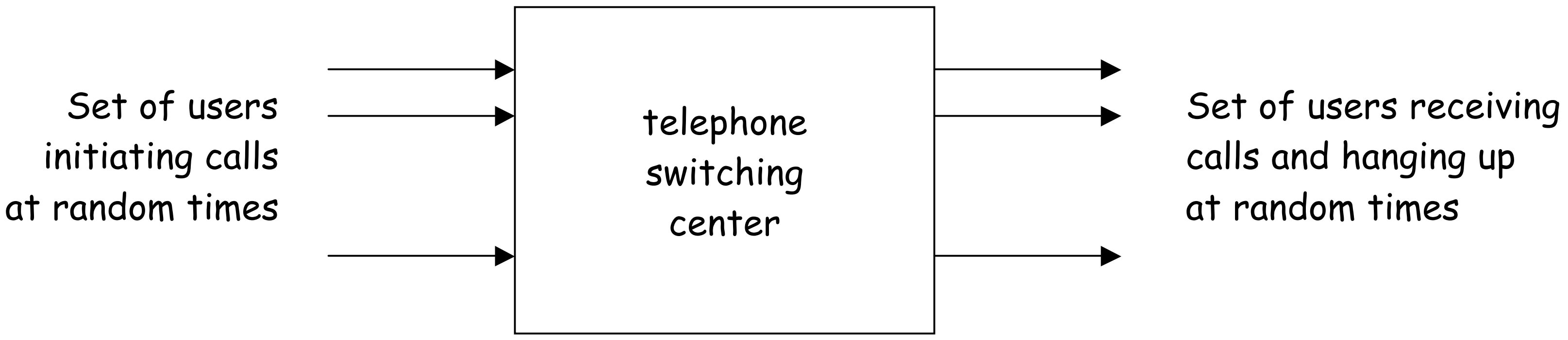
- What is the throughput (jobs per second) completed by the entire system?
- What is the response time experienced by an agent waiting for a transaction?
- Can these questions be answered precisely? Approximately? Not at all?

Tools

- Queueing theory gives the basic tools for answering such questions.
- The theory deals with randomness in physical processes such as
  - the arrival times of agent requests
  - the service times at the disks and CPUs
  - lengths of queues
  - variations in response times
- The theory allows us to characterize the performance measures statistically, in terms of averages, given the statistics of arrivals and services

Erlang's Model

- The first use of queueing theory in engineering occurred around 1909 when the Danish engineer A. K. Erlang modeled telephone systems, including interarrival times and lengths of calls.
- His model gave accurate predictions of the number of active calls, important for the sizing of telephone switching centers.



User i picks up the phone;
gets a dial tone;
dials the number of user j;
who picks up the phone;
they talk together;
and they hang up.

Assumptions:

The next call starts randomly in time with rate a.
A call terminates randomly in time with rate b.
State of system is n, number of calls in progress.
Number of switch points, N, is less than number of users.

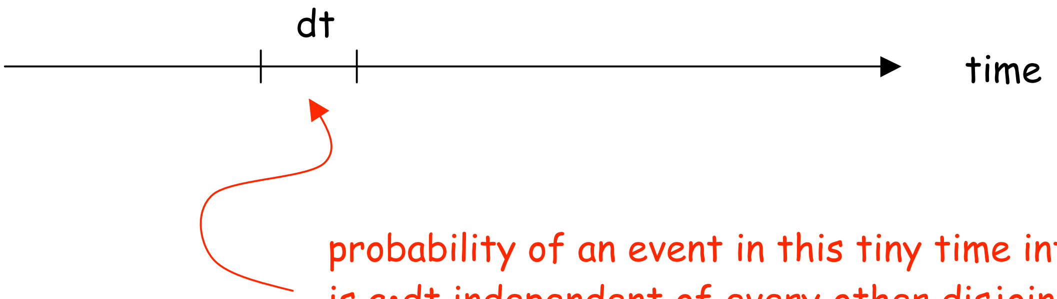
Question:

What is the probability P(n) that the system will be
the state where n calls are simultaneously in progress?

Rationale:

P(N) is probably that all N crosspoints will be in use.
No dial tone if someone attempts a call when state n > N.

What does "random rate a" mean?



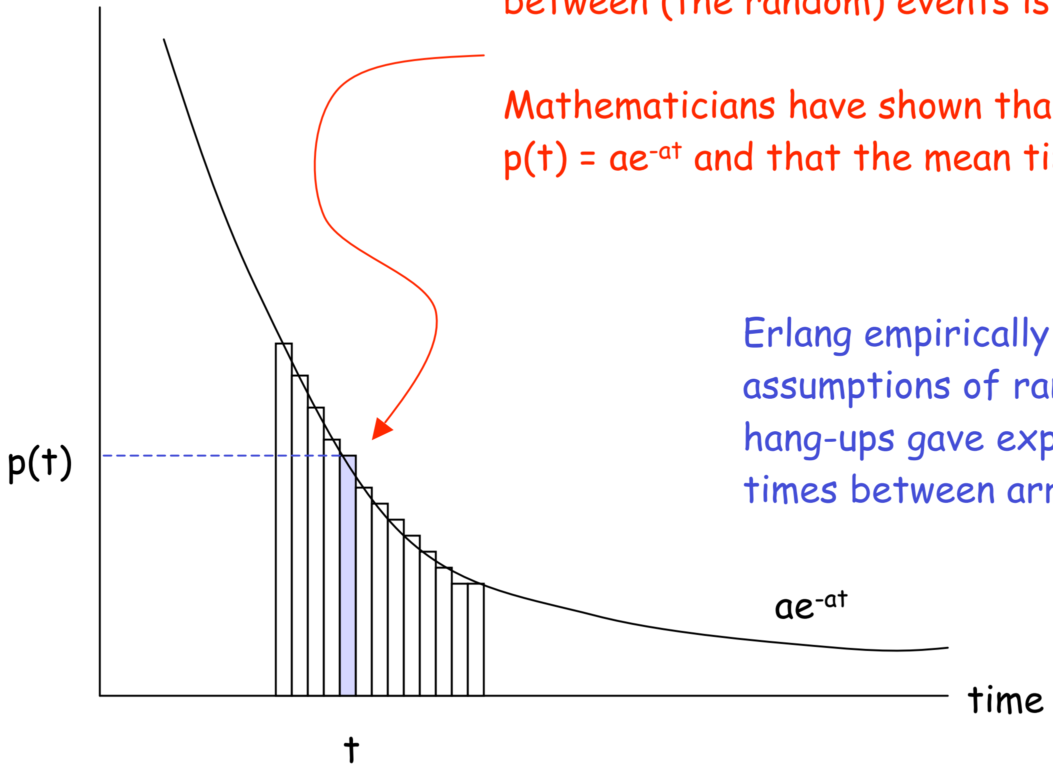
probability of an event in this tiny time interval (dt)
is a•dt independent of every other disjoint time interval

With this assumption, the histogram of times between events is exponential with parameter a and mean 1/a. (Interpretation on next picture.)

Histogram

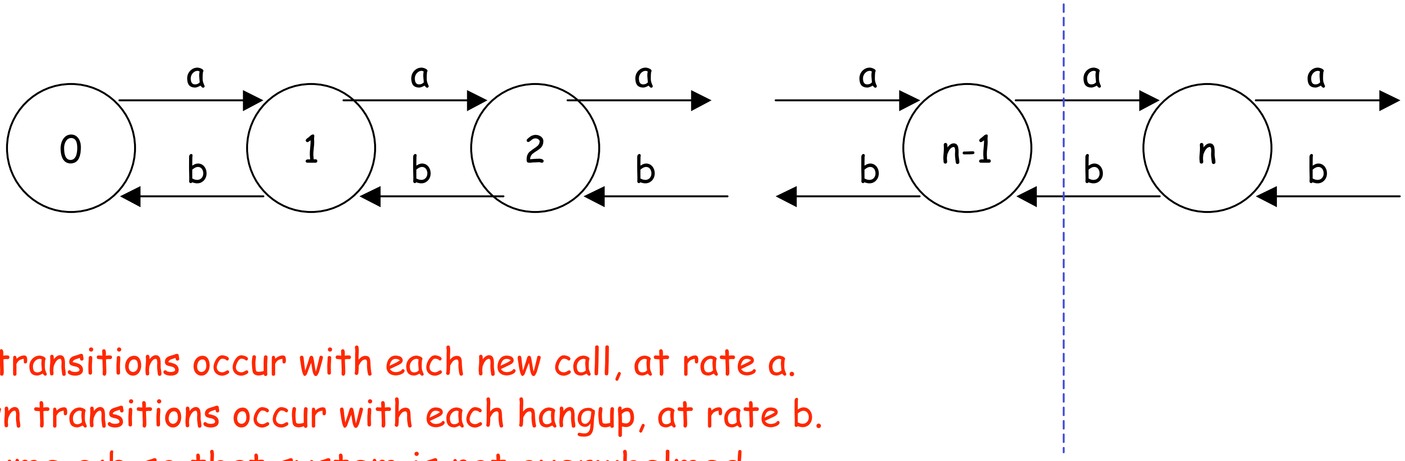
The bar represents the probability that the time between (the random) events is between t and t+dt.

Mathematicians have shown that this height must be $p(t) = ae^{-at}$ and that the mean time between events is 1/a.



Erlang empirically verified that his assumptions of random arrivals and hang-ups gave exponential distributions of times between arrivals and call holding times.

Erlang's state space



Up-transitions occur with each new call, at rate a.
Down transitions occur with each hangup, at rate b.
Assume a<b so that system is not overwhelmed.
The rates are independent of state.
There is no limit on the maximum number of calls.
Let p(n) denote the fraction of time system state = n.

At any cut, the flow up must balance the flow down, or

$p(n-1)a = p(n)b$

thus

$p(n) = (a/b)p(n-1) = (a/b)^n p(0)$

which is a geometric series. The sum of the series for all p(n) must be 1:

$\sum p(n) = p(0) \sum (a/b)^n = p(0)/(1-(a/b))$

or

$p(0) = 1-(a/b)$

It is usually easier to compute the p(n) through simple iterative methods than to evaluate a closed-form mathematical expression, especially when the mathematics allow n to become infinite whereas n is bounded in the real system.

