YORK UNIVERSITÉ UNIVERSITY

ASRL
Adaptive Systems Research Lab

Model Based Adaptive Systems

Marin Litoiu
Department of Electrical Engineering and Computer Science
York University
mlitoiu@yorku.ca
http://www.ceraslabs.com

PID Controller

- **The PID algorithm is the most popular feedback controller algorithm used**
- **It is a robust easily understood algorithm that can provide excellent control performance despite the varied dynamic characteristics of processes**
- **PID algorithm consists of three basic modes:**
  - **P**roportional mode
  - **I**ntegral mode
  - **D**erivative mode

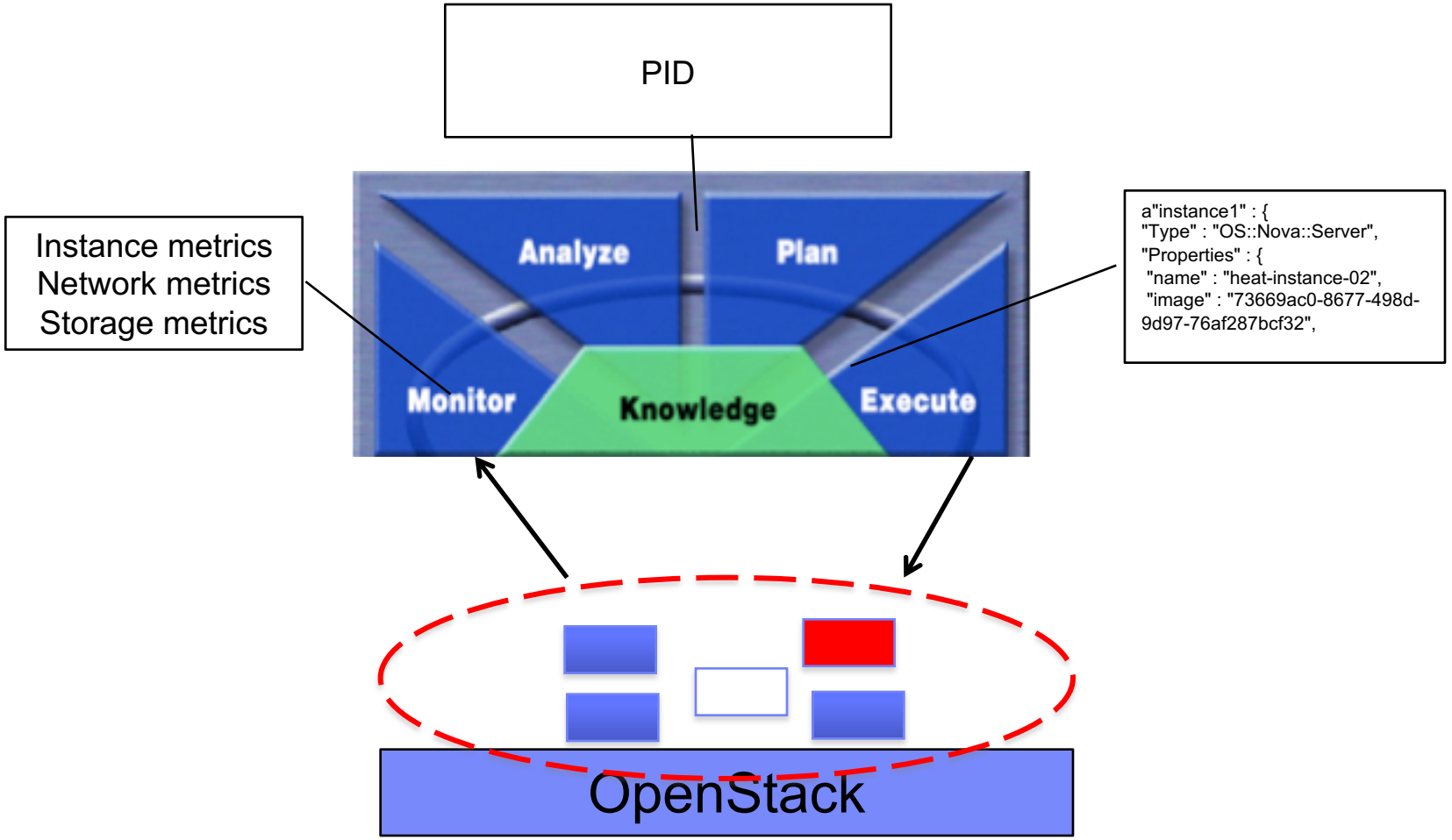
P, PI, or PID Controller

- **When utilizing the PID algorithm, it is necessary to decide which modes are to be used (P, I or D) and then specify the parameters (or settings) for each mode used.**
- **Generally, only three basic algorithms are used: P, PI or PID**

