So far, we have…

Thursday, February 18, 2010
2:38 PM

So far, we have…

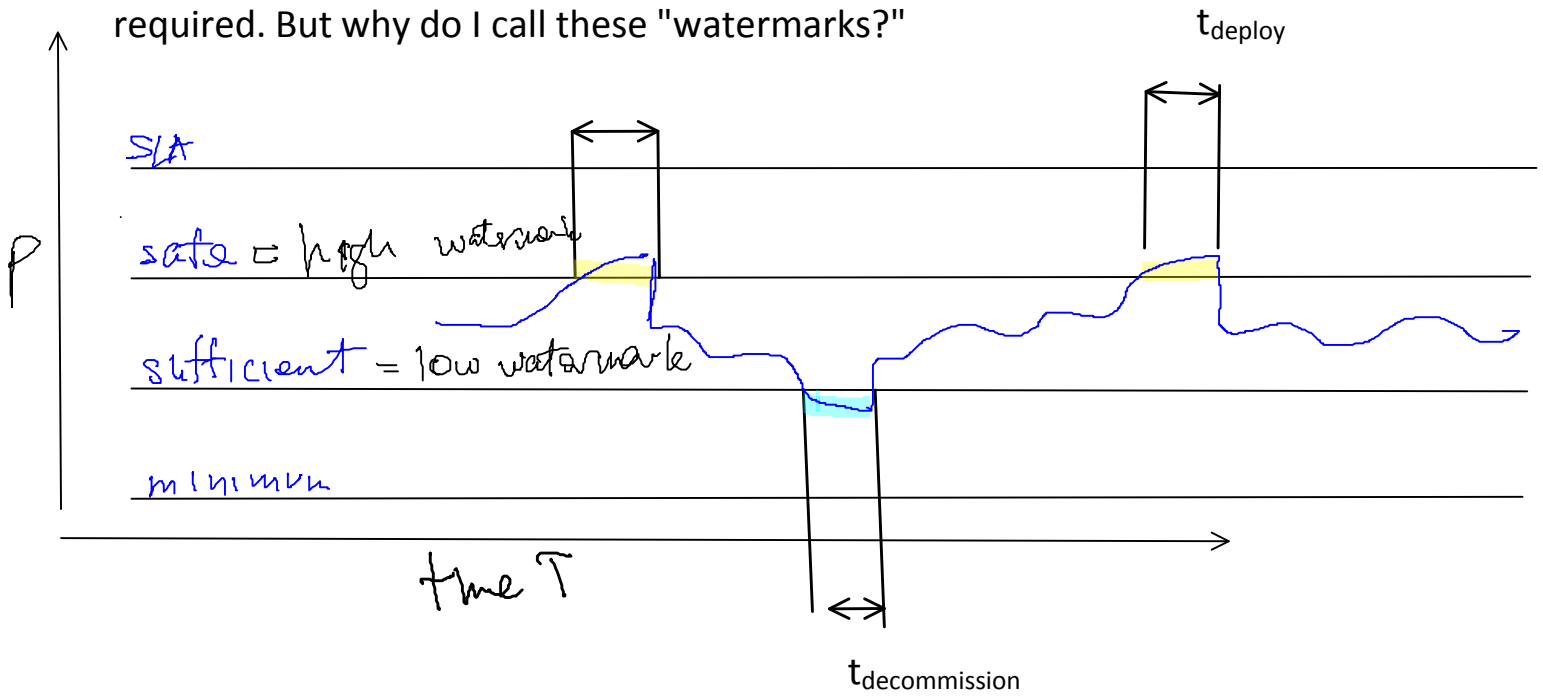
- Characterized the behavior of the cloud in the average case.
- Characterized what it means to comply with an SLA "on average" or not.
- Reasoned in terms of end-to-end request time (client to cloud and back).

Where we are

Thursday, February 18, 2010
8:18 AM

Where we are

P_{safe},t_{safe} are **high watermarks**: going higher is risky.
$P_{sufficient}$,$t_{sufficient}$ are **low watermarks**: going lower isn't required. But why do I call these "watermarks?"