Limit the state diagram to states 0,1,...,N. Use this procedure:

(1) Guess p(0) -- *e.g.*, set p(0)=1.
(2) Compute p(n) = p(n-1)(a/b) for n=1,...,N.
(3) Compute the sum S of the p(n). (S is called the “normalizing constant”)
(4) Replace each p(n) with p(n)/S.

Now we have a valid probability distribution: it satisfies the recursion and sums to 1.

When p(N) is small, the error between the math expression and the computer evaluation is small.

example (see next page)

new-call requests every 120 sec
(a = 1/120)

average call lasts 100 sec
(b = 1/100)

What is the median number of active calls?
(3)

What is probability that the telephone exchange is saturated?
(0.07%)

What is the probability that the telephone exchange is idle?
(16%)

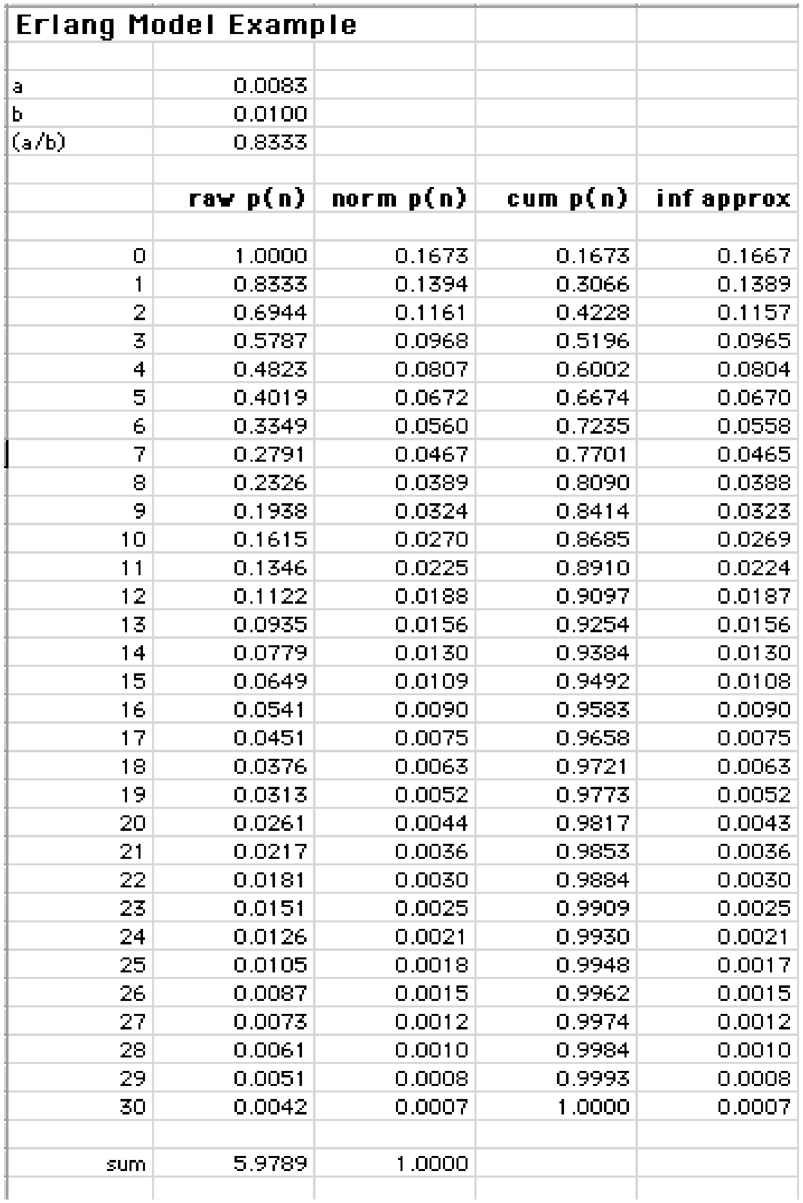
What is the 90th percentile of the number of active calls?
(11)

raw p(n): "guess" p(0)=1, then compute each new p(n) = (a/b)p(n-1). Gets ratios right.

norm p(n): divide each raw p(n) by the sum of all p(n). Now they all add up to 1 and have the proper ratios.

cum p(n): the cumulative sum of p(0)+…+p(n). Shows approach to 1.0 as n increases.

inf approx: pretends n goes to infinity. Starts with p(0) = 1-a/b and uses the same recursion.



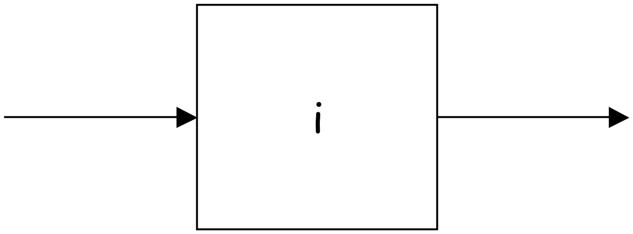
Erlang Model Example

a	0.0083			
b	0.0100			
(a/b)	0.8333			
	raw p(n)	**norm p(n)**	**cum p(n)**	**inf approx**
0	1.0000	0.1673	0.1673	0.1667
1	0.8333	0.1394	0.3066	0.1389
2	0.6944	0.1161	0.4228	0.1157
3	0.5787	0.0968	0.5196	0.0965
4	0.4823	0.0807	0.6002	0.0804
5	0.4019	0.0672	0.6674	0.0670
6	0.3349	0.0560	0.7235	0.0558
7	0.2791	0.0467	0.7701	0.0465
8	0.2326	0.0389	0.8090	0.0388
9	0.1938	0.0324	0.8414	0.0323
10	0.1615	0.0270	0.8685	0.0269
11	0.1346	0.0225	0.8910	0.0224
12	0.1122	0.0188	0.9097	0.0187
13	0.0935	0.0156	0.9254	0.0156
14	0.0779	0.0130	0.9384	0.0130
15	0.0649	0.0109	0.9492	0.0108
16	0.0541	0.0090	0.9583	0.0090
17	0.0451	0.0075	0.9658	0.0075
18	0.0376	0.0063	0.9721	0.0063
19	0.0313	0.0052	0.9773	0.0052
20	0.0261	0.0044	0.9817	0.0043
21	0.0217	0.0036	0.9853	0.0036
22	0.0181	0.0030	0.9884	0.0030
23	0.0151	0.0025	0.9909	0.0025
24	0.0126	0.0021	0.9930	0.0021
25	0.0105	0.0018	0.9948	0.0017
26	0.0087	0.0015	0.9962	0.0015
27	0.0073	0.0012	0.9974	0.0012
28	0.0061	0.0010	0.9984	0.0010
29	0.0051	0.0008	0.9993	0.0008
30	0.0042	0.0007	1.0000	0.0007
sum	5.9789	1.0000		

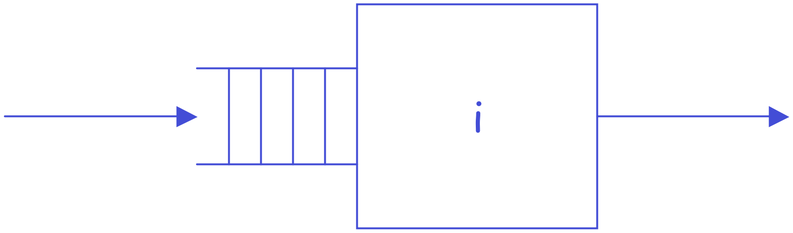
Servers

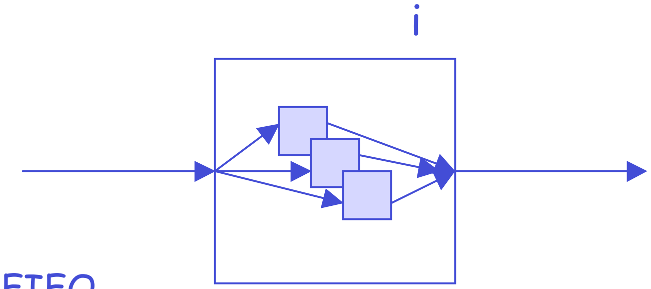
- Server is a station that satisfies certain tasks within jobs.
- Has one or more internal parallel processors (we assume one).
- Has a queueing mechanism to make tasks not in service wait.
- Has input point for task arrivals.
- Has output point for task completions.

simple notation for single server
with FIFO queueing



more complex notation for single server
with FIFO queueing, showing the queue.





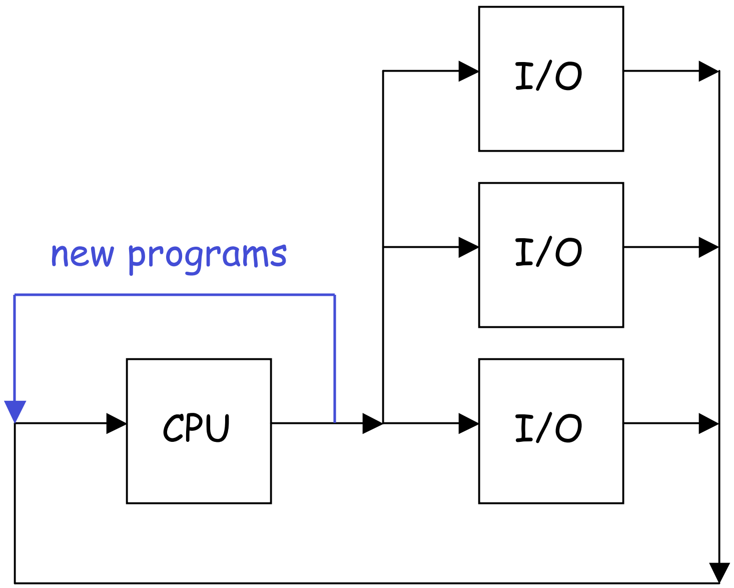
notation for multi-server with FIFO
queueing, showing internal processors.

Network of Servers

Set of servers with interconnection pathways.