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

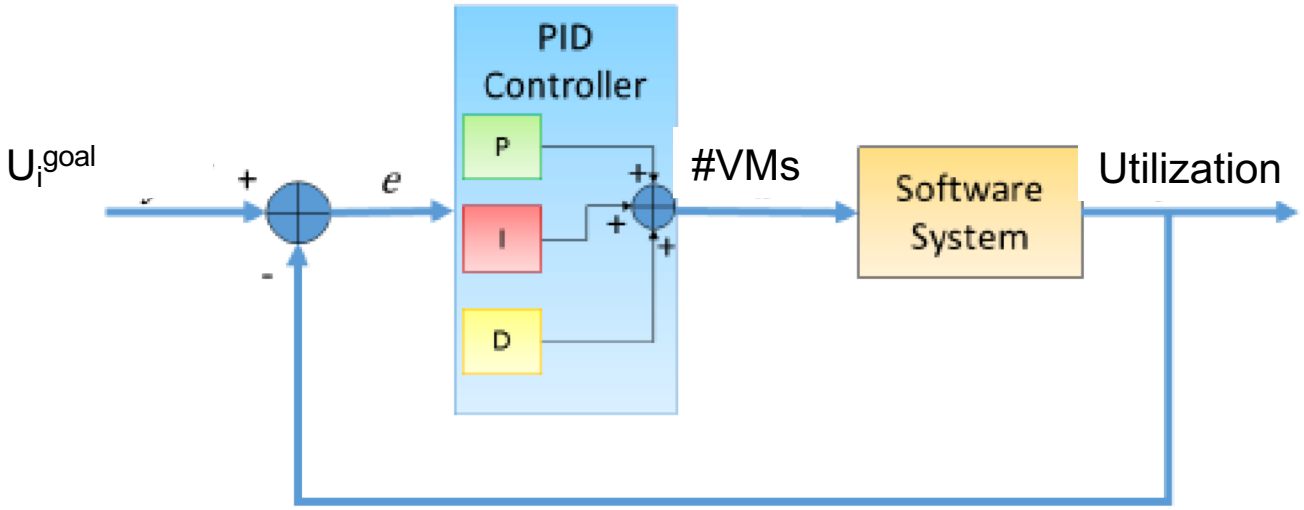
proportional gain integral gain derivative gain

Case Study*

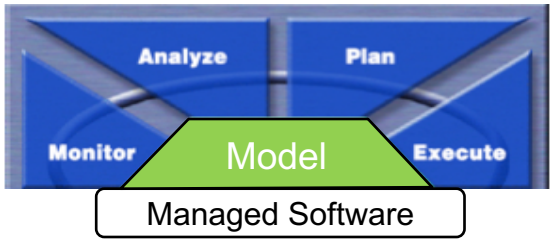


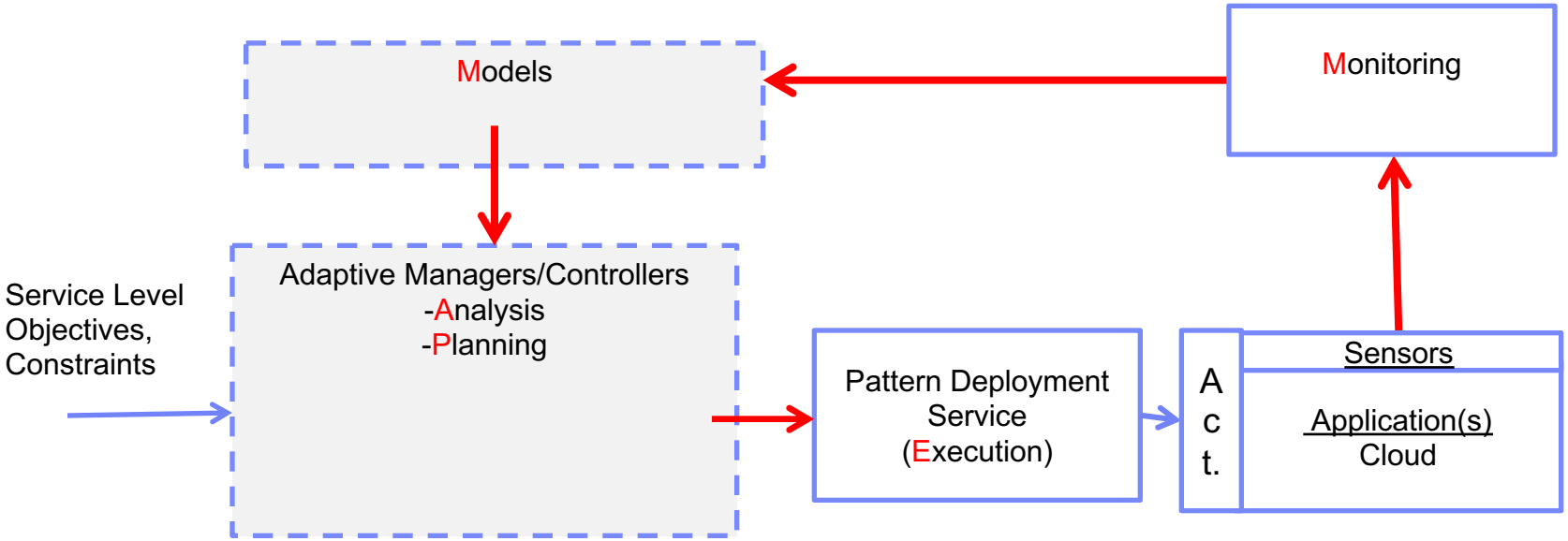
C. Barna, M. Fokaefs, M. Litoiu, M. Shtern and J. Wigglesworth, "Cloud Adaptation with Control Theory in Industrial Clouds," *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, Berlin, 2016, pp. 231-238.