The picture of execution so far

Thursday, February 18, 2010
8:37 AM

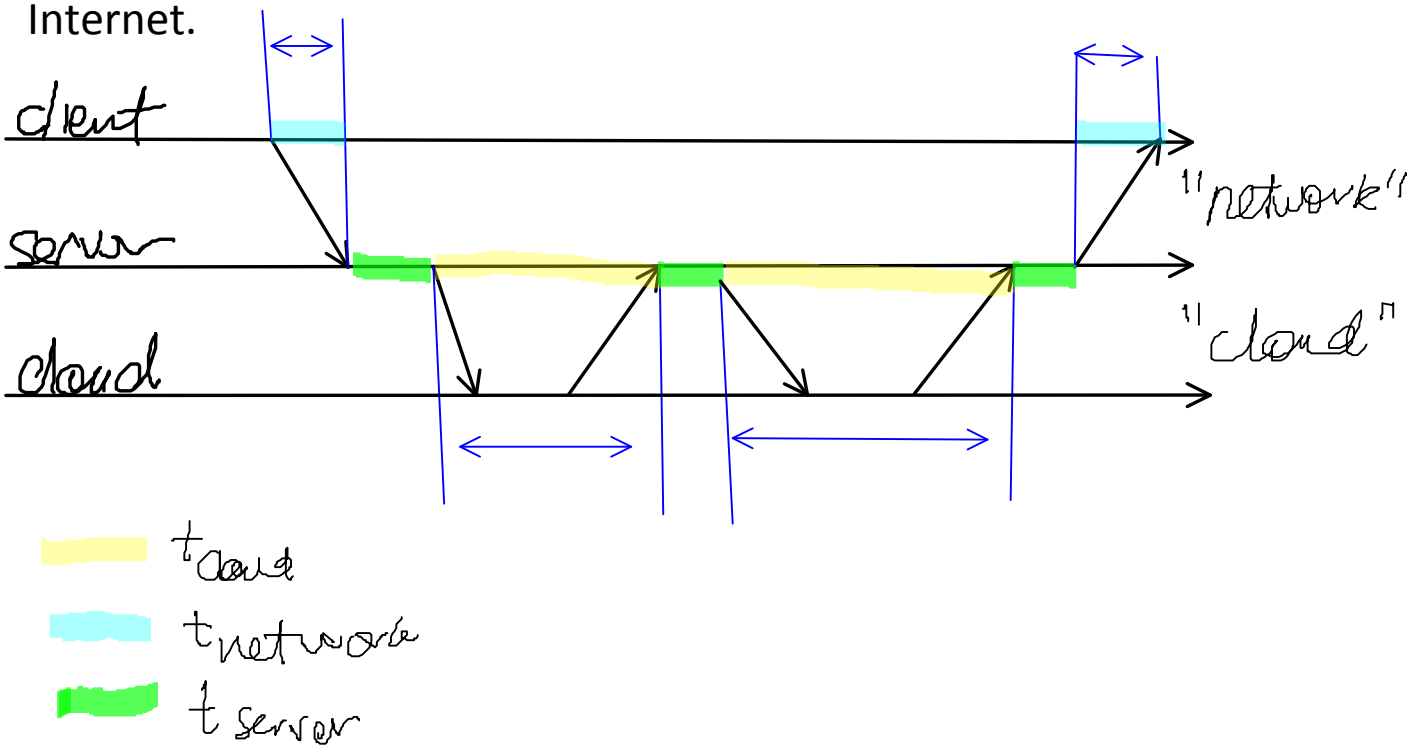
The picture of execution so far

$t_{request}=t_{server}+t_{cloud}+t_{network}$.

"You control" t_{server} = time spent in your application.

"The cloud controls" t_{cloud} = time spent waiting for cloud services.

"The network controls" $t_{network}$ = time spent waiting for requests and responses to be sent over the Internet.



Thursday, February 18, 2010
9:33 AM

Commodity computing

One reason this material is difficult at first glance: a new **world view.**

Computing cycles are a **commodity** (like electricity, water, or sewer).

If you come up short, you **add cycles**.

If you have too much, you **remove cycles**.

Cycles cost money, so **you never want to have more than you need.**

Thursday, February 18, 2010
9:36 AM

Capacity planning

If computing cycles are a commodity, then one must think about demand and capacity

Demand is what cycles you need (not controlled; varies with time)

Capacity is your ability to provide cycles (you control).

A power analogy

Thursday, February 18, 2010
9:41 AM

Power analogy

- If demand increases (air conditioning in summer), run more generators.
- If demand decreases (fall, spring: no heat or air conditioning), shut down generators.

The (typical) SLA for home power:

- 120VAC±10 VAC @ 60 cycles/sec, 100 Amps.

This analogy is more accurate than you might think!

In this episode…

Thursday, February 18, 2010
11:18 AM

In this episode….

Change units to make them easier to measure and observe, and more intuitive to control.