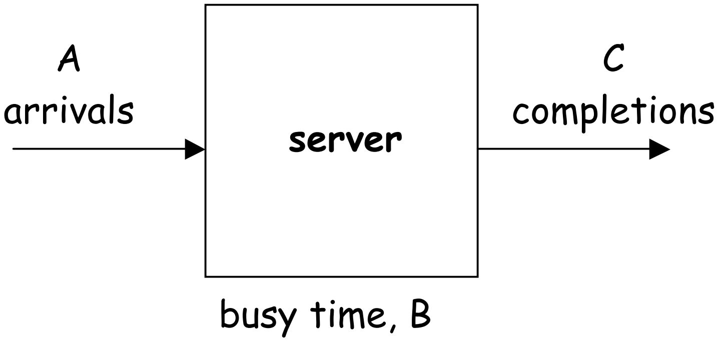
Open or closed.

Closed network includes all its customers in a finite population of N jobs.



© 2002, P. J. Denning

Measuring a Server


A
arrivals

server

C
completions

busy time, B

observation period: T

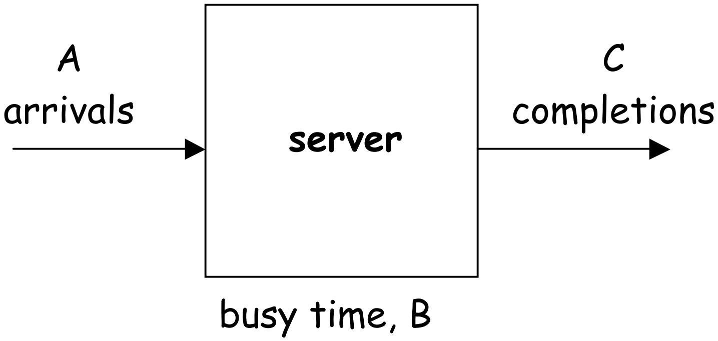
arrival rate: λ = A/T

completion rate: X = C/T

utilization: U = B/T

mean service time: S = B/C

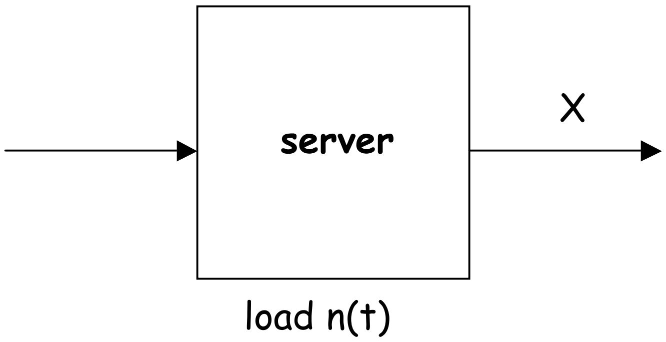
Measuring a Server

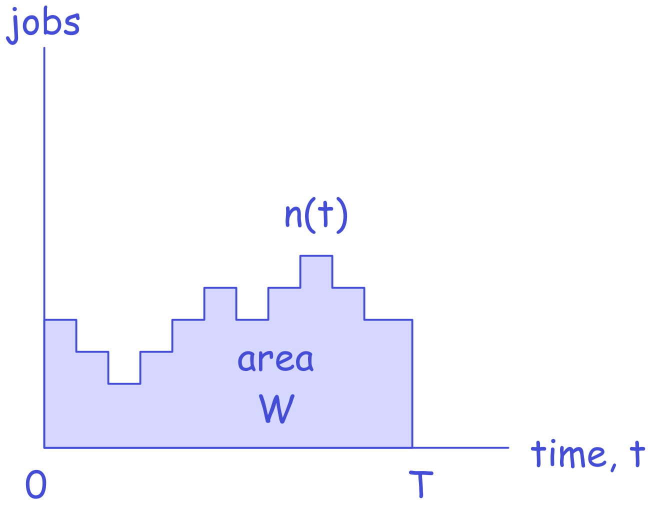


Flow balance: A=C

Utilization law: U = SX

Measuring a Server



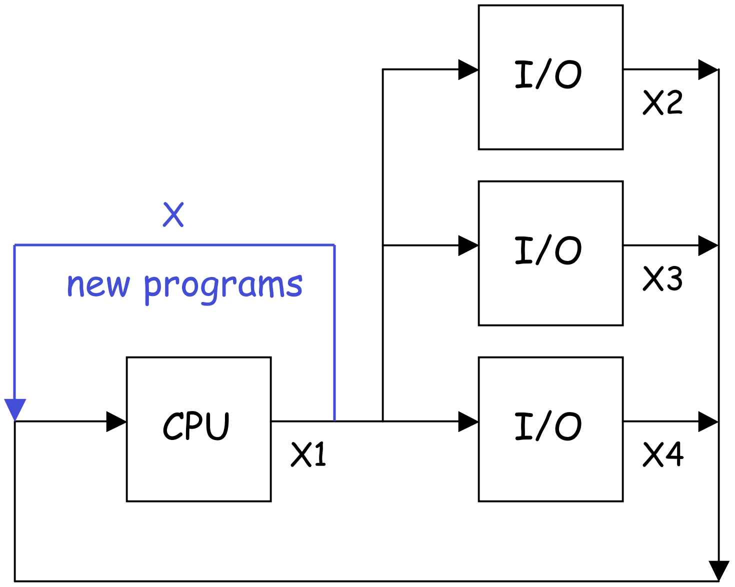


Mean load: $Q = W/T$

Mean response time: $R = W/C$

Little's Law: $Q = RX$

Measuring a Network



job = sequence of tasks,
each at one server
=> job visits servers one at a time

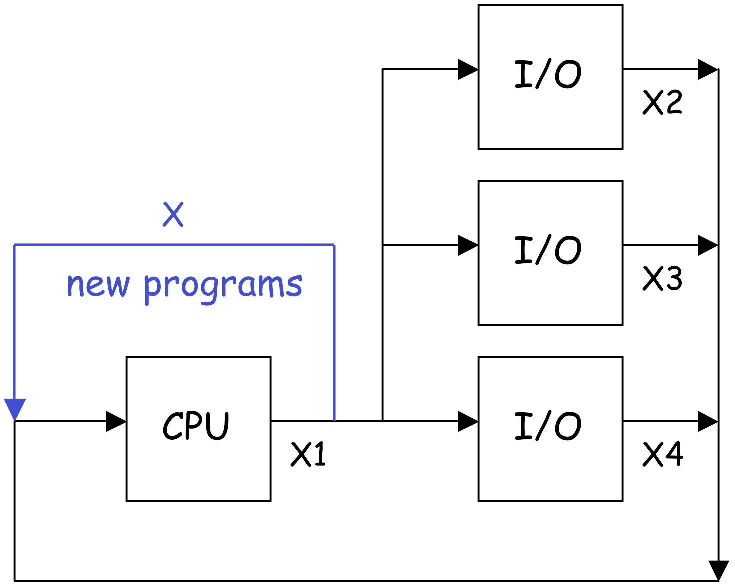
C0 = jobs leaving system along
"new programs" path
(system throughput X = C0/T)

Ci = jobs leaving server i
(server throughput Xi = Ci/T)

Visit ratio Vi
= Ci/C0
= number of visits to
server i per job

Forced Flow Law: Xi = Vi X0

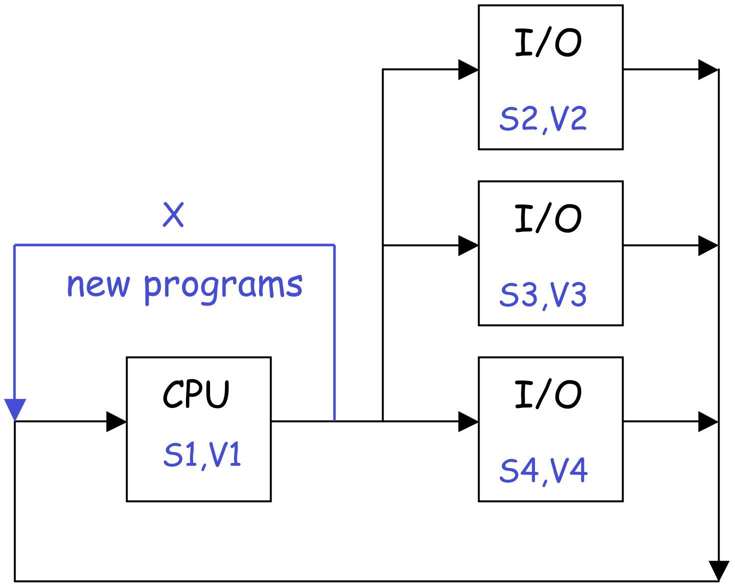
Measuring a Network



Flow at any one point in the system determines flow everywhere

Forced Flow Law: Xi = Vi X

Measuring a Network -- Central Server System Example



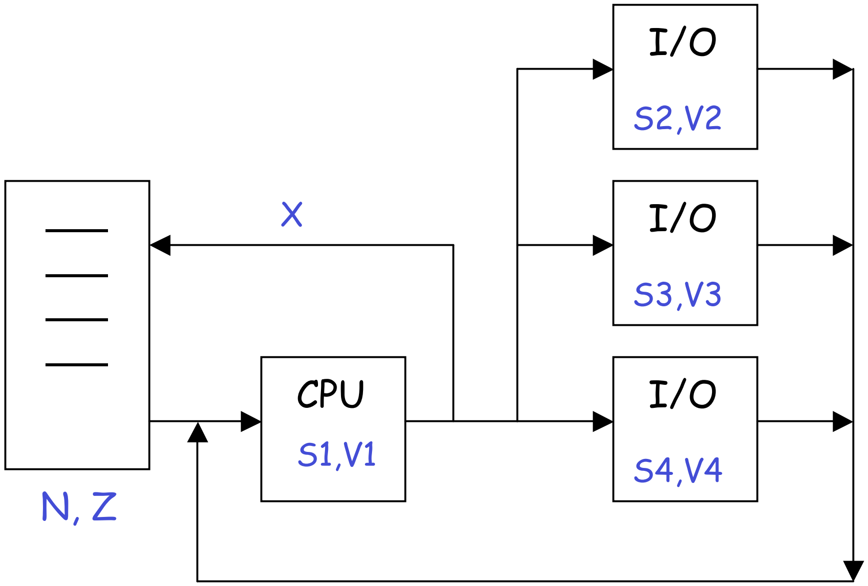
parameters of system:

