CS 147:
Computer Systems Performance Analysis
Networks of Queues

CS 147:
Computer Systems Performance Analysis
Networks of Queues

Overview

Overview

Types of Networks

Queues in Computer Systems
Operational Quantities
Operational Laws
Bottleneck Analysis

Tricks for Solving Networks
Mean Value Analysis
Hierarchical Decomposition

Limitations

Networks of Queues

- Many systems consist of interconnected queueing systems
  - CPU→disk→network
  - Web client→Web server→Web client
  - Network of freeways
- Fortunate property: M/M/m queues have Poisson departures
  - ⇒ Next queue is M/*/m
  - Usually, we assume Poisson service times to make everything simple

Networks of Queues

- Many systems consist of interconnected queueing systems
  - CPU→disk→network
  - Web client→Web server→Web client
  - Network of freeways
- Fortunate property: M/M/m queues have Poisson departures
  - ⇒ Next queue is M/*/m
  - Usually, we assume Poisson service times to make everything simple

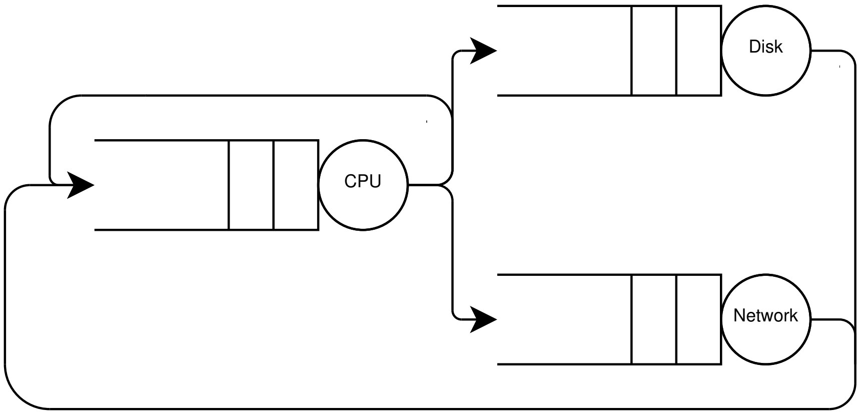
Open and Closed Networks

- Closed network recirculates jobs
- Open network has external arrivals and departures
  - May also allow recycling
- Mixed networks also possible

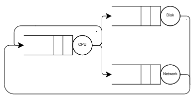
Open and Closed Networks

- Closed network recirculates jobs
- Open network has external arrivals and departures
  - May also allow recycling
- Mixed networks also possible

An Example Closed Network

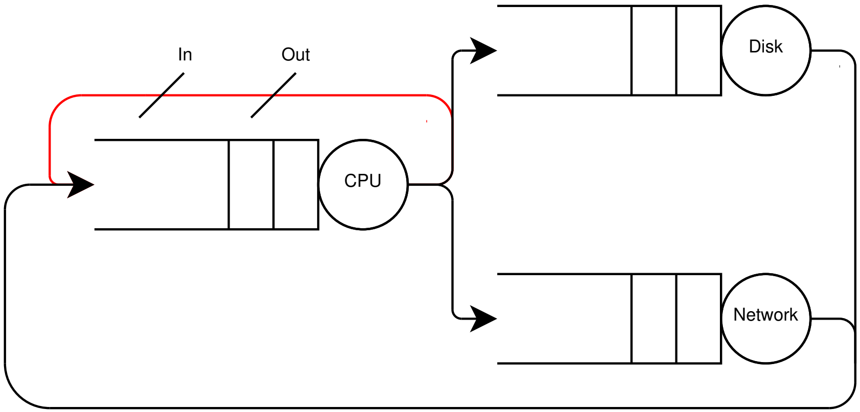


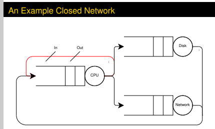
An Example Closed Network



A closed network can be converted into an open one by cutting any arbitrary flow path; see next slide.

An Example Closed Network

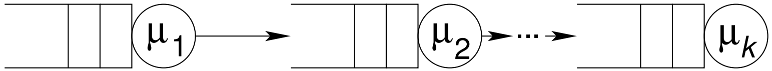




A closed network can be considered as an open network in which jobs leaving "Out" immediately reenter "In", i.e., an equilibrium network in which $\mu_{\text{Out}} = \lambda_{\text{In}}$.

