THE C10K problem

concurrently handling ten thousand connections

“

> The C10k problem is the problem of optimising network sockets to handle a large number of clients at the same time.

“

The C10k problem is the problem of optimising network ~~sockets~~ to handle a large number of clients at the same time.

> “
>
> The C10k problem is the problem of optimising network sockets / network bandwidth / RAM / CPU / OS to handle a large number of clients at the same time.

Little's Law

$$L = \lambda W$$

“

The long-term average number of customers in a stable system L is equal to the long-term average effective arrival rate λ, multiplied by the average time a customer spends in the system W expressed algebraically: $L = \lambda W$.

But what does it mean for us

$$L = \lambda W \rightarrow \lambda = L/W$$

L - system capacity

λ - average request arrival rate

W - average request processing time

To handle more requests, we need to increase L, our capacity, or decrease W, our processing time, or latency.

What Dominates the Capacity (L)

- number of concurrent TCP connections
- network bandwidth
- RAM
- CPU
- OS

BUT does it?

number of concurrent TCP connections

- Server can support several tens-of-thousands of concurrent TCP connections

- Some have had success maintaining over 2 million open connections:
  http://blog.whatsapp.com/196/1-million-is-so-2011

network bandwidth

- Requests travelling back and forth are no more than a few kilobytes in length (well under 1MB).
- Given today’s high-bandwidth LANs, our network could support anywhere between 100K to over a million concurrent requests.

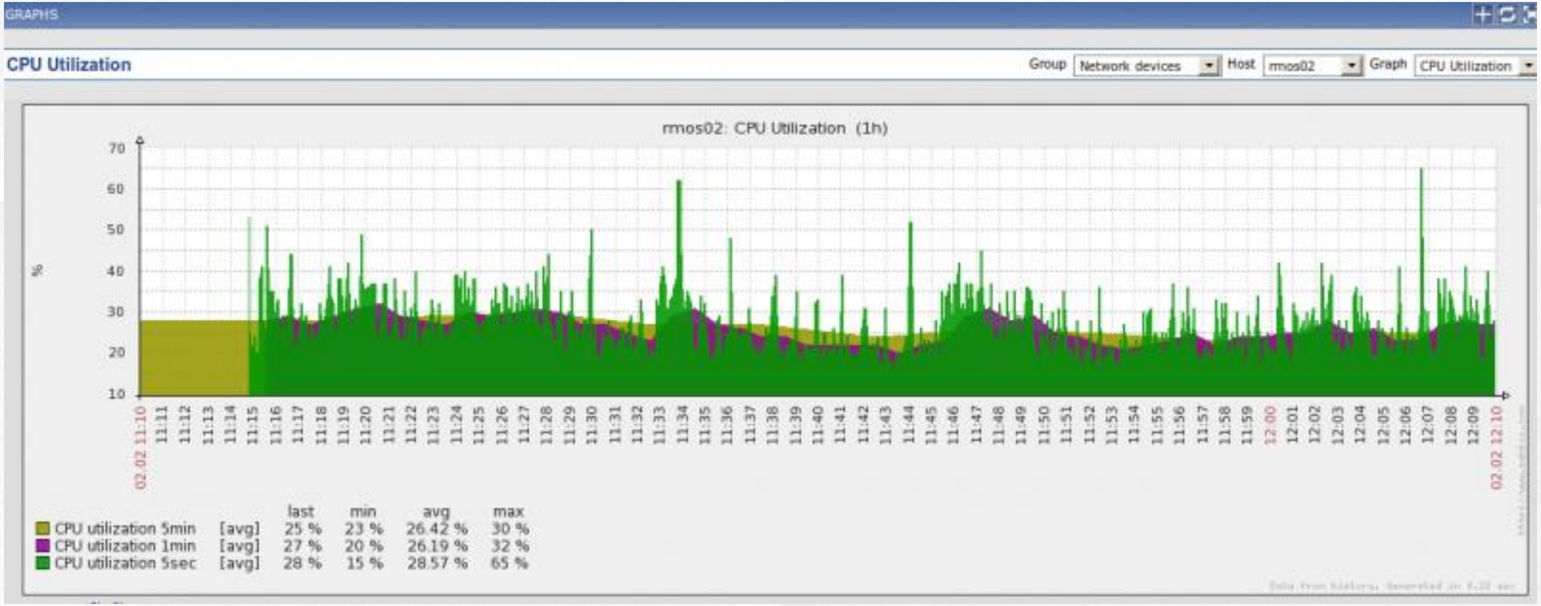
RAM

- how much memory each request consumes
  - let’s assume we can keep it below 1MB
- with 500GB of RAM it makes more than 0.5 million **simultaneous** requests
- with 32GB of RAM which is normal even for laptops it makes hundred thousands **concurrent** requests