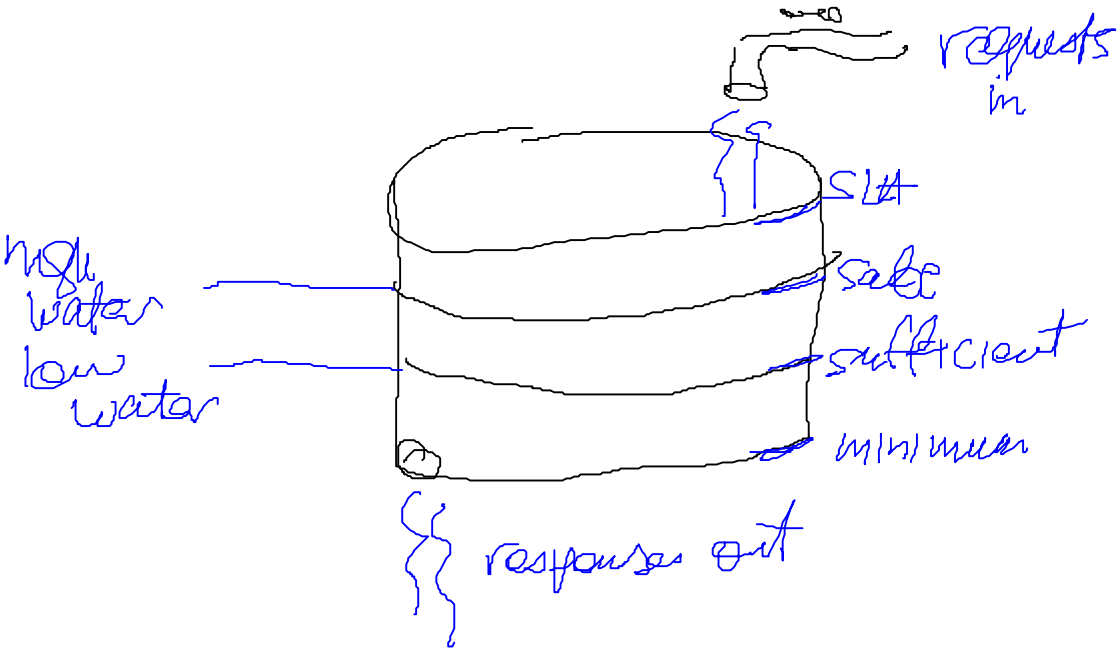
"From $t_{request}$ to load average L"

The watermark analogy

Thursday, February 18, 2010
9:05 AM

The watermark analogy

Think of pending requests as water in a basin.
Requests flow in by being made.
Requests flow out by being serviced.
The top lip of the basin is the SLA.
Key is to keep the basin at a "comfortable state": neither empty nor full.
The fullness of the basin is called **load**!

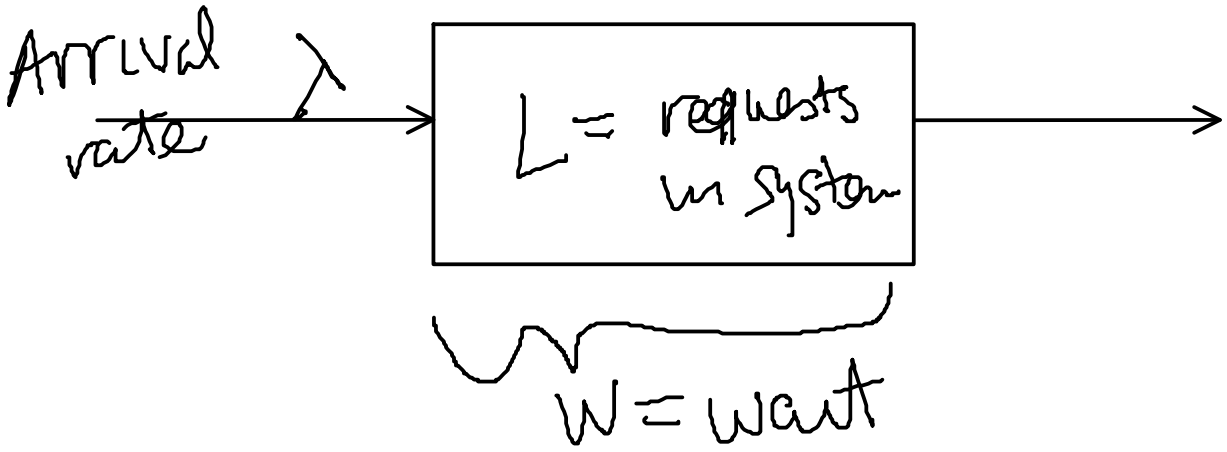


A basic service model

Requests flow into the system at some **arrival rate** λ, e.g., "5 requests per second."

At any time, there are L **requests in system**, awaiting responses.

At any time, there is an **average time in system** W between receiving a request and servicing it.



Thursday, February 18, 2010
2:51 PM

Steady-state behavior

A request system is said to be in steady state if

- Request arrival rate is constant.
- Response rate = request arrival rate.
- Time-in-system for a request is constant.

This implies that L is constant too!

