CSCI 1380

Day 3 [Jan 28]

Theophilus Benson

Last Class

(1) Motivation for D.S. (i.e. MapReduce)

(2) Failures (heartbeats)

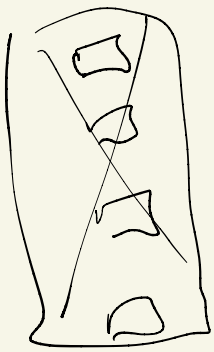
This Class

(3) Networking

(4) Fallacies of distributed computing

(5) Performance optimizations

First: the heartbeat dilemma



a company with 10k servers.

30k of these servers fail & restart. What sort of bad things can happen?

AWS

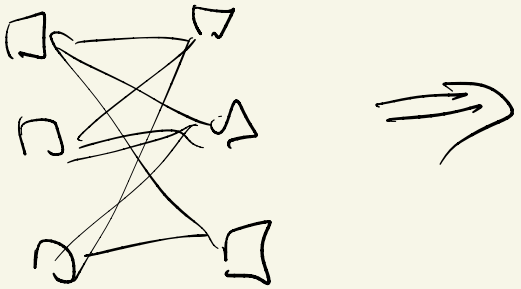
new cluster of machines are added

all machines try to connect to each other

a thread to manage connections

solution : more hardware

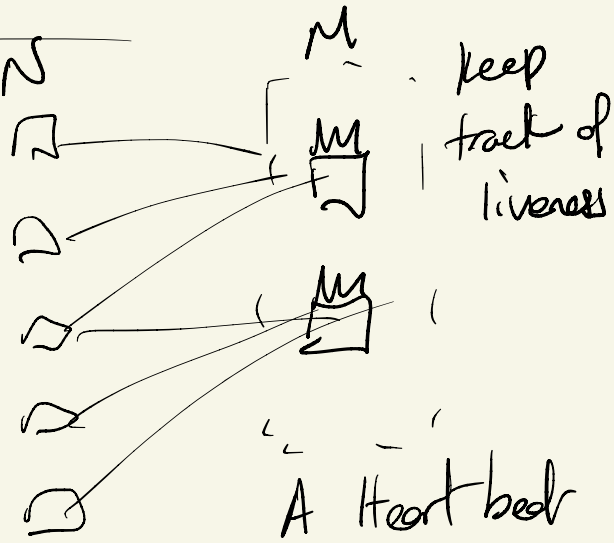
FB



N x M

lots of overhead

but finely



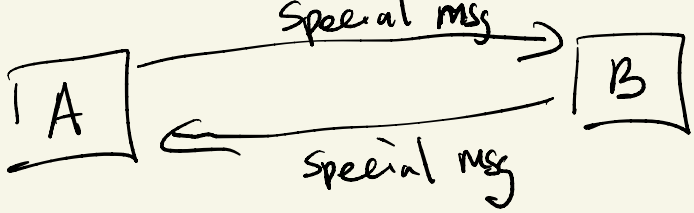
O(N x M)

less of overhead

but takes a little longer to get info

Protocols (TCP/UDP)

TCP



(1) need to make a connection with special msg (TCP handshake)

(2) after handshake: send/receive data

(3) special rate control algorithm & retransmit algorithm

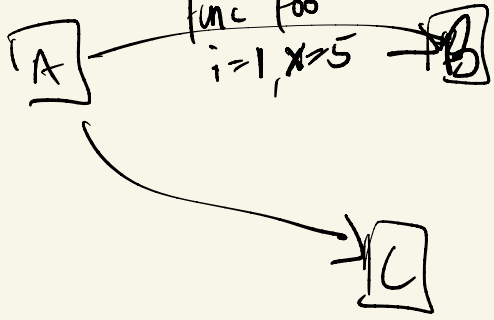
You never worry about lost packets & TCP let's you share the network fairly

Problems

(1) overheads

(2) lots of messages

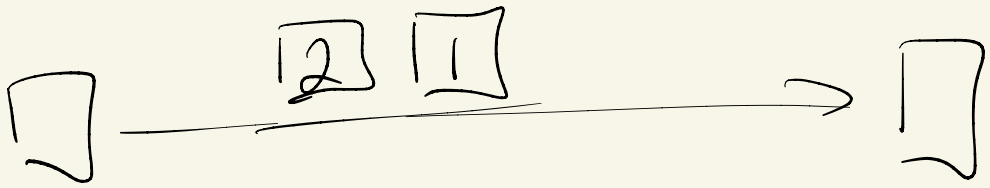
(3) No one cares about lost packets



remote function calls (RPC) ⇒ UDP

You want to avoid the startup overhead of TCP

UDP (reordering / loss)



How to go from monolithic to distributed?