Si = mean service time per visit to server i

Vi = mean number of visits to server i

N = total number of jobs in the system

Measuring a Network -- Time Sharing System Example



parameters of system:

Si = mean service time
per visit to server i

Vi = mean number of visits
to server i

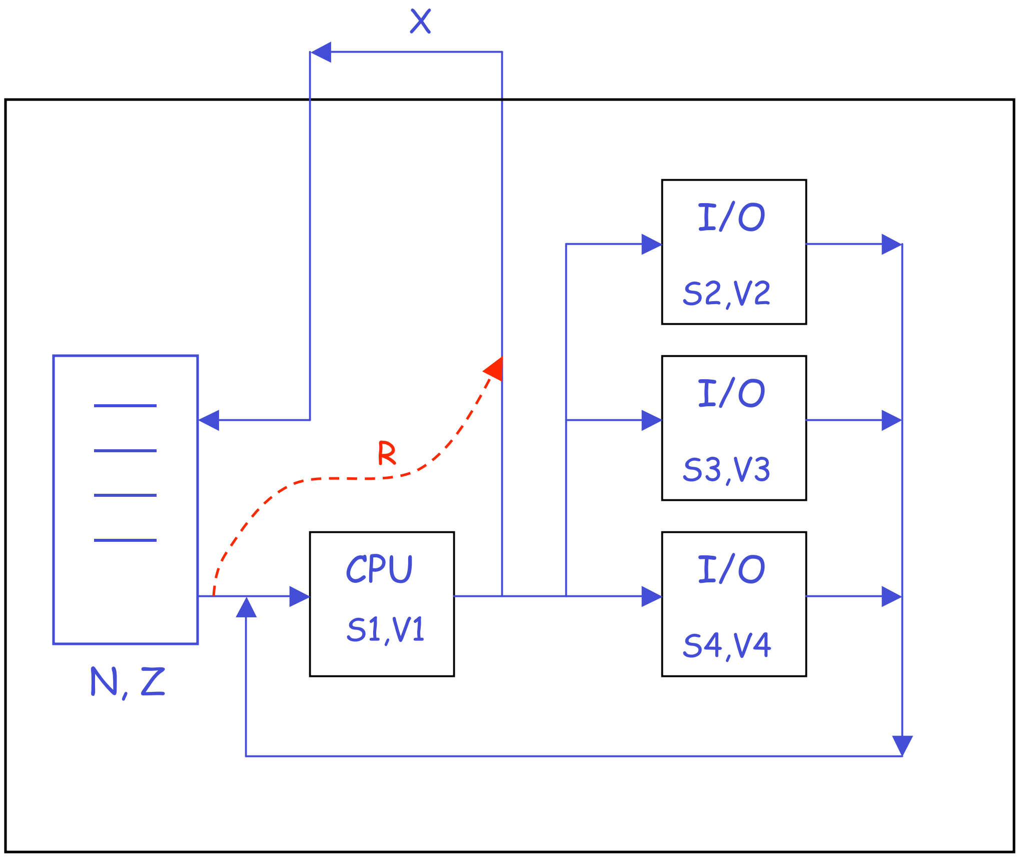
N = total number of jobs
in the system

Z = mean think time between
requests for the system

User model: (think, wait)*

Execution model: (CPU)(I/O, CPU)*

Measuring a Network -- Time Sharing System Example



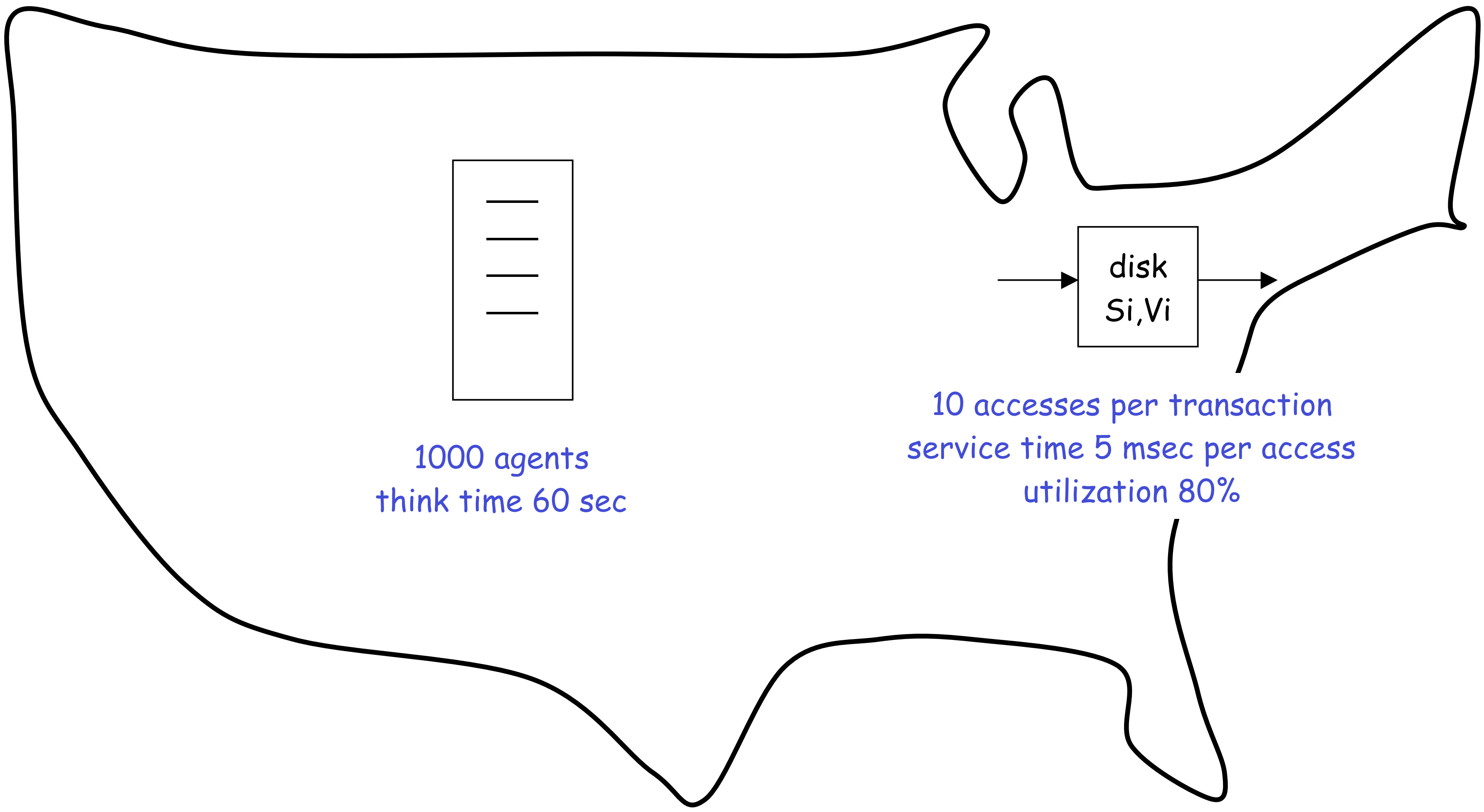
Little's Law for the entire box says

$$N = (R+Z)\ X$$

Response Time Law:
R = N/X - Z

© 2002, P. J. Denning

Airline Reservations System Example



disk
Si,Vi

1000 agents
think time 60 sec

10 accesses per transaction
service time 5 msec per access
utilization 80%

disk throughput:

$X_i = U_i/S_i$
$= 0.8/0.005$
$= 160$ tasks/sec

system throughput:

$X = X_i/V_i$
$= 160/10$
$= 16$ transactions/sec

response time:

$R = N/X - Z$
$= 1000/16 - 60$
$= 62.5 - 60$
$= 2.5$ sec

the throughput and response time
can be answered exactly
using the operational laws

Prediction in Airline Reservations System Example

What if disk access method were changed to reduce accesses to 8 per transaction?

Well ...

Xi = 160 accesses per second

X = Xi/Vi = 160/8 = 20 transactions per second

R = 1000/20 - 60 = 50 - 60 = -10 ???

The problem is that changing disk accesses affects relative demand for other servers, which in turn affects flow reaching the disk, affecting its utilization.

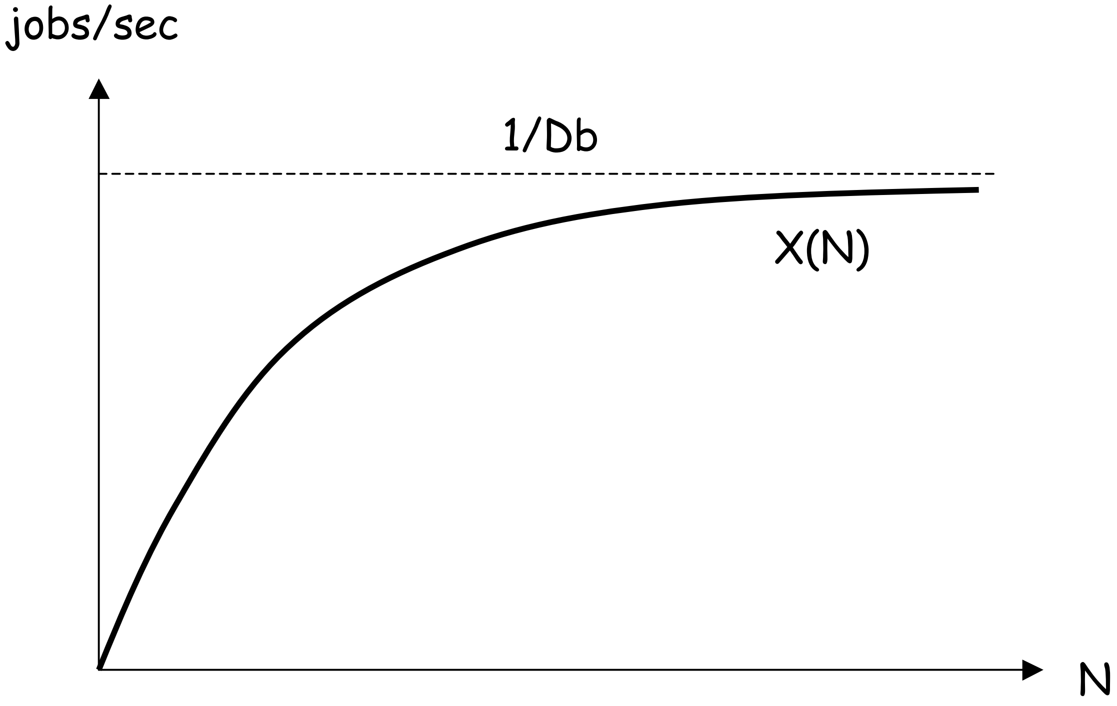
How it does so depends on parameters of the other servers.