Product-Form Networks

- We are interested in $P(n_1, n_2, \ldots, n_k)$, i.e., the probability that there are n_1 customers in the first queue, n_2 in the second, etc.
- Consider simple linear network:



- Arrival rate for each queue is λ (why?)
- Utilization $\rho_i = \lambda/\mu_i$
- P(n_i jobs in i^{th} queue $= p_i(n_i) = (1-\rho_i)\rho_i^{n_i}$
- $P(n_1, n_2, \ldots, n_k) = p_1(n_1)p_2(n_2)\cdots p_k(n_k)$

2015-06-15

CS147
└Types of Networks
└Product-Form Networks

Product-Form Networks

- We are interested in $P(n_1, n_2, \ldots, n_k)$, i.e., the probability that there are n_1 customers in the first queue, n_2 in the second, etc.
- Consider simple linear network:
- Arrival rate for each queue is λ (why?)
- Utilization $\rho_i = \lambda/\mu_i$
- P(n_i jobs in i^{th} queue $= p_i(n_i) = (1-\rho_i)\rho_i^{n_i}$
- $P(n_1, n_2, \ldots, n_k) = p_1(n_1)p_2(n_2)\cdots p_k(n_k)$

Generalizing Product-Form Networks

- General form of equilibrium probability:

$$P(n_1, n_2, \ldots, n_k) = \frac{1}{G(N)} \prod_{i=1} kf_i(n_i)$$

- $G(N)$ is normalizing constant, function of total jobs in system
- $f_i(n_i)$ is function of (only) system parameters and n_i
- Not always true that each queue behaves as M/M/1
- ... But analysis of each queue is separable
- Surprisingly large classes of networks are product-form

Generalizing Product-Form Networks

- General form of equilibrium probability:

$$P(n_1, n_2, \ldots, n_k) = \frac{1}{G(N)} \prod_{i=1} kf_i(n_i)$$

- $G(N)$ is normalizing constant, function of total jobs in system
- $f_i(n_i)$ is function of (only) system parameters and n_i
- Not always true that each queue behaves as M/M/1
- ... But analysis of each queue is separable
- Surprisingly large classes of networks are product-form

Computer Systems as Queueing Networks

Three general types of queues appear in computer systems:

Fixed-capacity service center Service time doesn't depend on number of jobs; i.e., single server with queueing

Delay center Service time is random but no queueing; i.e. infinite number of servers (sometimes called *IS*)

Load-dependent service center Service rate depends on load; e.g., M/M/m with $m > 1$ (runs faster as more servers used)

Computer Systems as Queueing Networks

Three general types of queues appear in computer systems:

Fixed-capacity service center Service time doesn't depend on number of jobs; i.e., single server with queueing

Delay center Service time is random but no queueing; i.e. infinite number of servers (sometimes called *IS*)

Load-dependent service center Service rate depends on load; e.g., M/M/m with $m > 1$ (runs faster as more servers used)

Operational Quantities

- An *operational quantity* is something that can be observed
  - Necessarily over some period of time
  - If period is long enough, approximates a system parameter
- Examples:
  - Arrival rate $\lambda_i = \dfrac{\text{number of arrivals}}{\text{time}} = \dfrac{A_i}{T} \approx \lambda$
  - Throughput $X_i = \dfrac{\text{number of completions}}{\text{time}} = \dfrac{C_i}{T} \approx \lambda$
  - Utilization $U_i = \dfrac{\text{busy time}}{\text{total time}} = \dfrac{B_i}{T} \approx \rho$
  - Mean service time $S_i = \dfrac{\text{total time served}}{\text{number served}} = \dfrac{B_i}{C_i} \approx \mu$

Operational Quantities
- An *operational quantity* is something that can be observed
  - Necessarily over some period of time
  - If period is long enough, approximates a system parameter
- Examples:
  - Arrival rate $\lambda_i = \dfrac{\text{number of arrivals}}{\text{time}} = \dfrac{A_i}{T} \approx \lambda$
  - Throughput $X_i = \dfrac{\text{number of completions}}{\text{time}} = \dfrac{C_i}{T} \approx \lambda$
  - Utilization $U_i = \dfrac{\text{busy time}}{\text{total time}} = \dfrac{B_i}{T} \approx \rho$
  - Mean service time $S_i = \dfrac{\text{total time served}}{\text{number served}} = \dfrac{B_i}{C_i} \approx \mu$