PID Controller for Utilization

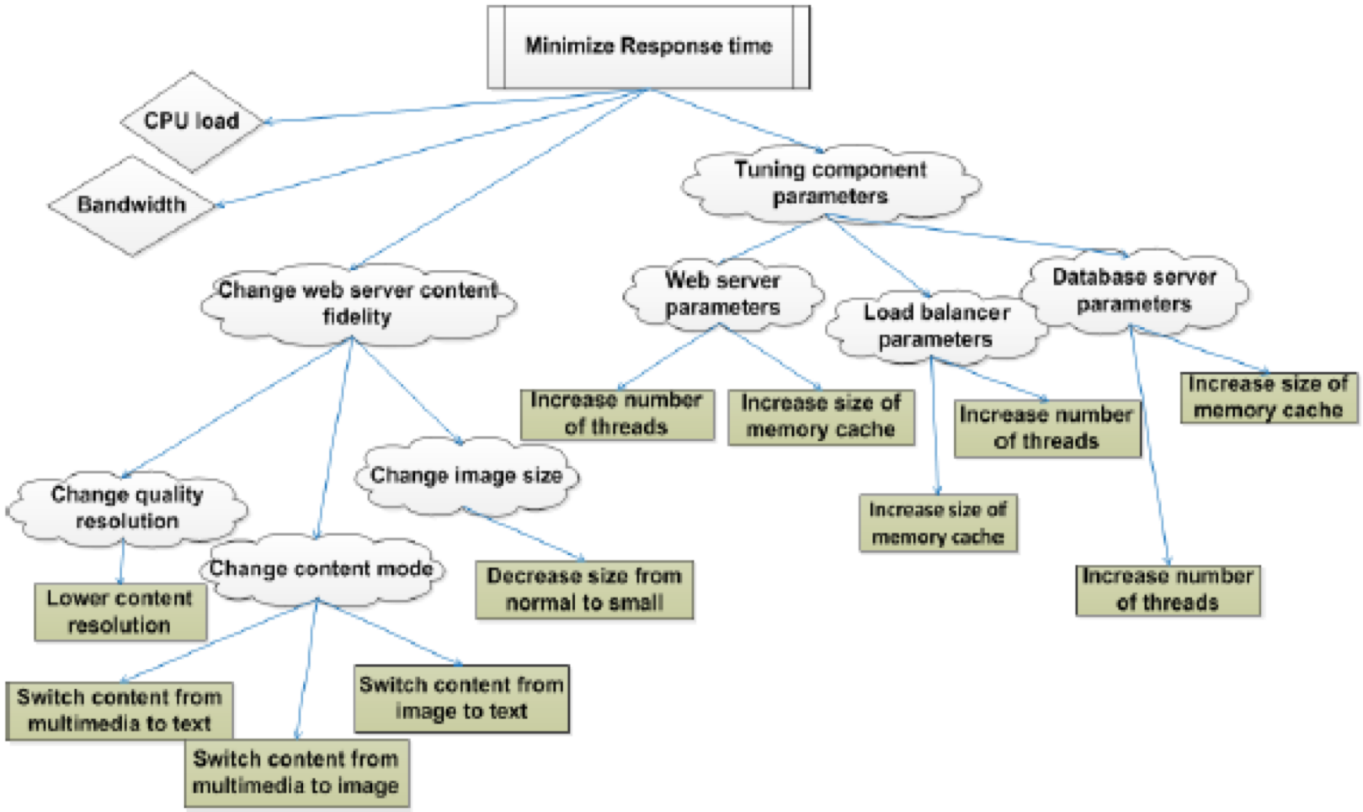


Model-based Adaptive Software





What can I change at runtime to affect performance?