Big issue: if a system is big enough, i.e., has enough clients, then it approaches steady state through scale, and not through actions of any one service or client.

(End of 2/14/2011 lecture)

A summary of this lecture

Wednesday, February 16, 2011
4:03 PM

Little's law

t system

Requests pending		Request arrival rate		Processing time
$L_{request}$	=	$\lambda_{request}$	*	$W_{request}$

determined by structure of the app

L_{CPU}	=	λ_{CPU}	*	W_{CPU}
Processes waiting		Process phase arrival rate		Phase processing time

"load average"

Where

All values are average, system in steady state.

A $\propto$ B means $A = \alpha B$, for some constant α

We don't know α.

We know L_{CPU} = load average

We know $\lambda_{request}$ = request arrival rate.

Punch line

Wednesday, February 16, 2011
4:36 PM

In last exercise, I asked you to compute $t_{sufficient}$

t_{safe} = response time at which one should add resources.

$t_{sufficient}$ = response time at which one should remove resources.

t_{safe} arises from SLA.

$t_{sufficient}$ arises from load average.

Little's Law

Thursday, February 18, 2010
9:13 AM

Little's Law

A system is in **steady state** if the average number of requests leaving the system = the average number of requests coming in.

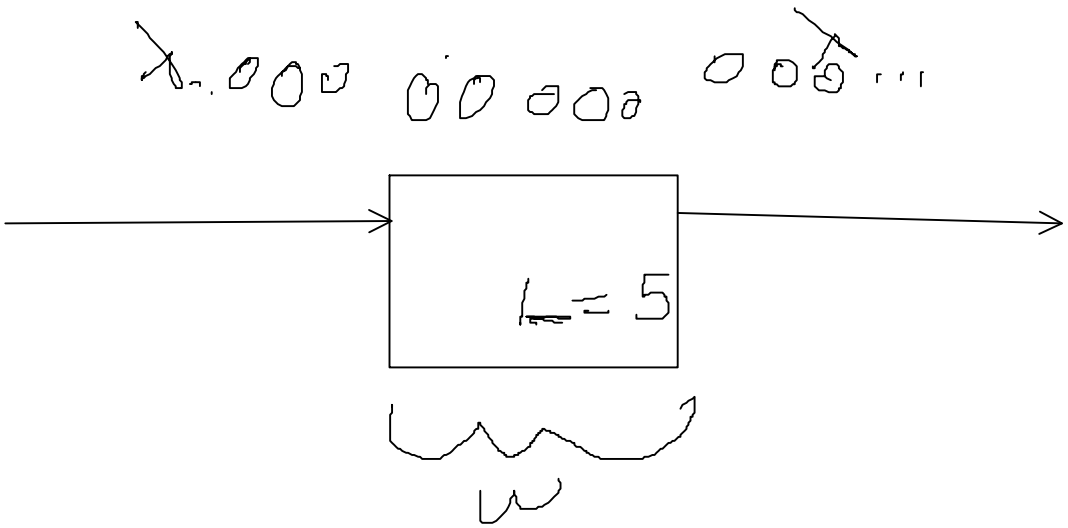
Little's law relates

The **average arrival rate** λ (e.g**.,** requests arriving per second) (average inter-arrival time is 1/λ)

The **average waiting time in system W** (e.g., seconds between request and response)

The **average requests in system L** (e.g., the number of requests received and awaiting service)

If the system is in steady state, then L = λW.



An easy analogy

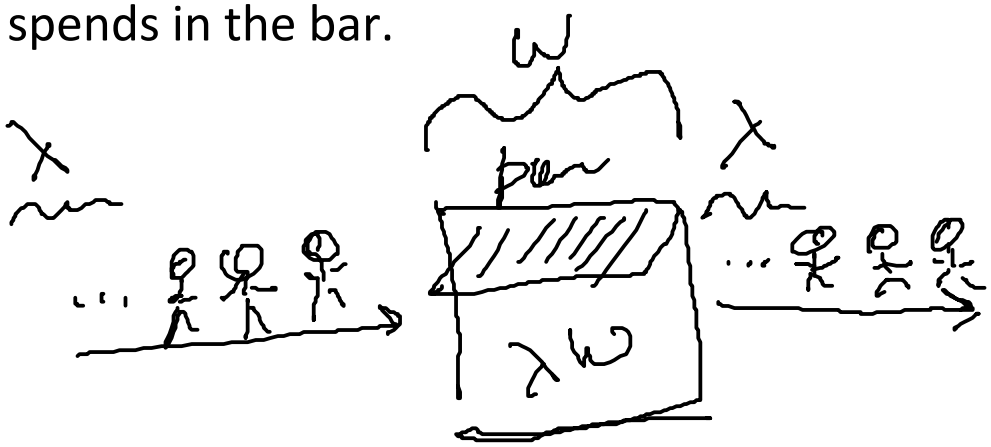
Thursday, February 18, 2010
9:47 AM

An easy analogy (due to Daniel Menasce)

Customers wait in line for a crowded bar.