Other Useful Quantities

- Number of devices M
- Visits per job V_i = Number of requests each job makes for device i (can be fractional)
- Demand D_i = Seconds of service needed from device i by each job = $V_i S_i$
- Overall system throughput $X = \frac{\text{jobs completed}}{\text{total time}} = \frac{C_0}{T}$
- Queue length at i: Q_i
- Response time at i: R_i
- Think time in interactive systems: Z

Other Useful Quantities

- Number of devices M
- Visits per job V_i = Number of requests each job makes for device i (can be fractional)
- Demand D_i = Seconds of service needed from device i by each job = $V_i S_i$
- Overall system throughput $X = \frac{\text{jobs completed}}{\text{total time}} = \frac{C_0}{T}$
- Queue length at i: Q_i
- Response time at i: R_i
- Think time in interactive systems: Z

Operational Laws

Utilization Law $U_i = \frac{B_i}{T} = \frac{C_i}{T} \times \frac{B_i}{C_i} = X_i S_i$

Forced Flow Law $X_i = X V_i$

- In other words, device i's throughput had better be V_i times the system throughput or it won't be able to handle the load

Little's Law $Q_i = X_i R_i$

General Response Time Law $R = \sum_{i=1}^{M} R_i V_i$

Interactive Response Time Law For N users, $R = (N/X) - Z$

- Not very profound, since R includes queueing effects: response time is round trip minus what you wasted on your own

Operational Laws

Utilization Law $U_i = \frac{B_i}{T} = \frac{C_i}{T} \times \frac{B_i}{C_i} = X_i S_i$

Forced Flow Law $X_i = X V_i$

- In other words, device i's throughput had better be V_i times the system throughput or it won't be able to handle the load

Little's Law $Q_i = X_i R_i$

General Response Time Law $R = \sum_{i=1}^{M} R_i V_i$

Interactive Response Time Law For N users, $R = (N/X) - Z$

- Not very profound, since R includes queueing effects: response time is round trip minus what you wasted on your own

Bottleneck Analysis

- Note that device demands D_i are total seconds of service needed from device i
- Some device (or devices) will be the max: $D_{\max}$
- This device is the *bottleneck device*
  - Improving other device performances can still improve response time, but most benefit will happen at bottleneck
- Asymptotic bounds on performance, as functions of N:

$$
\begin{aligned}
X(N) &\leq \min\left\{\frac{1}{D_{\max}}, \frac{N}{D+Z}\right\} \\
R(N) &\geq \max\left\{D, ND_{\max} - Z\right\}
\end{aligned}
$$

where $D = \sum D_i$

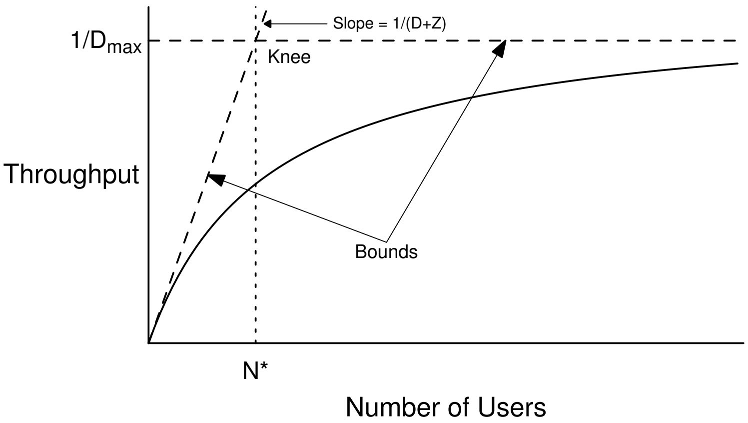
Bottleneck Analysis

- Note that device demands D_i are total seconds of service needed from device i
- Some device (or devices) will be the max: $D_{\max}$
- This device is the *bottleneck device*
  - Improving other device performances can still improve response time, but most benefit will happen at bottleneck
- Asymptotic bounds on performance, as functions of N:

$$
\begin{aligned}
X(N) &\leq \min\left\{\frac{1}{D_{\max}}, \frac{N}{D+Z}\right\} \\
R(N) &\geq \max\left\{D, ND_{\max} - Z\right\}
\end{aligned}
$$