The Role of Models

- **Y= F(U, P, X)**
  - Y outputs: response time, etc
  - P disturbances: number of users, arrival rate, etc..
  - X states: utilization, queue length, etc
- **F a (non-linear) function**
- **An autonomic manager, periodically**
  - Will find U that gives the desired Y, given the P and X
  - Will execute a plan that implements U

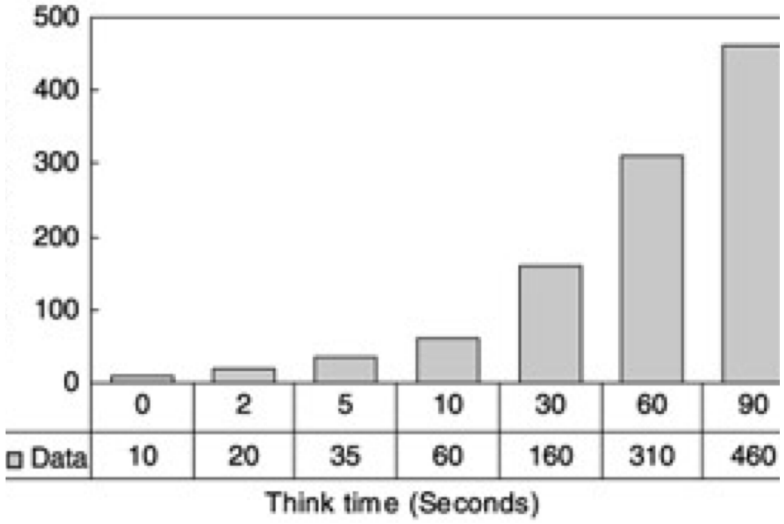
Models

- **Queuing Network Models**
- **Control Theoretic Models**
- **Machine Learning Models**
  - Regression models
  - Neural networks models

Performance

- **Performance Modeling**
  - Fundamental Laws
  - Open Models
  - Closed Models

User interaction: think time

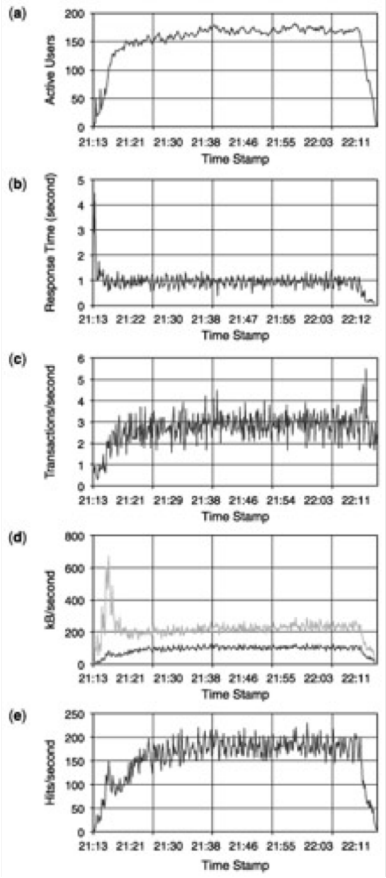


$$N_{concurrent} = N_{active}\left(\frac{Response\ time}{Response\ time + Think\ time}\right)$$

- **Think time: is the time the user "thinks" between two requests**
- **Think time adds to the response time and makes the execution of a use case longer**
- **Not all users have the same "think time" and the think time depends on the task:**
  - You think longer when "adding to the shop-cart" then when "browsing"
- **On the left, the same application supports many type of users with different "think times"**
  - There are 460 users with think time=90s
  - There are 20 users with think time=2s
- **Concurrent users: those who stress the system at the same time**
- **Active users: concurrent users + those who "think"**
- **Concurrent users are a fraction of the active users (see the formula on the left)**

Typical performance testing graphs

- **A number of active users are simulated(that is, they are programs, not people:-)**
- **Response time depends on the number of users**
  - The response time is higher at the beginning of the test
- **Throughput, the number of transactions (responses) per second is a function of the number of users**
- **Data transfer rate (HTTP request and reply)**
- **Hit rate (requests + responses)**



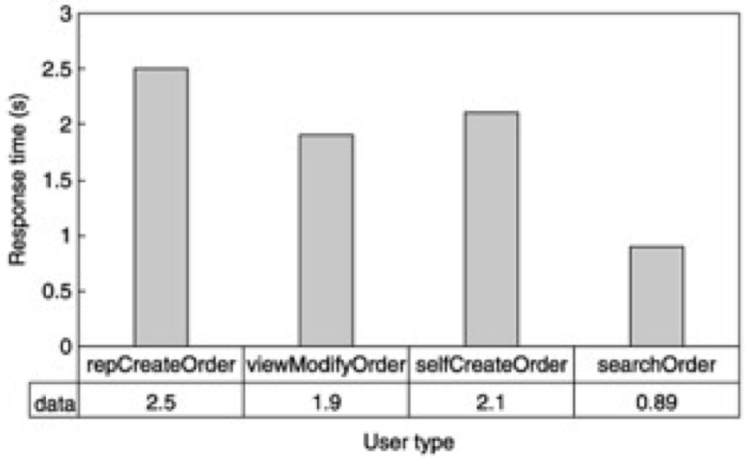
Performance is stochastic

- **If you repeat two performance experiments in identical conditions, you get different performance results**
- **To characterize a metric, use : mean, min, max, standard deviation (see right)**
- **Percentile is better for user perception**