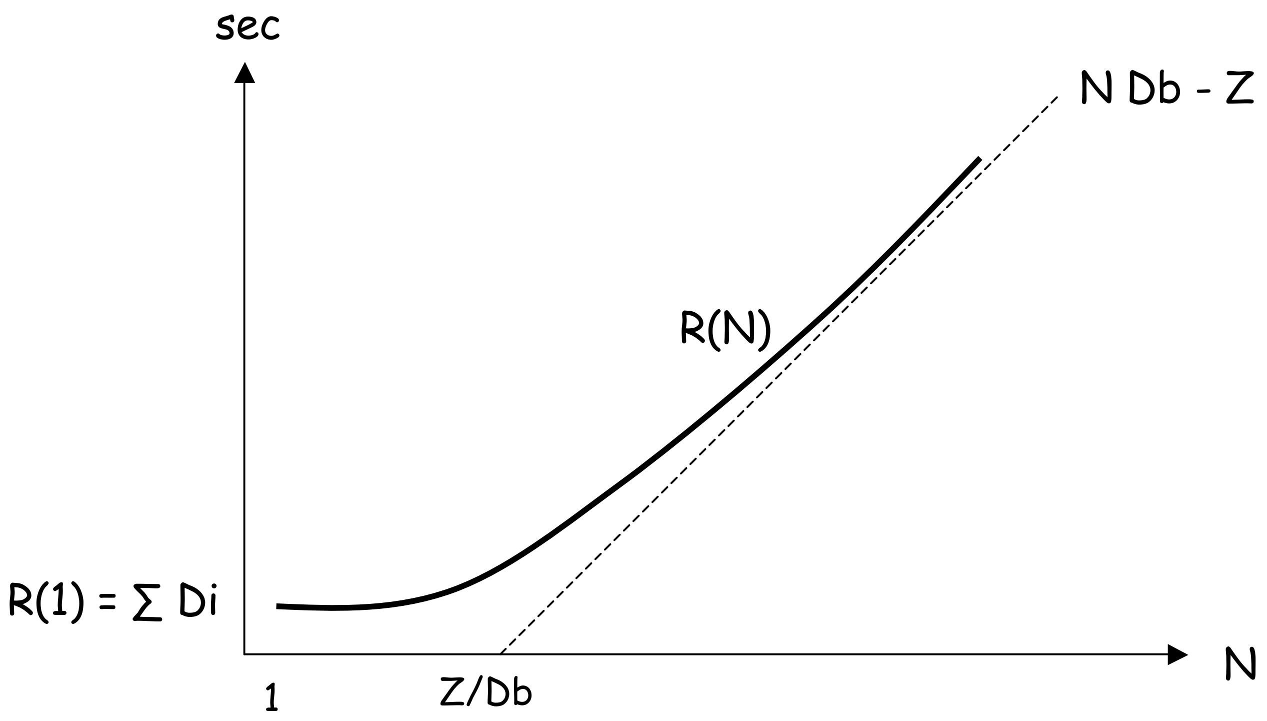
Cannot do predictions without knowledge of the whole system.

The simplest prediction method is bottleneck analysis.

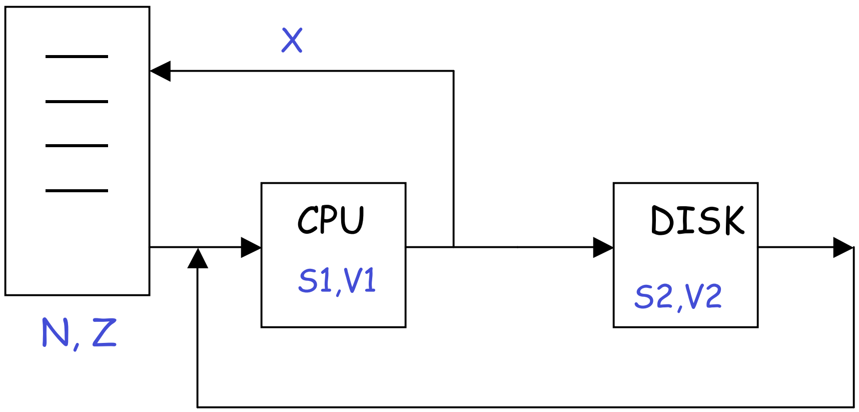
Bottleneck Analysis

- Bottleneck: a choke point in the system's flow structure -- tasks pile up there because they flow past too slowly.
- Utilization and forced flow laws tell that $Ui = XiSi = ViSiX = DiX$. (Define demand $Di = ViSi$.)
- For given X, servers with larger demand Di have higher utilizations; server with highest demand Di has highest utilization.
- Bottleneck server b is one for which $Db = \max\{Di\}$.
- Since utilizations cannot exceed 1, server with highest demand limits throughput: $X = Ub/Db \leq 1/Db$.
- This also limits response time: $R = N/X - Z \geq N\,Db - Z$.


jobs/sec
1/Db
X(N)
N


sec
N Db - Z
R(N)
R(1) = ∑ Di
N
1
Z/Db

Bottleneck Example



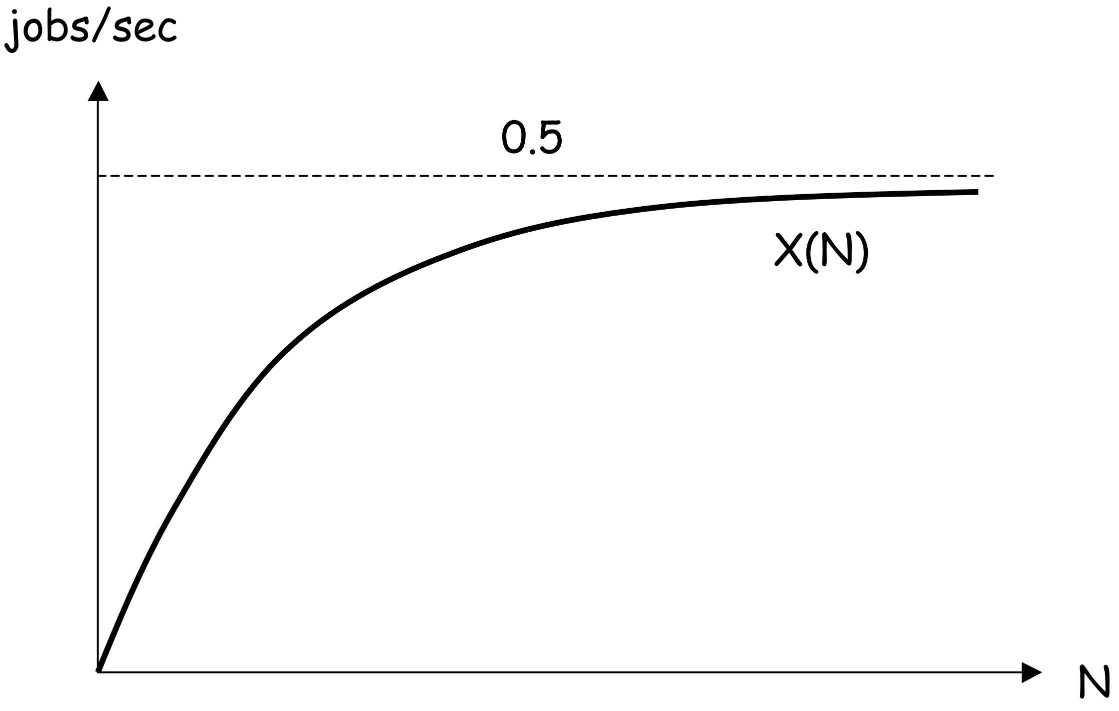
Given:

CPU time per job 1 sec
100 disk accesses per job
20 msec per disk access
think time 30 sec

What are model parameters?