CPU

- It depends on the application logic ;)
- Most of the time our requests just wait for:
  - microservices
  - database
  - files
- Usually this number is anywhere between several hundreds of thousands and several millions...

in practice productions systems rarely report ~~CPU as their bottleneck~~



GRAPHS

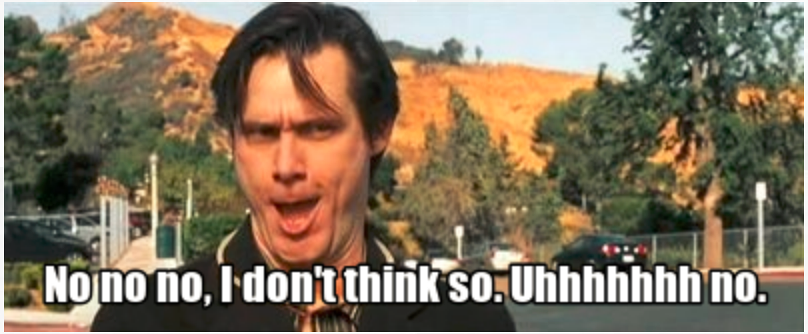
CPU Utilization

Group: Network devices Host: rmos02 Graph: CPU Utilization

		last	min	avg	max
🟨 CPU utilization 5min	[avg]	25 %	23 %	26.42 %	30 %
🟪 CPU utilization 1min	[avg]	27 %	20 %	26.19 %	32 %
🟩 CPU utilization 5sec	[avg]	28 %	15 %	28.57 %	65 %

L = 100K - 1M ???

L = 100K - 1M ???



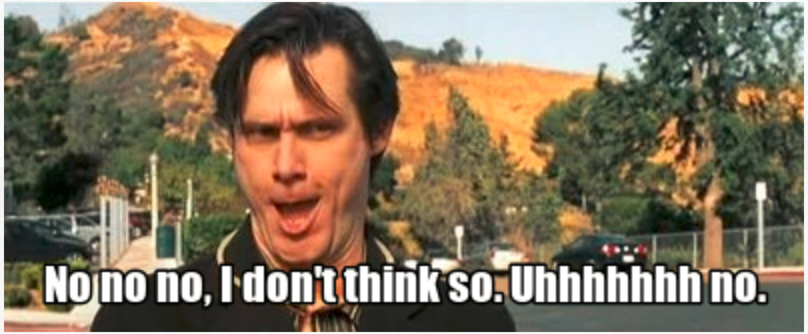
OS

- Number of concurrent request we can handle is also limited by the number of threads the OS can handle
- This number is somewhere between 500 and 15K
- OK, I know there is NPTL for Linux but let’s leave it for now...

http://paultyma.blogspot.in/2008/03/writing-java-multithreaded-servers.html

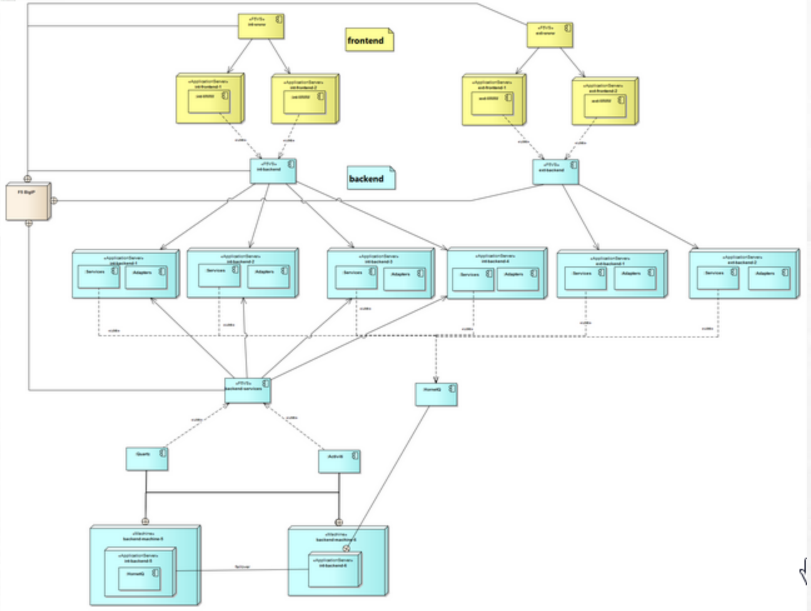
L = 10K ???