Artifact	Measured Data for Each Attribute
Number of active users	182
Overall response time	Minimum: 0 s
	Average: 0.94 s
	Maximum: 28 s
	Standard deviation: 2.9 s
Transactions	Total: 10,344
	Average rate: 2.7/second
Data transfer	Total: 1,175 MB
• (request + response)	Transfer rate: 307 kB/s
Web server hits	Total: 600,064
	Average rate: 157/second

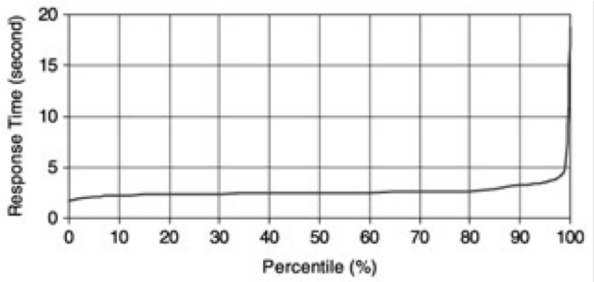
More drilling down

- **Performance is typical per user group**
  - Different use cases (functions) of the application have different performance requirements
  - On the right, you have 4 functions, each one has a different average response time



Percentile

- **Study shows that the user remembers the "worst case"**
- **To increase user satisfaction, do not make the mean(average) response time small**
- **Instead, improve the percentile, the percent of responses under a specific value**
  - 90 percentile, means that 90% of response times are under 3.3 second



User Group	Average (s)	90th Percentile (s)
repCreateOrder	2.5	3.3
viewModifyOrder	1.9	2.4
selfCreateOrder	2.1	2.7
searchOrder	0.9	1.1

Measurements

- **Workload generators: tools that generate synthetic workloads, any big company has one (IBM, Oracle, HP, CA, JMeter)**
  - Simulate the number of users and think times for each class of users
- **Measurements: use OS performance tools**
  - Windows: perfmon
  - SNMP, Ceilemoter (Open Stack), CloudWatch(Amazon)
- **How do we measure the utilization?**
  - Operating systems have performance monitors that show the CPU, disk and network utilization
  - In general, each monitoring tool (OS, Platform, Application) provides utilization, make sure you read the documentation of each monitor

Performance Modeling

- Queuing theory
- Little's Law
- Queing Networks

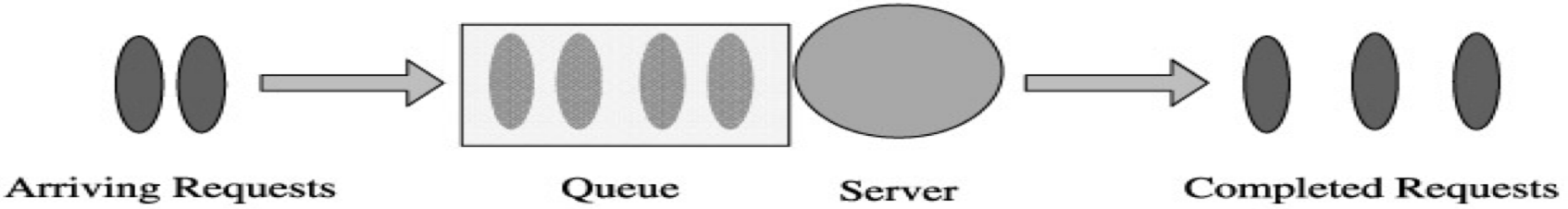
Queuing – Basic concepts

- **Queuing is not developed specifically for software performance**
  - Manufacturing
  - Customer facing offices
  - Networks
  - ..wherever there is line of things, people..
- **It helps you better understand performance**
  - Find the cause of a performance problem
  - Find the solution to a performance problem

Concepts and terminology

- **Think that you visit your local bank office and have to wait in line. These are the terms of the queuing theory**
  - *Server.* This is bank office fulfilling the customer's service requests.
  - *Customer*. This is the initiator of service requests (you and others in line).
  - *Wait Time*. This is the time duration a customer has to spend waiting in line.
  - *Service Time*. This is the time duration from when a teller starts to service a customer to when the customer is leaving the teller and the next customer is called in for service.
  - *Arrival Rate*. This is the rate at which customers arrive for service.
  - *Service Rate*. This is the rate at which customers are serviced.
  - *Utilization.* This is the portion of a teller's time actually servicing customers rather than idling.
  - *Queue Length.* This is the total number of customers waiting or being serviced or both.
  - *Response Time*. This is the sum of wait time and service time for one visit to a teller.
  - *Residence Time.* This is the total response time if the teller is visited multiple times for one transaction.
  - *Throughput.* This is the rate at which customers are serviced. A banking center certainly is interested in knowing how fast it can service customers without losing them because of long wait times.
- **Typical Question: For a given average arrival rate λ and a given service rate μ, how many parallel servers (m) are required in order for the office to operate under steady-state conditions(not overflow)?**