App.c

(1) makes distributed calls: needs info about other components

(2) should we create each as a unique process?

keep track of connections

metadata: where are the processes located?

"Port"

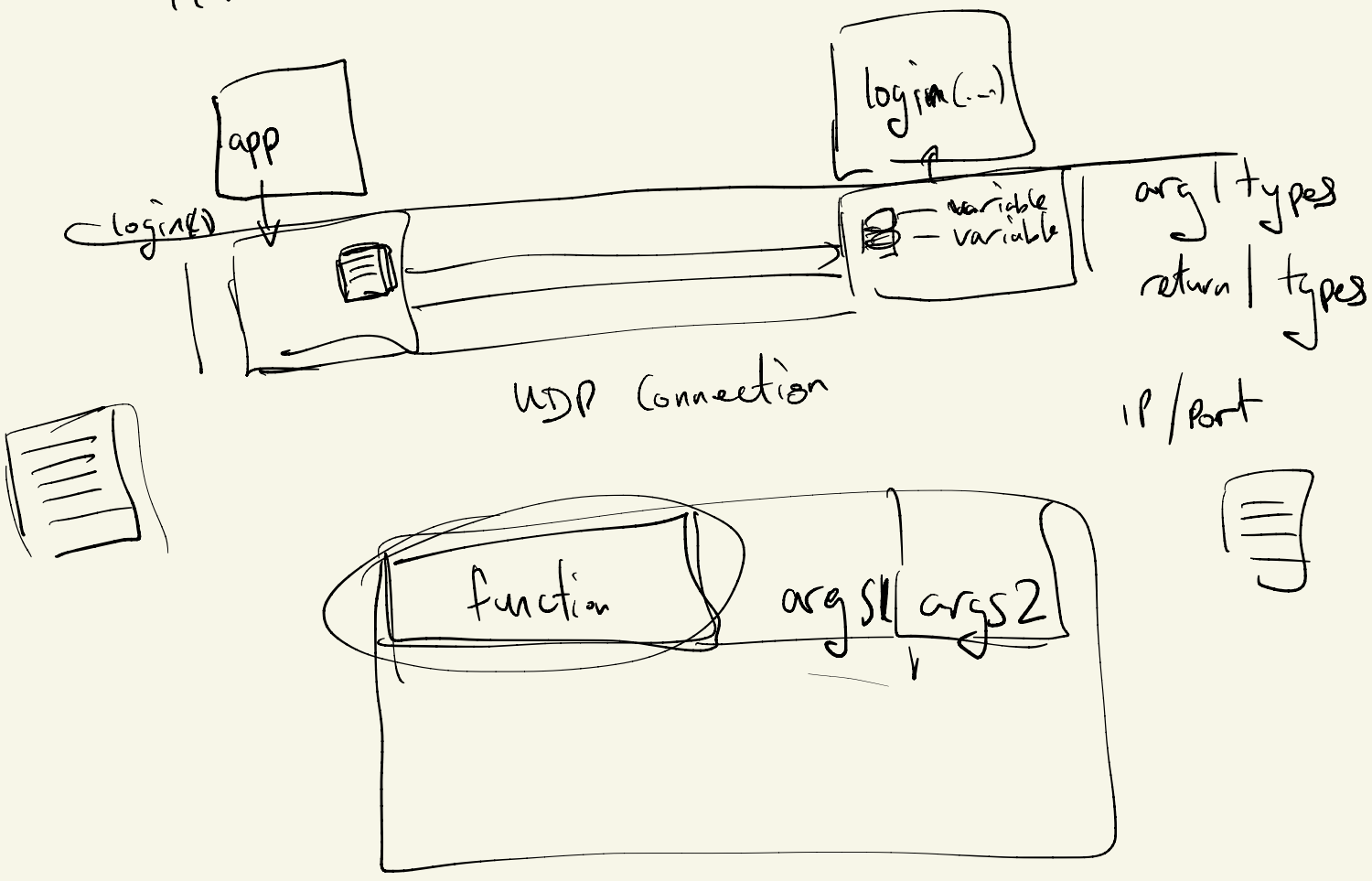
IP address + port

① each function becomes a process

② "app" needs information about the other process

③ "Special" to make network connections

RPC ⇒ Remote Procedure Calls



(1) Heartbeats (when things go wrong)

(2) N/W ⇒ (TCP v. UDP)

(3) socket code | RPC

Next class

RPC semantics | RPC structure

Performance | loadbalancers