Steady state means average arrival rate λ = average departure rate λ.

Little's law says that the number of customers in the bar is equal to λ times the average time a customer spends in the bar.

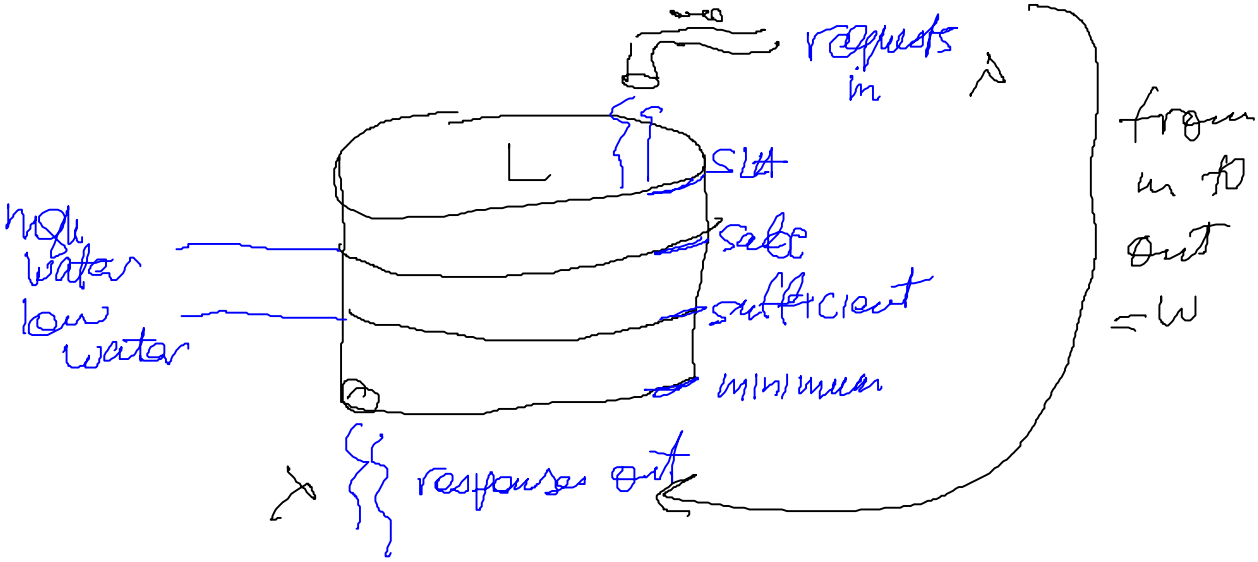


The basin analogy

Thursday, February 18, 2010
9:05 AM

Little's law and the basin:

- Requests coming in are λ.
- Requests flowing out are λ.
- Water level is "average requests in system" L.
- W is response time = t_{server}+t_{cloud}.
- In this lecture, we ignore $t_{network}$, and assume that it is governed by its own SLA.



Impact of Little's law

- "Average number of requests awaiting service" (L) and "average time spent waiting" (W) are interchangeable concepts!
- "Average requests waiting for service" L is a concept of **load** L=λW.
- Can characterize the effect of L as either
  - Changes in arrival rate **λ.**
  - Changes in the time spent waiting **W.**
  - Where performance **P = W + $t_{network}$.**
- Thus performance **P = $t_{network}$+L/λ**

Load averages

In a modern system, the load average A is defined as the average number of processes ready to run and awaiting CPU time, including the ones that are running.

A=0: everybody waiting for something, no one ready to run.

A=1: on average, one process is ready to run

A=30: on average, 30 processes are "competing" for the CPU(s).

Load average is a **relative** measure:

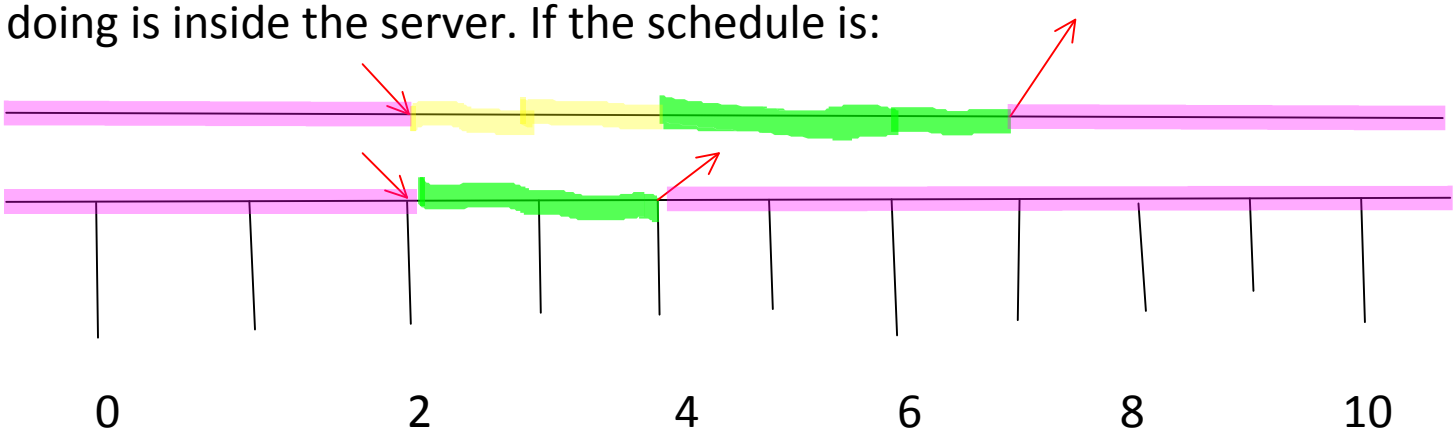
On a single-core machine, A=30 is **bad**; system is **overloaded**.

On an 8-core machine, it's quite **comfortable**!

Understanding load

Thursday, February 18, 2010
11:36 AM

Suppose we have 2 requests pending and all they are doing is inside the server. If the schedule is:



Green=running, yellow=ready, pink=waiting (for something)
Then the load average is (2*2+3*1)/10 = 0.7.

Friday, February 19, 2010
9:32 AM

Two concepts of load

- Load as outstanding requests ($L_{request}$): correlates with request time-in-system, SLA violations.
- Load as pending computation(L_{CPU}): reported by operating system (as output of uptime command).
- How are these related?

Relating two kinds of load

Thursday, February 18, 2010
10:03 AM

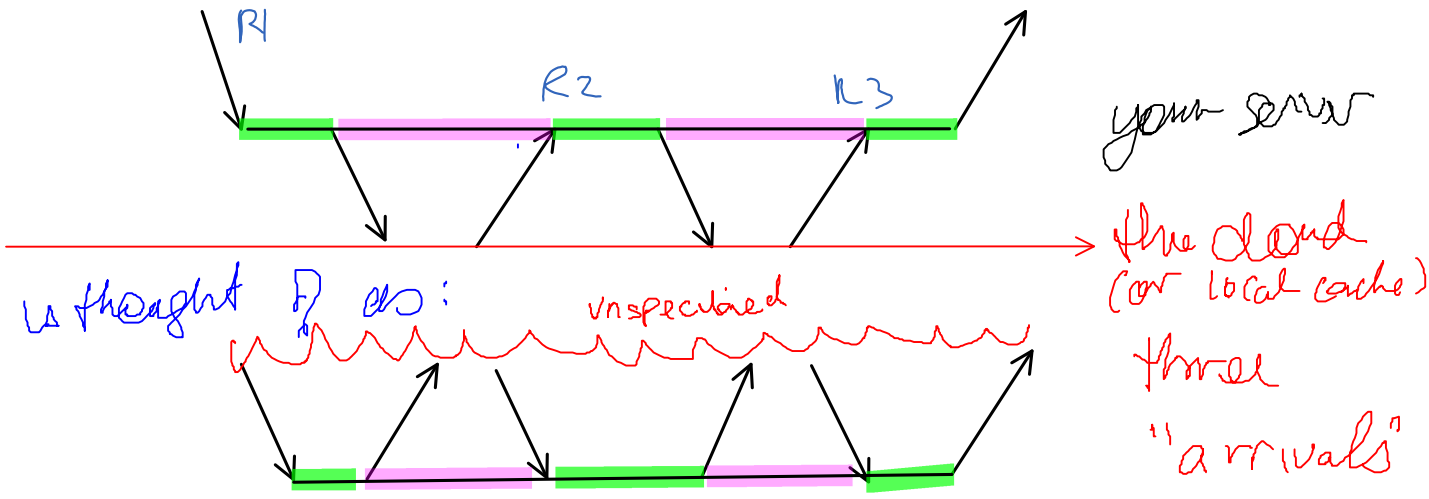
Relating $L_{request}$ and L_{CPU}

$L_{request}/\lambda_{request}=W_{request}$ = P-$t_{network}$ (and from now on, we will ignore $t_{network}$).

$L_{CPU}/\lambda_{CPU}=W_{CPU}$ where λ_{CPU} counts both arrivals from outside ($\lambda_{request}$) and in addition, treats responses from the cloud as "requests".