Notations



Symbol	Semantics
S	Service time
V	Number of visits to the server
*D=V*S*	Service demand
R	Response time
R'	Residence time
X	Throughput
λ	Arrival rate
U	Utilization
W	Wait time
N	Total queue length (waiting and/or being serviced)

Examples of Queues
-CPU
-DISK
-Network card
-a critical section
-a semaphore
-a threading pool

Little's Law

- **$N_i=X_i*R_i$; created by John Little (1961)**
  - where i is the queue; R_i is the response time; X_i is the throughput of queue;
- **Since $R_i=(W_i+S_i)$, waiting time plus service time**
- **Then $N_i= X_i*W_i+ X_i*S_i$**
- **$N_{i\_wait}= X_i*W_i$; requests waiting**
- **$N_{i\_service}= X_i*S_i$; requests being served**
- **Quiz:**
  - If the customers arrive 10 per hour and leave the bank office after 0.5 hours then there how many customers are in the bank office?
    - N=?
  - Further, if the service time per customer is 0.2 hours, how many customers are served at one time? How many are waiting
  - $N_{i\_service}$=? N_{i_wait}=?

Fundamental Laws: Utilization Law

- **Relates the utilization, mean throughput (X) and service time or demand (D)**
- **$U_i=X_i*D_i$; where i is the queue; D_i is the Demand(service time to complete a request at queue i); X_i is the throughput of queue i;**
  - Proof: if B is the total time the server i is busy over time T, and N is the number of customers, then U=B/T=(B/N)*(N/T)=D*X;.
- **Notes**
  - Utilization law applies to any subsystem (hardware and software)
  - It is assumed that the arrivals equals the completions(no customers/requests are lost inside the system)
  - No assumptions made on the arrival or service time distributions
- **We can write the utilization for each class of customers c: If Xc is the throughput of customers in class c and Dk,c is the demand of customers c at queue k, then**
  - Uk,c=Xc* Dk,c
- **Then the total utilization of queue k is the sum of per class utilizations**