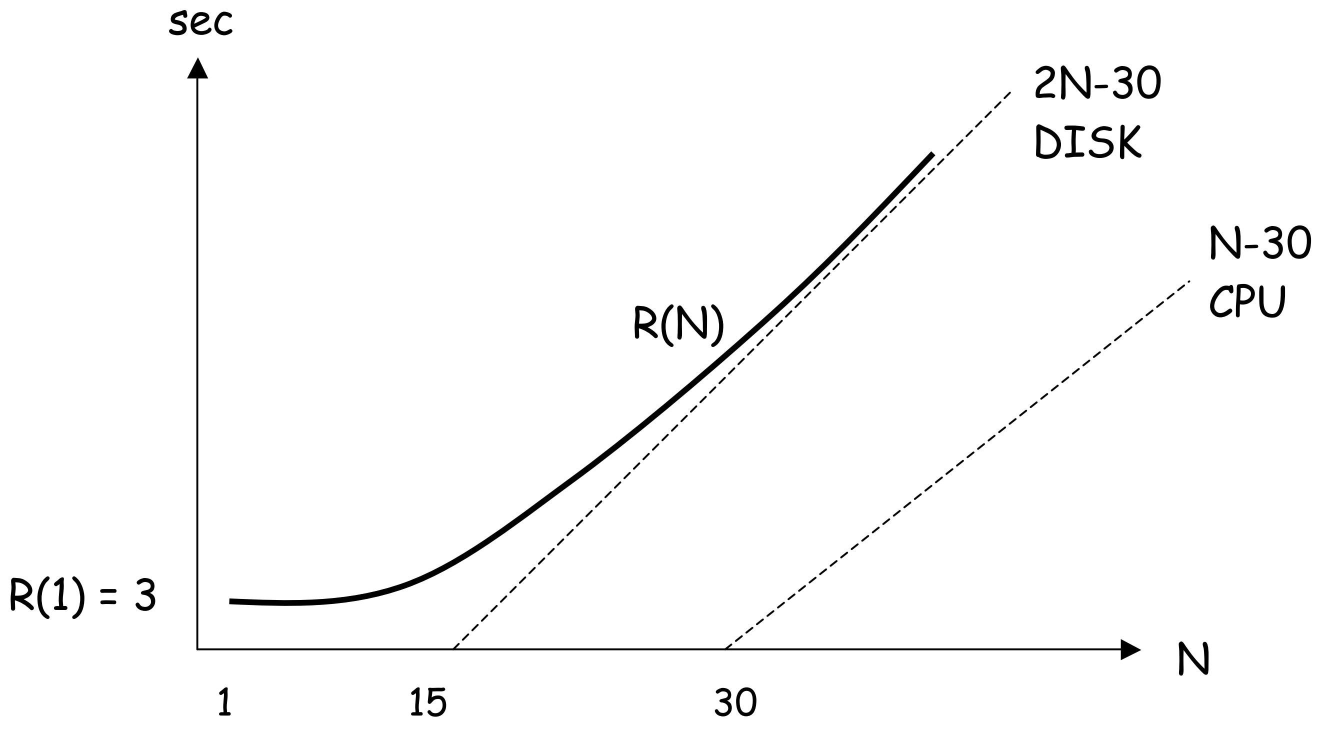
V2 = 100
S2 = 0.02 sec
V1 = 1+V2 (why?) = 101
S1 = D1/V1 = 1/101 = 0.0099 sec.
D1 = V1S1 = 1 sec.
D2 = V2S2 = (100)(0.02) = 2 sec.
Z = 30

Bottleneck Example --throughput asymptotes



DISK is the bottleneck
b=2 and Db = 2.
X ≤ 1/Db = 0.5

Bottleneck Example --response time asymptotes



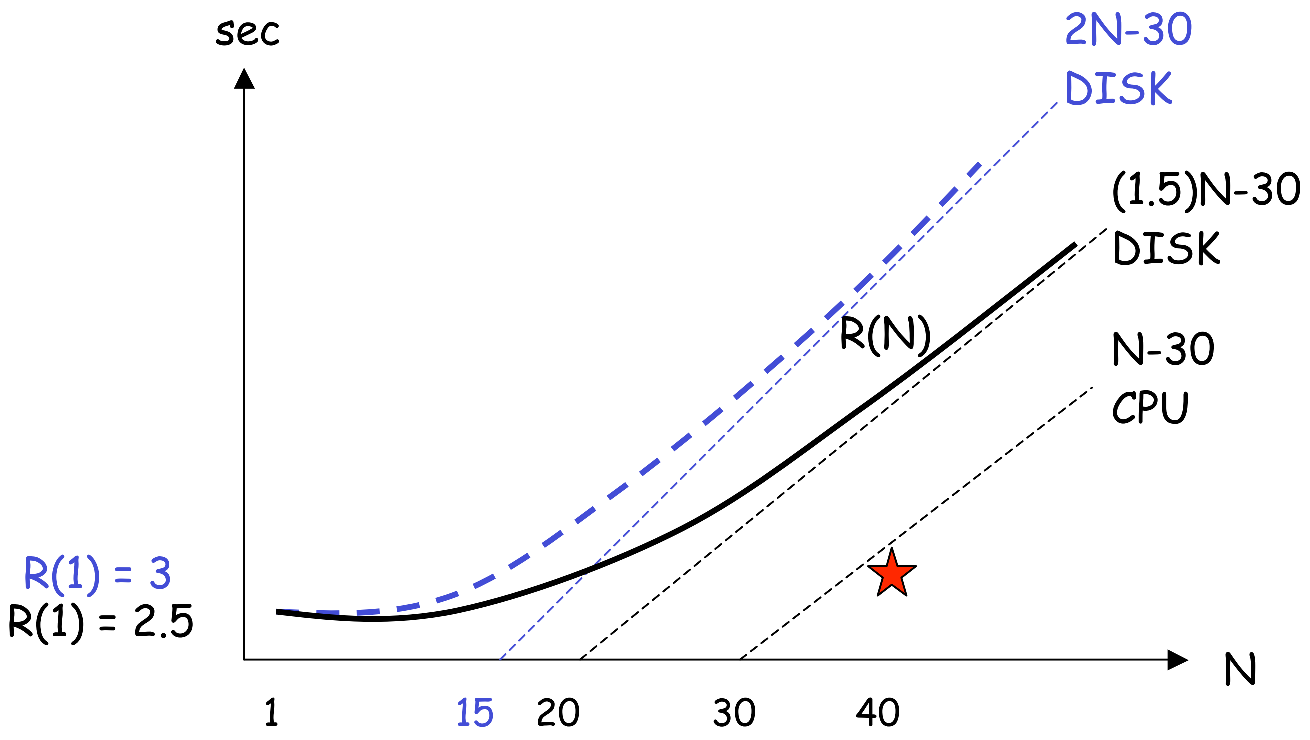
© 2002, P. J. Denning

Bottleneck Example

Faster disk has access time 15 msec.
Is 5-sec response time feasible with
40 users?

Change S2 to 0.015
Now D2 = (100)(0.015) = 1.5
DISK is still bottleneck (D1 = 1.0)
$R(N) \geq NDb-Z = (40)(1.5)-30 = 30$ sec.

No, 5-sec response time not feasible.


sec
2N-30
DISK
(1.5)N-30
DISK
R(N)
N-30
CPU
R(1) = 3
R(1) = 2.5
N
1
15
20
30
40

Bottleneck Example

New index structure reduces disk accesses to 50 on the faster disk. Is 5-sec response time feasible with 40 users?

Change V2 to 50

Now D2 = (50)(0.015) = 0.75

Now CPU is bottleneck (D1 = 1.0)

$R(N) \geq ND_b - Z = (40)(1) - 30 = 10$ sec.

No, 5-sec response time not feasible. To achieve it, need to speed up the CPU.

Bottleneck Example

Use 2x faster CPU plus the improved disk. Is 5-sec response time feasible with 40 users?

Change D1 to 0.5 sec.
Now DISK is bottleneck (D2 = 0.75)
$R(N) \geq NDb-Z = (40)(0.75)-30 = 0$ sec.
Also $R(N) \geq R(1) = D1+D2 = 1+0.75 = 1.75$

Yes, 5-sec response time is feasible.

Bottleneck Example

When speeding up a bottleneck,
watch our for the next bottleneck.

Response time objective may need several servers to become faster so that all potential bottlenecks are have their asymptotes to the right of the desired operating point.

Computational Algorithms

- Bottleneck analysis useful to discover if desired operating points are in feasible regions and tell which servers need additional capacity.
- What algorithm can we use to calculate the entire curve of R(N)?
- The Mean Value Analysis (MVA) algorithm does this.

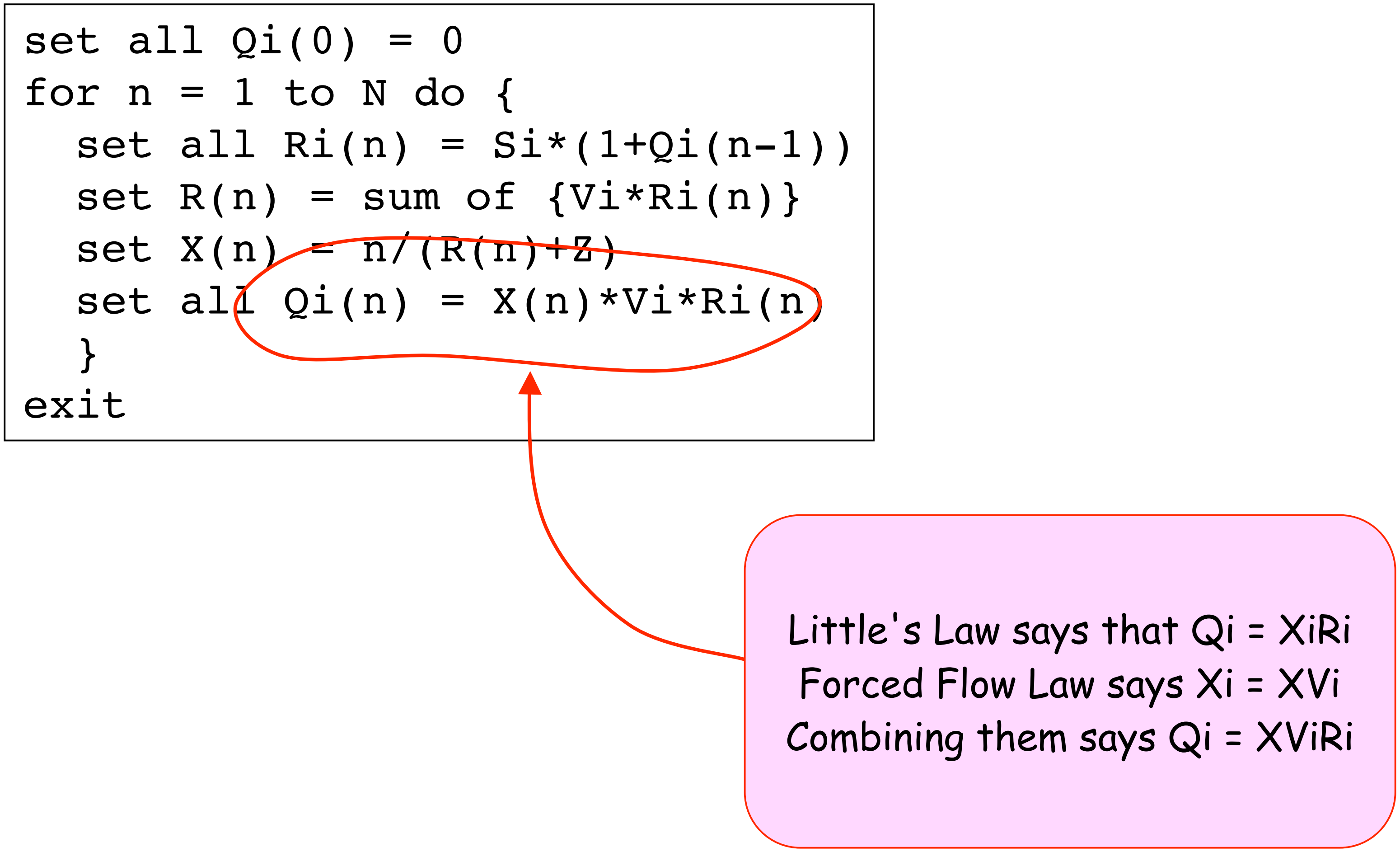
Mean Value Analysis (MVA)

- MVA algorithm calculates several mean values together --
  - Ri = mean response time per visit to server i
  - R = mean service time per visit to the system
  - X = throughput of the system
  - Qi = mean queue length at server i
- MVA does this for n = 0, 1, ... , N.
- The set of mean values for n-1 is used to compute the set of mean values for n.