Because of the simple structure of cloud (edge) services as programs, $\lambda_{CPU}=\alpha\lambda_{request}$ for some proportionality constant $\alpha \geq 1$.

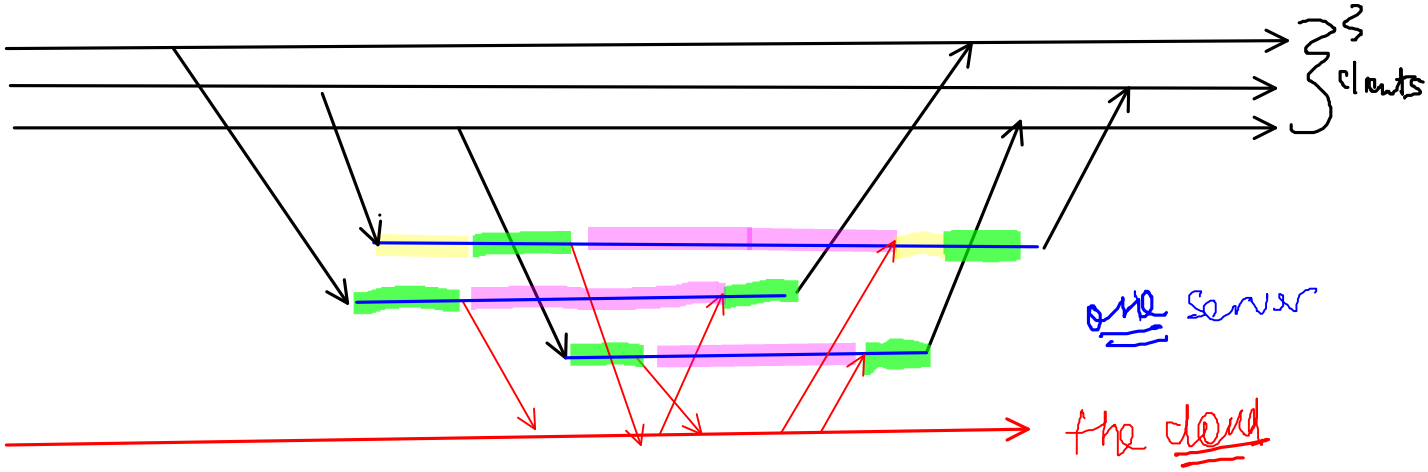
$\alpha = 3$



Reconciling cloud "as a service" vs the cloud "as load"

Thursday, February 18, 2010
4:44 PM

Reconciling the cloud "as a service" vs the cloud "as load"



A complicated dance

Response time P depends upon t_{server},
which depends upon load and schedule.
Schedule is determined by $\lambda_{request}$ = request arrival rate.
Over a long enough time, Little's law says that (average) t_{server}= mean time in system = $W_{request}$ = $L_{request}/\lambda_{request}$.
But equivalently, $W_{CPU}=L_{CPU}/\lambda_{CPU}$ and we know L_{CPU}, but not $L_{request}$.
We **do not know** $W_{request}$, but we do know it is **proportional** to W_{CPU} (via some unknown constant), which is **proportional** to L_{CPU},
which is the output of the UNIX uptime command.

Thus we know that L_{CPU} is proportional to $W_{request}$, which is internal response time, but we **do not know the proportion.**

A recharacterization

Thursday, February 18, 2010
10:30 AM

A recharacterization of control

Load average L_{CPU} is proportional to time-in-system $W_{request}$.

$W_{request}$ is hard to measure, while L_{CPU} is easy to measure.

So, we **watermark L_{CPU},** and use that to provision to control $W_{request}$ and thus $P=W_{request}+t_{network}$!

Note that the load average on a **specific machine** is **representative** of the load average on **all machines**, provided that the **servers are all the same** and **switching is flowless round-robin**.