$$Uk = \sum_{c=1..C} Uk, c$$

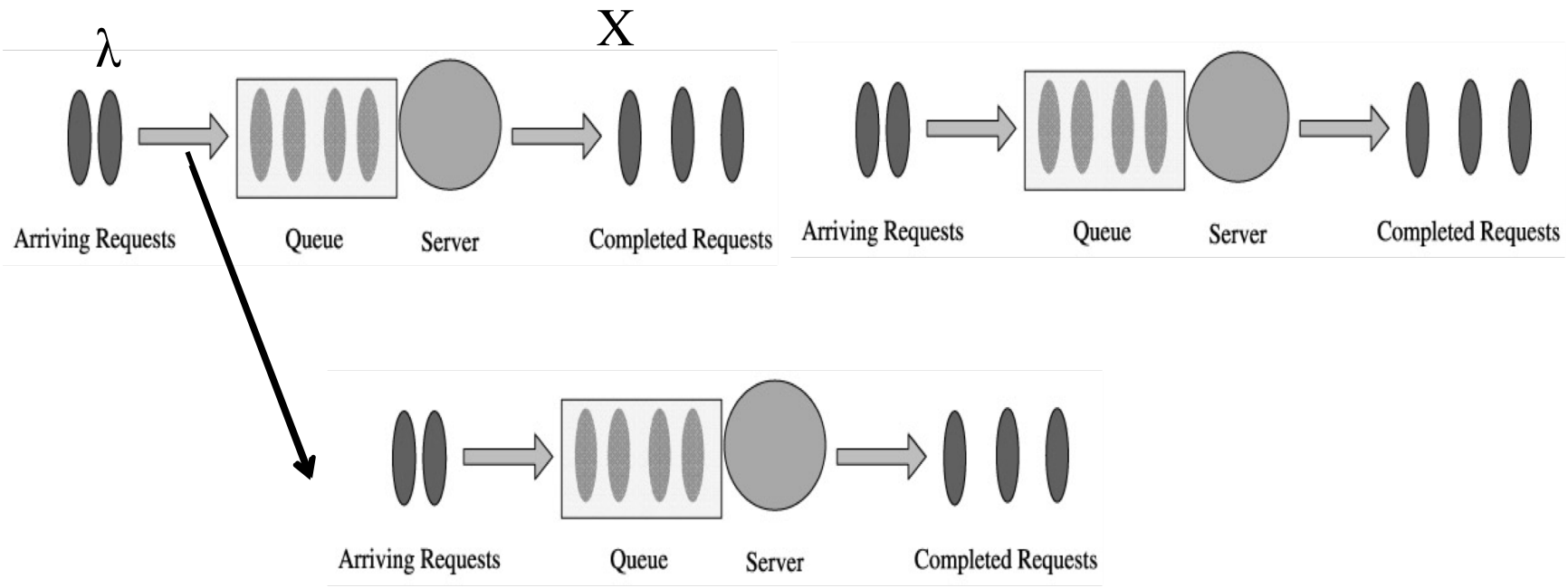
Utilization Analysis

- **We can use the utilization law to check if the resources exceeds (or a close to) their capacity.**
  - Reminder: Utilization cannot exceed 100%. Rule of thumb: Utilization should be well below 100%, for example, it should be less than 60%.
  - If we know the throughput we want to support in the system and the demand Dk at each resource k, then we can compute the utilization of each resource
  - If the utilization is too high, then we either reduce the throughput or make Dk smaller
- **Consider a 2-tier application, with a database and web server. The queuing centres are the DISKs and CPUs at each server.**
  - The demands are:
    - Dweb-cpu = 10ms, Dweb-disk=8ms;
    - Ddb-cpu = 12ms, Dweb-disk=4ms;
  - The application needs to support a throughput of 100 transactions/seconds (the throughput is the same at each device, that is called *the forced flow law*) .
  - What is the utilization of each device?
  - What is the maximum throughput we can support?
- **Utilization can be applied to software components as well**
  - Assume the webserver container (that runs the servlets) has a Response time of R=1second
  - Its utilization is Us=Rs*X=100.
  - Note that the utilization is greater than 1, that is because the container runs threads, therefore a container can be multiplied as needed
  - The meaning is that there are 100 threads running on average

Your turn

- **Assuming you have a running system described by the simple models**
  - Little formula
  - Utilization law
- **How would you use that to create autonomic system**
- **Y=?**
- **U=?**
- **What would be the effectors (those parameters, variables, etc..) you can change?**

Queues: there is a network of queues

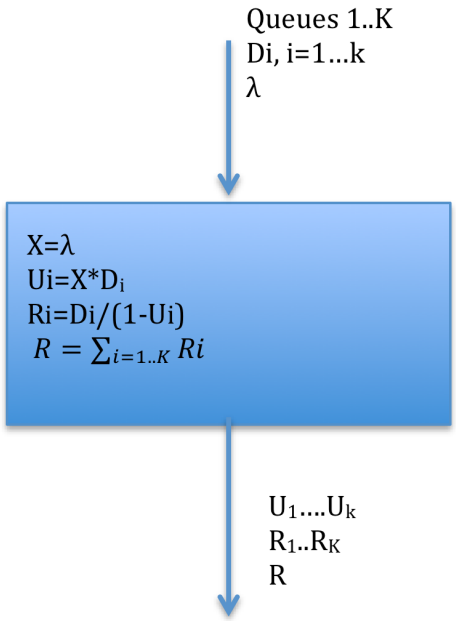


Queuing Network Models

- **Approximations of the real system**
  - Approximate the user interaction, the structure and the behavior of the system
  - Information about the sequence of operations is not included in the model
  - The system is a set of queues and customers' requests flow from queue to queue with fixed probabilities
- **Open Queuing Network Models**
  - Consider the system has an infinite number of customers that arrive with a specific rate, usually called lambda (measured in requests per second)
  - Used for large systems where the response time does not affect the arrival rate (requests per second)
- **Closed Models**
  - Consider the system has N customers and each customer has a think time Z
  - Used for system in which the arrival rate depends on the response time
- **Both closed and open models can distinguish between classes of customers; typically a class is specified by an index *c***
  - In the simplest case, a "class" is a use case(scenario)

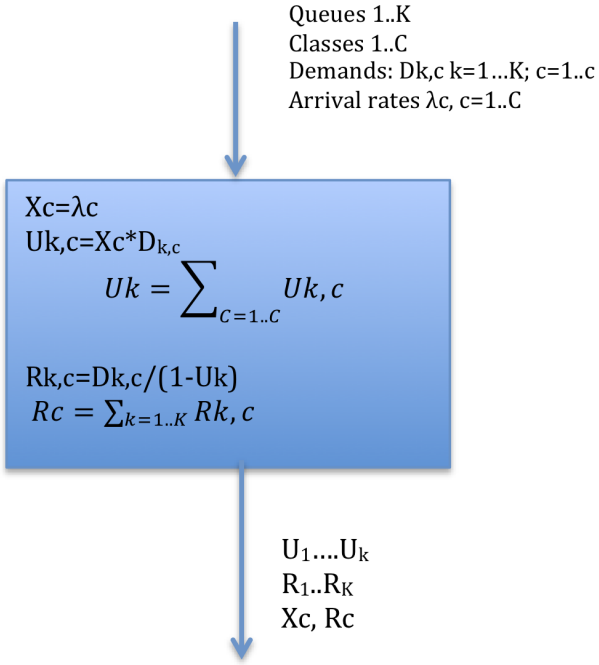
Open Models

- Very simple and fast formulas
- Inputs: Queues and demands in queues; arrival rate (req/sec)
- Outputs: overall response time; throughput and per server utilization
- Note that Xi=X (forced flow law): the customers requests go through all queues and are not lost.