L = 10K ???



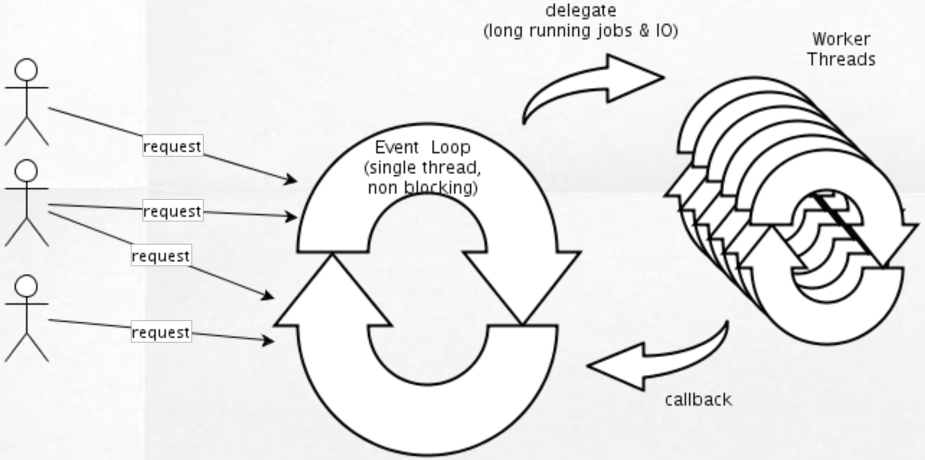
Enterprise architecture ;)

- clusters
- load balancers
- caches
- session replication
- sticky sessions
- ...



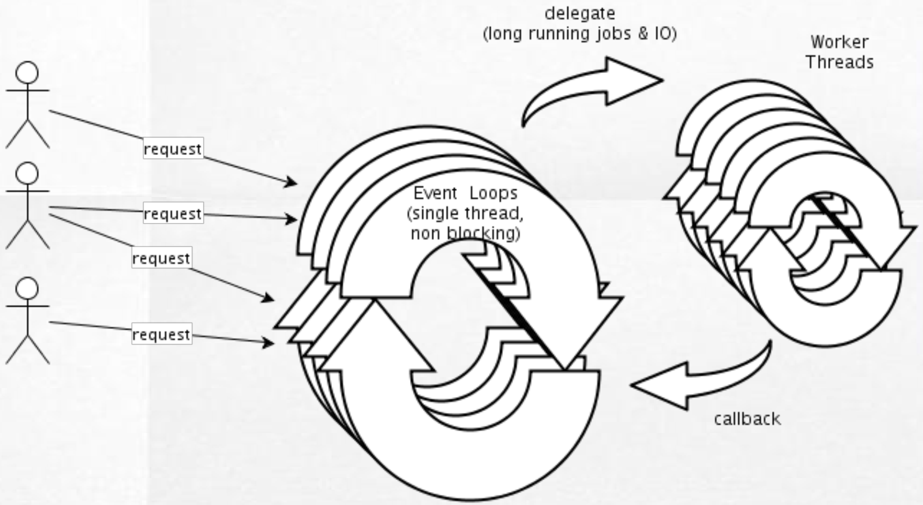
Reactor pattern

- Event Loop
  - non-blocking
  - single thread
  - event handling
- Worker threads
  - concurrency
  - thread pool
  - asynchrony



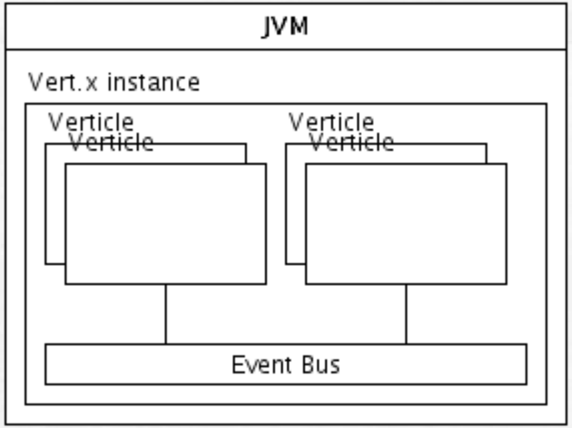
MultiReactor pattern

- Event Loops
  - non-blocking
  - multiple threads
  - event handling
- Worker threads
  - concurrency
  - thread pool
  - asynchrony