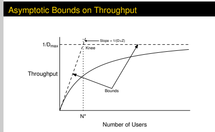
where $D = \sum D_i$

Asymptotic Bounds on Throughput

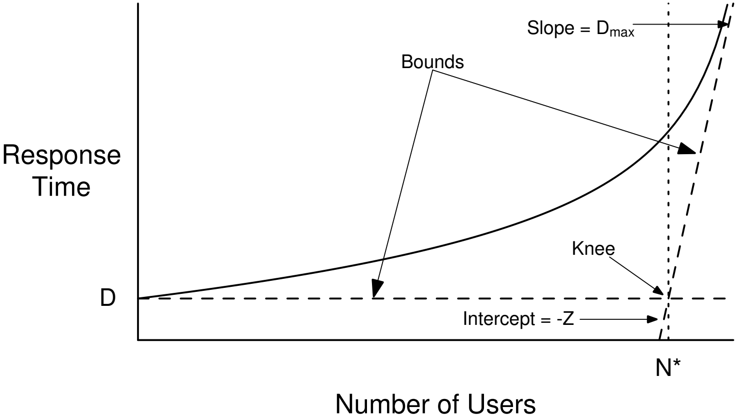


2015-06-15

CS147
└Queues in Computer Systems
  └Bottleneck Analysis
    └Asymptotic Bounds on Throughput

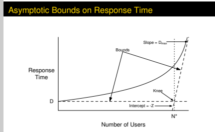


Asymptotic Bounds on Response Time



2015-06-15

CS147
└Queues in Computer Systems
 └Bottleneck Analysis
  └Asymptotic Bounds on Response Time



Mean Value Analysis

- Iterative procedure for calculating per-device parameters (response time, queue length, etc.)
- Basic approach:
  - Assume queue length = 0 for all devices
  - For increasing user counts, calculate response times, then new queue lengths
- Complexity is $O(MN)$ for M devices, N maximum users
  - Approximations exist for reducing complexity

Mean Value Analysis

- Iterative procedure for calculating per-device parameters (response time, queue length, etc.)
- Basic approach:
  - Assume queue length = 0 for all devices
  - For increasing user counts, calculate response times, then new queue lengths
- Complexity is $O(MN)$ for M devices, N maximum users
  - Approximations exist for reducing complexity

Hierarchical Decomposition

- Large networks are hard to deal with
- Stems comes to the rescue!
  - In a queueing network, a complex subsystem with one input and one output can be replaced by a single queue tuned to the same behavior
  - In particular, if you're interested in device i, the *entire rest of the network* has just one input and output
- Techniques are similar to things used in Stems
- Advantage: easy to study lots of settings for one device

Hierarchical Decomposition

- Large networks are hard to deal with
- Stems comes to the rescue!
  - In a queueing network, a complex subsystem with one input and one output can be replaced by a single queue tuned to the same behavior
  - In particular, if you're interested in device i, the *entire rest of the network* has just one input and output
- Techniques are similar to things used in Stems
- Advantage: easy to study lots of settings for one device

Studying One Device

1. Pick a device to study (also works for subnetwork)
2. Set device's service times to zero, solve remaining network
3. Replace remaining network with single load-dependent queue, using solved parameters
4. Reset device's service time and solve result

Studying One Device

1. Pick a device to study (also works for subnetwork)
2. Set device's service times to zero, solve remaining network
3. Replace remaining network with single load-dependent queue, using solved parameters
4. Reset device's service time and solve result

Limitations of Queueing Theory

Queueing theory is useful but has limitations:

- Nonexponential service times
- Self-similar (“train”) arrivals
- Load-dependent arrivals
- Response-dependent arrivals (e.g., retransmissions)
- Defections after joining queue
- Transient analysis generally not possible
- Fork and join make jobs interdependent
- Contention for resources
- Holding multiple resources
- Mutual exclusion among jobs
- Blocking of other devices

Limitations of Queueing Theory

Queueing theory is useful but has limitations:
- Nonexponential service times
- Self-similar (“train”) arrivals
- Load-dependent arrivals
- Response-dependent arrivals (e.g., retransmissions)
- Defections after joining queue
- Transient analysis generally not possible
- Fork and join make jobs interdependent
- Contention for resources
- Holding multiple resources
- Mutual exclusion among jobs
- Blocking of other devices