Open Models for Multiple Classes

- **Very simple and fast formulas**
- **Inputs: Queues and demands in queues; arrival rate (req/sec) for each class of request**
- **Outputs: overall response time per each class; throughput and per server utilization per each class of request**
- **You can implement this in Excel**
  - Keep in mind U<1, therefore, (1-Uk) should not be 0 or negative



Data for a a 3 tier app, 2 classes, 2 queuing centres (application server and database server)

lambda_1	lambda_2	D11	D12	D21	D22	U11	U12	U21	U22	U1	U2	R11	R12	R21	R22	R1	R2
0.15	0.1	0.1	0.15	0.2	0.25	0.015	0.015	0.03	0.025	0.03	0.055	0.103092784	0.154639175	0.211640212	0.264550265	0.314732995	0.41918944
0.3	0.2	0.1	0.15	0.2	0.25	0.03	0.03	0.06	0.05	0.06	0.11	0.106382979	0.159574468	0.224719101	0.280898876	0.33110208	0.440473344
0.45	0.3	0.1	0.15	0.2	0.25	0.045	0.045	0.09	0.075	0.09	0.165	0.10989011	0.164835165	0.239520958	0.299401198	0.349411068	0.464236362
0.6	0.4	0.1	0.15	0.2	0.25	0.06	0.06	0.12	0.1	0.12	0.22	0.113636364	0.170454545	0.256410256	0.320512821	0.37004662	0.490967366
0.75	0.5	0.1	0.15	0.2	0.25	0.075	0.075	0.15	0.125	0.15	0.275	0.117647059	0.176470588	0.275862069	0.344827586	0.393509128	0.521298174
0.9	0.6	0.1	0.15	0.2	0.25	0.09	0.09	0.18	0.15	0.18	0.33	0.12195122	0.182926829	0.298507463	0.373134328	0.420458682	0.556061158
1.05	0.7	0.1	0.15	0.2	0.25	0.105	0.105	0.21	0.175	0.21	0.385	0.126582278	0.189873418	0.325203252	0.406504065	0.451785531	0.596377483
1.2	0.8	0.1	0.15	0.2	0.25	0.12	0.12	0.24	0.2	0.24	0.44	0.131578947	0.197368421	0.357142857	0.446428571	0.488721805	0.643796992
1.35	0.9	0.1	0.15	0.2	0.25	0.135	0.135	0.27	0.225	0.27	0.495	0.136986301	0.205479452	0.396039604	0.495049505	0.533025905	0.700528957
1.5	1	0.1	0.15	0.2	0.25	0.15	0.15	0.3	0.25	0.3	0.55	0.142857143	0.214285714	0.444444444	0.555555556	0.587301587	0.76984127
1.65	1.1	0.1	0.15	0.2	0.25	0.165	0.165	0.33	0.275	0.33	0.605	0.149253731	0.223880597	0.506329114	0.632911392	0.655582845	0.856791989
1.8	1.2	0.1	0.15	0.2	0.25	0.18	0.18	0.36	0.3	0.36	0.66	0.15625	0.234375	0.588235294	0.735294118	0.744485294	0.969669118
1.95	1.3	0.1	0.15	0.2	0.25	0.195	0.195	0.39	0.325	0.39	0.715	0.163934426	0.245901639	0.701754386	0.877192982	0.865688812	1.123094622
2.1	1.4	0.1	0.15	0.2	0.25	0.21	0.21	0.42	0.35	0.42	0.77	0.172413793	0.25862069	0.869565217	1.086956522	1.04197901	1.345577211
2.25	1.5	0.1	0.15	0.2	0.25	0.225	0.225	0.45	0.375	0.45	0.825	0.181818182	0.272727273	1.142857143	1.428571429	1.324675325	1.701298701
2.4	1.6	0.1	0.15	0.2	0.25	0.24	0.24	0.48	0.4	0.48	0.88	0.192307692	0.288461538	1.666666667	2.083333333	1.858974359	2.371794872
2.55	1.7	0.1	0.15	0.2	0.25	0.255	0.255	0.51	0.425	0.51	0.935	0.204081633	0.306122449	3.076923077	3.846153846	3.28100471	4.152276295
2.7	1.8	0.1	0.15	0.2	0.25	0.27	0.27	0.54	0.45	0.54	0.99	0.217391304	0.326086957	20	25	20.2173913	25.32608696

Q: what device is the bottleneck? What is the max arrival rate you can accommodate above?

Exercise: Extend the model to 3 classes: use demands and arrival rates to be half of class 1. What do you notice with regard to Utilization and Response time?

Summary

- **Performance metrics**
  - Utilization, response time, throughput
- **Scalability depends on how fast ONE transaction is, that is depends on D (demand); minimizing D makes an application more scalable;**