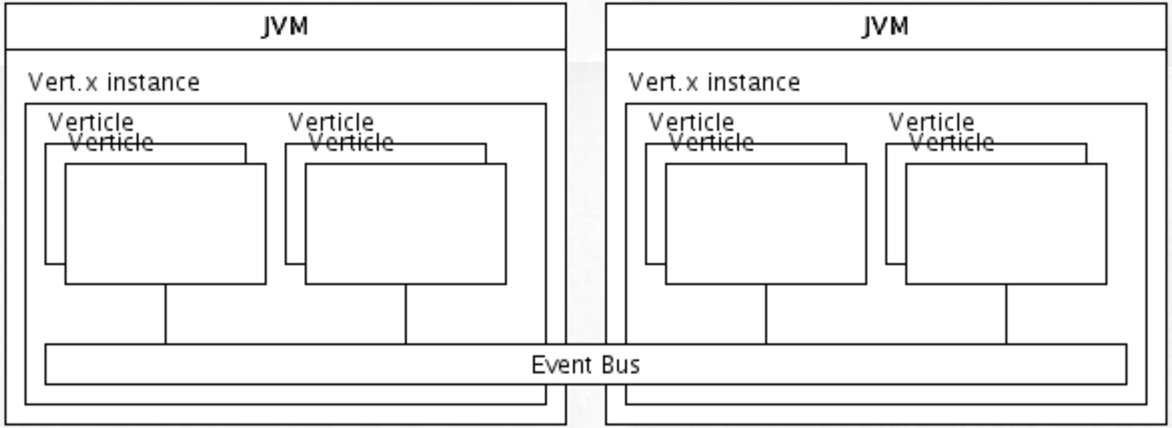
Vert.x overview

- Vert.x instance
- Verticle
- Event Bus

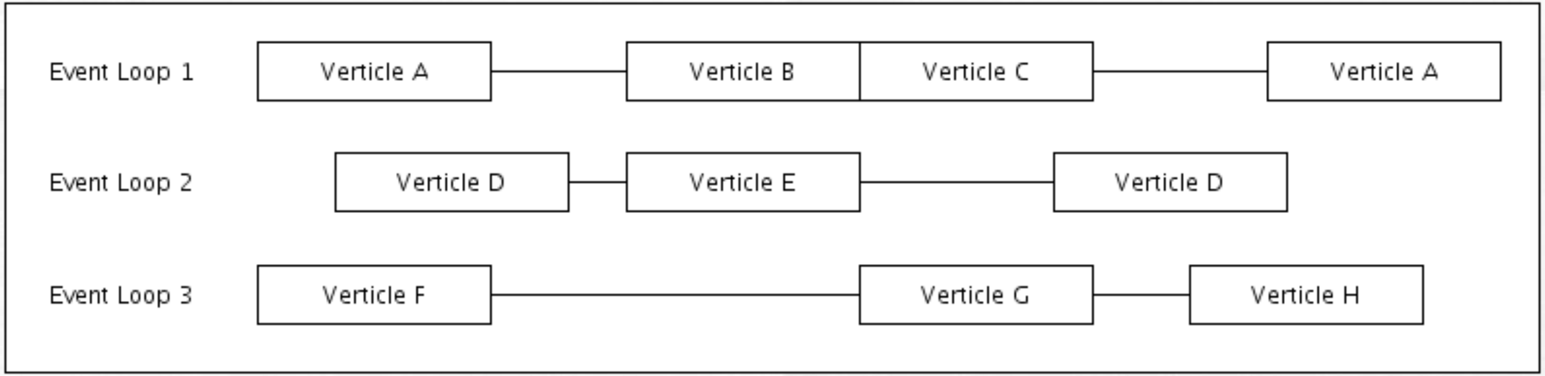


Vert.x clustering

- Event Bus @ Hazelcast



Vert.x loops



Event Loop 1 Verticle A Verticle B Verticle C Verticle A

Event Loop 2 Verticle D Verticle E Verticle D

Event Loop 3 Verticle F Verticle G Verticle H

Links

- http://blog.paralleluniverse.co/2014/02/04/littles-law/
- http://blog.paralleluniverse.co/2014/05/29/cascading-failures/
- http://en.wikipedia.org/wiki/Little%27s_law
- http://en.wikipedia.org/wiki/Reactor_pattern