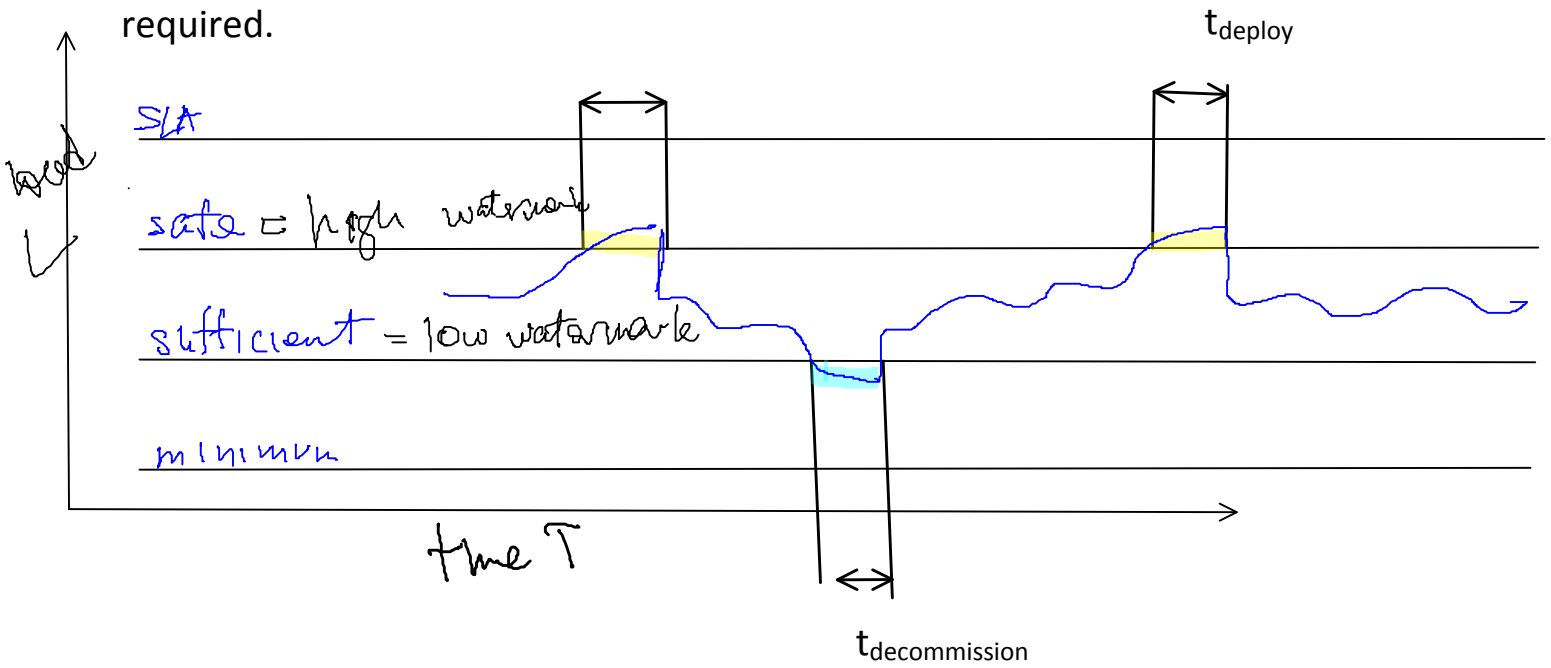
Watermarks

Thursday, February 18, 2010
8:18 AM

Watermarks

L_{safe} is a **high watermark**: going higher is risky.

$L_{sufficient}$ is a **low watermark**: going lower isn't required.



The point of the last two lectures

Wednesday, February 16, 2011
2:39 PM

The point of the last two lectures

The customer defines an SLA which -- together with time to deployment -- determines the high watermark W_{safe}.

The load average defines the low watermark $L_{sufficient}$, as a load average of 1 unit per core (everybody busy, on average, and no one waiting).

In practice, we guess about the value of L_{safe}!

Limitation of domain

Thursday, February 18, 2010
2:04 PM

The key to this story is **limitation of domain**

"In general", there is no correlation between L and λ on a regular computer because **other things are going on.**

But we "stacked the deck" in the sense that only our services are running and answering requests.

Thus, **in this situation only**, there is strong correlation between L and λ.

Load and virtualization

Thursday, February 18, 2010
2:06 PM

There is strong relationship between load and virtualization scheduling:

If we get 50% of the CPU cycles on a machine, that (roughly) doubles our load over the case in which we get 100% of the machine.

Likewise, if we increase from 50% to 100% of a physical server, that (roughly) cuts our load in half.

Scotty, I need more power!

There is also a direct relationship between load and power.

On average, "computing" takes slightly more power than remaining "idle".

Thus the load on a server is (roughly) proportional to its power consumption over a baseline idle consumption:

power consumption $J \approx J_{base} + \alpha W$ where

J_{base} = **idle power consumption** in Joules.

W is **work** related to **load**.

α is some **constant of proportionality.**

The real problem

Wednesday, February 16, 2011
4:56 PM

Meet the SLA
Subject to: minimize power.

Minimizing power means:

- minimizing swings between server roles.
- minimizing the number of powered up servers.
- minimizing thrashing: unneeded context switches.

From disadvantage to advantage

Thursday, February 18, 2010
2:58 PM

From disadvantage to advantage

Old computing model: big loads are bad, because we **can't handle them**.

Cloud computing model: small loads are bad, because we **can't analyze them.**

In cloud computing, it becomes an advantage to **clump lots of small loads together** so that their composite behavior can be analyzed.