MVA:

```
set all Qi(0) = 0
for n = 1 to N do {
  set all Ri(n) = Si*(1+Qi(n-1))
  set R(n) = sum of {Vi*Ri(n)}
  set X(n) = n/(R(n)+Z)
  set all Qi(n) = X(n)*Vi*Ri(n)
  }
exit
```

MVA:



```
set all Qi(0) = 0
for n = 1 to N do {
  set all Ri(n) = Si*(1+Qi(n-1))
  set R(n) = sum of {Vi*Ri(n)}
  set X(n) = n/(R(n)+Z)
  set all Qi(n) = X(n)*Vi*Ri(n)
  }
exit
```

Little's Law says that Qi = XiRi
Forced Flow Law says Xi = XVi
Combining them says Qi = XViRi

MVA:

```
set all Qi(0) = 0
for n = 1 to N do {
  set all Ri(n) = Si*(1+Qi(n-1))
  set R(n) = sum of {Vi*Ri(n)}
  set X(n) = n/(R(n)+Z)
  set all Qi(n) = X(n)*Vi*Ri(n)
  }
exit
```

Response Time Law
says that X(R+Z) = N

MVA:

```
set all Qi(0) = 0
for n = 1 to N do {
  set all Ri(n) = Si*(1+Qi(n-1))
  set R(n) = sum of {Vi*Ri(n)}
  set X(n) = n/(R(n)+Z)
  set all Qi(n) = X(n)*Vi*Ri(n)
  }
exit
```

Response time is the sum of per-visit server response times weighted by number of visits to each server.

MVA:

```
set all Qi(0) = 0
for n = 1 to N do {
  set all Ri(n) = Si*(1+Qi(n-1))
  set R(n) = sum of {Vi*Ri(n)}
  set X(n) = n/(R(n)+Z)
  set all Qi(n) = X(n)*Vi*Ri(n)
  }
exit
```

Per-visit server response time at a server is the arriver's mean service plus the mean service needed by everyone queued in front of the arriver.
The arriver sees the same mean queue as would outside observer in a system with one less job (arriver removed).

Alternate Form MVA:

Define Residence Time Ti(n) = ViRi(n)

```
set all Qi(0) = 0
for n = 1 to N do {
  set all Ti(n) = Di*(1+Qi(n-1))
  set R(n) = sum of {Ti(n)}
  set X(n) = n/(R(n)+Z)
  set all Qi(n) = X(n)*Ti(n)
  }
exit
```

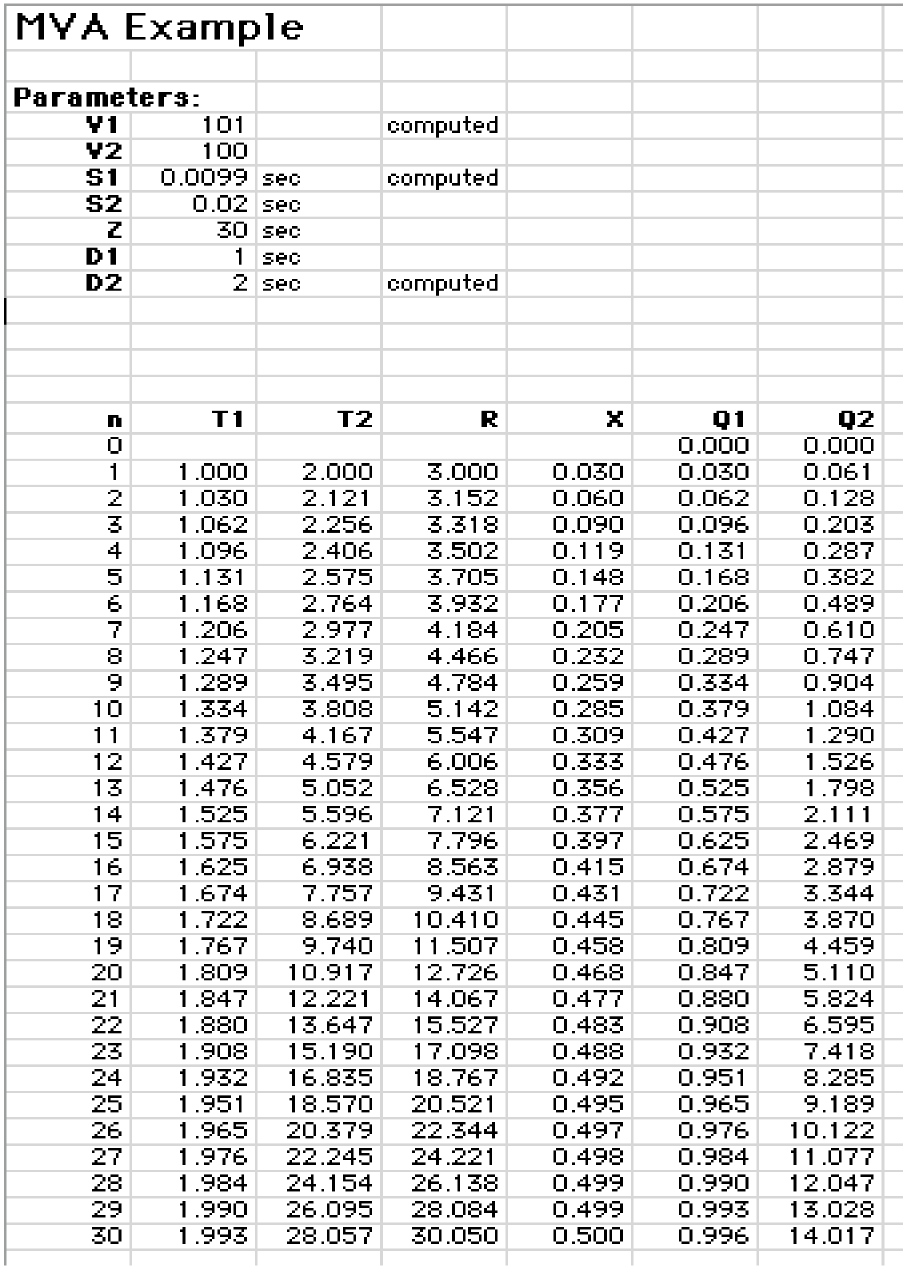
n	T1	...	TK	R	X	Q1	...	QK
0	0	...	0					
1	1	...	K	K+1	K+2	K+3	...	2K+2
2								
3								
...								

Having filled in one row, the algorithm fills in values in the next row in the order indicated by the numbers 1, ..., 2K+2.

When done, the R-column contains the complete curve R(n). Same for X-column.

Note that X approaches a saturation value, 1/D1 = 0.5

Note that R approaches a saturation line, n*Db-Z = 2*n-30

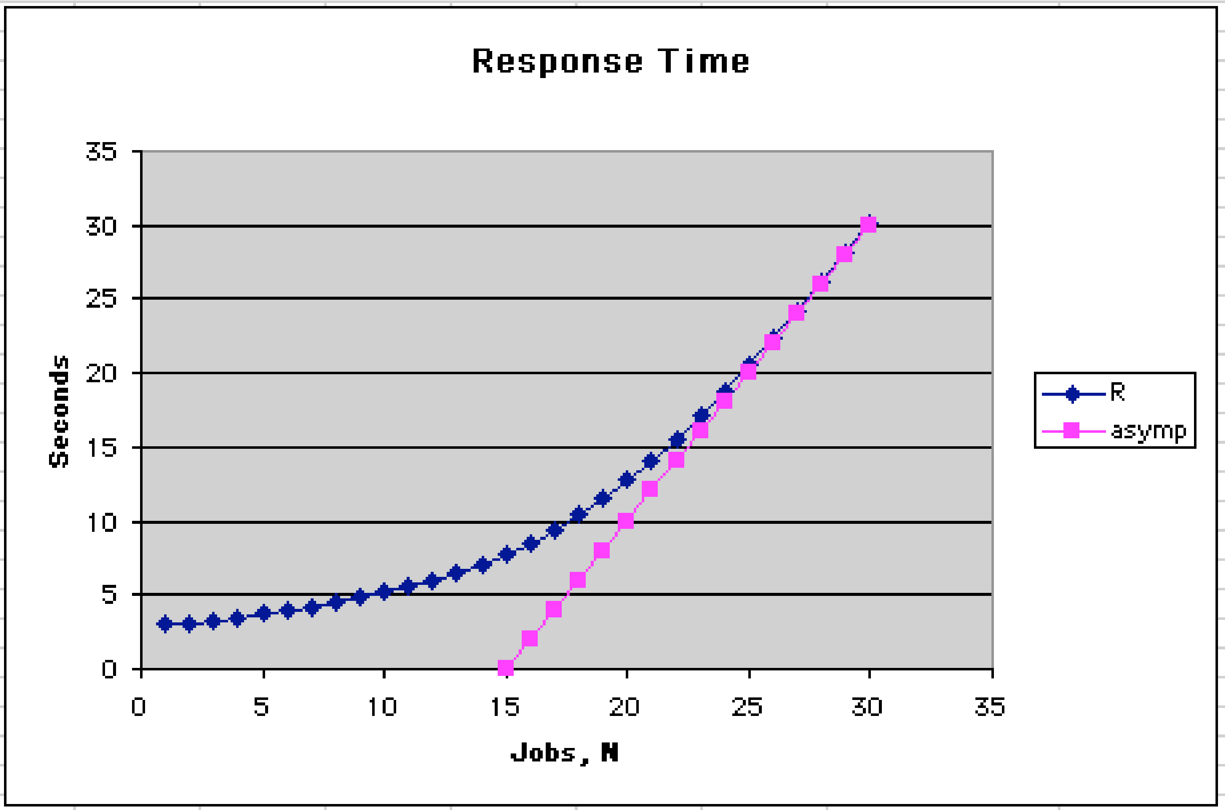


MVA Example

Parameters:

V1	101		computed
V2	100		
S1	0.0099	sec	computed
S2	0.02	sec	
Z	30	sec	
D1	1	sec	
D2	2	sec	computed

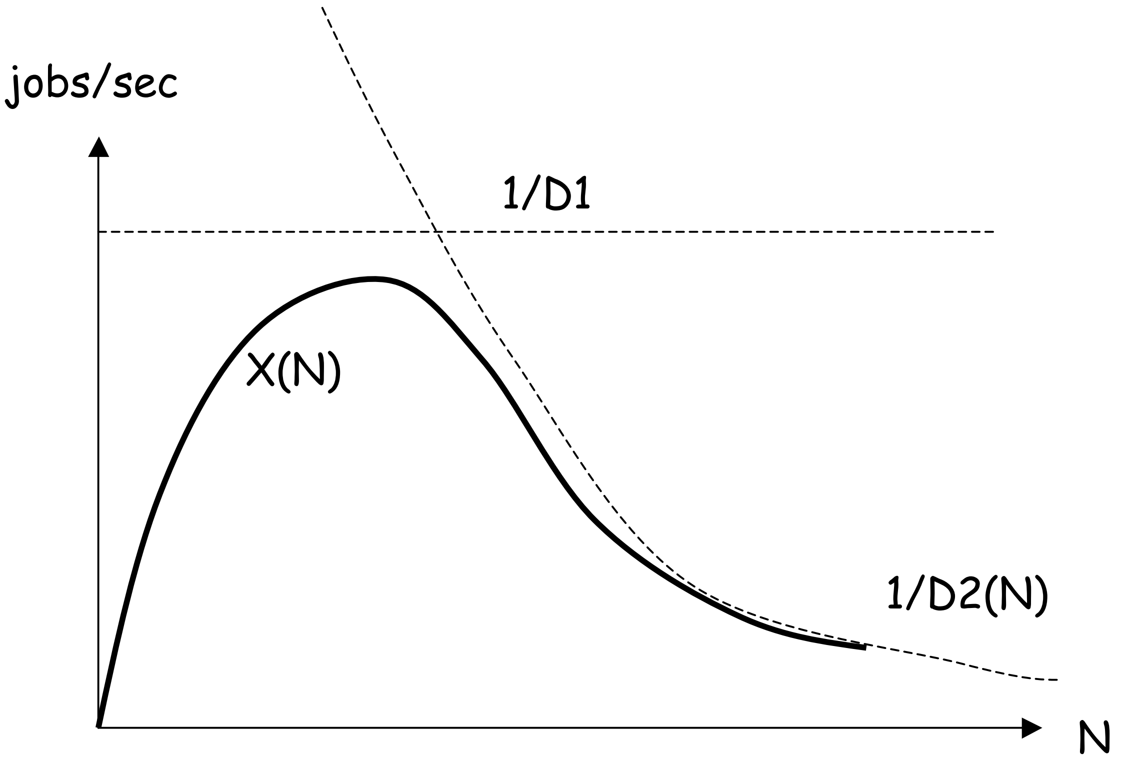
n	**T1**	**T2**	**R**	**X**	**Q1**	**Q2**
0					0.000	0.000
1	1.000	2.000	3.000	0.030	0.030	0.061
2	1.030	2.121	3.152	0.060	0.062	0.128
3	1.062	2.256	3.318	0.090	0.096	0.203
4	1.096	2.406	3.502	0.119	0.131	0.287
5	1.131	2.575	3.705	0.148	0.168	0.382
6	1.168	2.764	3.932	0.177	0.206	0.489
7	1.206	2.977	4.184	0.205	0.247	0.610
8	1.247	3.219	4.466	0.232	0.289	0.747
9	1.289	3.495	4.784	0.259	0.334	0.904
10	1.334	3.808	5.142	0.285	0.379	1.084
11	1.379	4.167	5.547	0.309	0.427	1.290
12	1.427	4.579	6.006	0.333	0.476	1.526
13	1.476	5.052	6.528	0.356	0.525	1.798
14	1.525	5.596	7.121	0.377	0.575	2.111
15	1.575	6.221	7.796	0.397	0.625	2.469
16	1.625	6.938	8.563	0.415	0.674	2.879
17	1.674	7.757	9.431	0.431	0.722	3.344
18	1.722	8.689	10.410	0.445	0.767	3.870
19	1.767	9.740	11.507	0.458	0.809	4.459
20	1.809	10.917	12.726	0.468	0.847	5.110
21	1.847	12.221	14.067	0.477	0.880	5.824
22	1.880	13.647	15.527	0.483	0.908	6.595
23	1.908	15.190	17.098	0.488	0.932	7.418
24	1.932	16.835	18.767	0.492	0.951	8.285
25	1.951	18.570	20.521	0.495	0.965	9.189
26	1.965	20.379	22.344	0.497	0.976	10.122
27	1.976	22.245	24.221	0.498	0.984	11.077
28	1.984	24.154	26.138	0.499	0.990	12.047
29	1.990	26.095	28.084	0.499	0.993	13.028
30	1.993	28.057	30.050	0.500	0.996	14.017


Response Time
35
30
25
20
15
10
5
0
Seconds
0
5
10
15
20
25
30
35
Jobs, N
R
asymp

Models for Multiprogramming

- In a virtual memory system the number of page faults generated by a job depends on how much space allocated to it.
- Crude model: assume the memory of M pages is equally allocated on average among N jobs; thus each has an average of M/N pages.
- Then set V2 (visits to paging disk) to F(M/N), where F is the fault function for the workload.

- This means D2 is a function of N.
- As N increases, F(M/N) increases (less space), and D2(N) increases.
- The throughput bound is now min{1/D1, 1/D2(N)}.
- For small N CPU may be bottleneck and $X(N) \leq 1/D1$.
- For large N, DISK is the bottleneck and $X(N) \leq 1/D2(N)$.

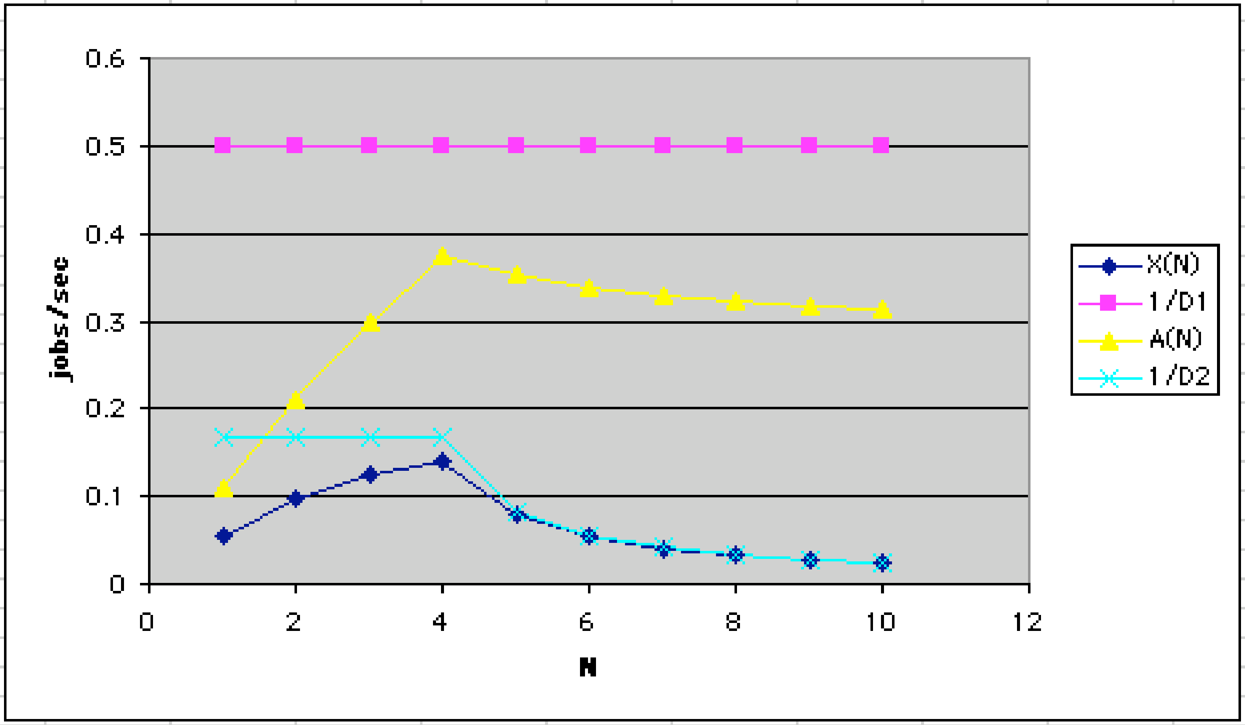


Optimal level of multiprogramming occurs near N that makes D1 = D2(N) (if bounds cross).

Bottlenecks explain thrashing.

Can use the MVA model to evaluate. To calculate X(N), set D2=D2(N) and compute X(n) for n=1,...,N with that fixed value of D2. Repeat for each value of N.

(Model assumes that the demands are constant across all rows; fails if this is not so.)



Example output from a model in which D2 is always larger than D1 and thus the bounds do not cross. However, X(N) still shows a peak and thrashing. (A(N) is an approximation that does not work well.)