Peer to Peer Networks

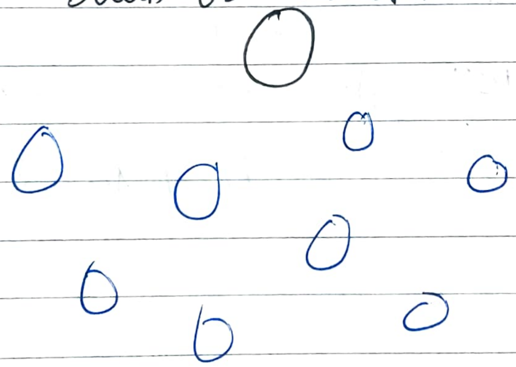
We gonna understand this P2P network concept thru use case

Problem : Gotta a find a soln. to distri. abt 5GB of data frm a server (that has thruput = 40Gbps = 5GBps) to a 1000 machines all over the globe

eg:- video footage frm CCTV camera (which you get every 15 mins) and you wanna share it all over.

eg 2:- large ML models that you wanna train on 1000machines, deployed multip times/day

I] M-1 → Plain ol 1 server → 1000 machines



For 1 machine

5 GB file recd in = ~~5G~~ $\left(\frac{\text{file size}}{\text{thruput}}\right)$ = 1s

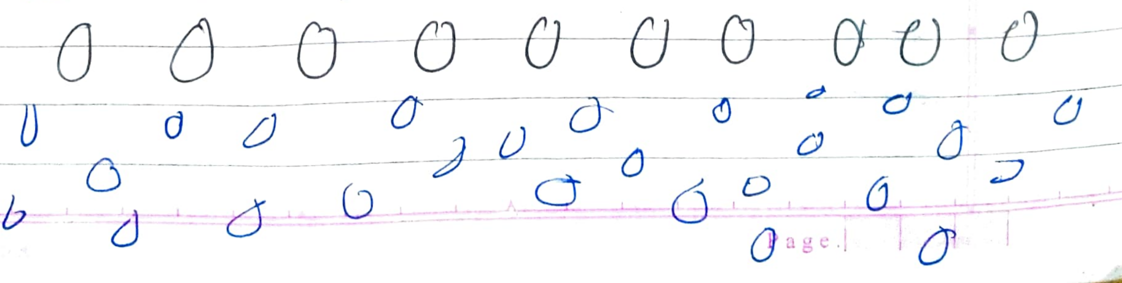
∴ After 1st machine, next machine gets data and so on

∴ for 1000th machine

5GB file recd = after 1000s

= 17mins ← Pretty slow + cloggin at 1 server

II] M-2 → $\overset{\text{Replica}^n}{\text{Hori}}$ scalin of server → 10 servers → 1000 Machines



Now, say each ~~machine~~ server handles equal amt of machines (say 100th machine)

Say 1 server + 100 machines = 1 set.

For 100th machine in any set,
5GB file read after = 100s = 1.7 mins

still kinda slow

Disadvan. of M-2 → You gotta replicate data from main server to other servers viz pretty inefficient when the file size is real freckin large.

III M-3 → Sharding : data from 1 server $\xrightarrow{\text{split}}$ 10 shards

each of the 1000 machines gotta visit all shards to accumulate all the data

Disadvan. → cloggin at shards

IV M-4 → P2P networks

each of the 1000 machines = peer

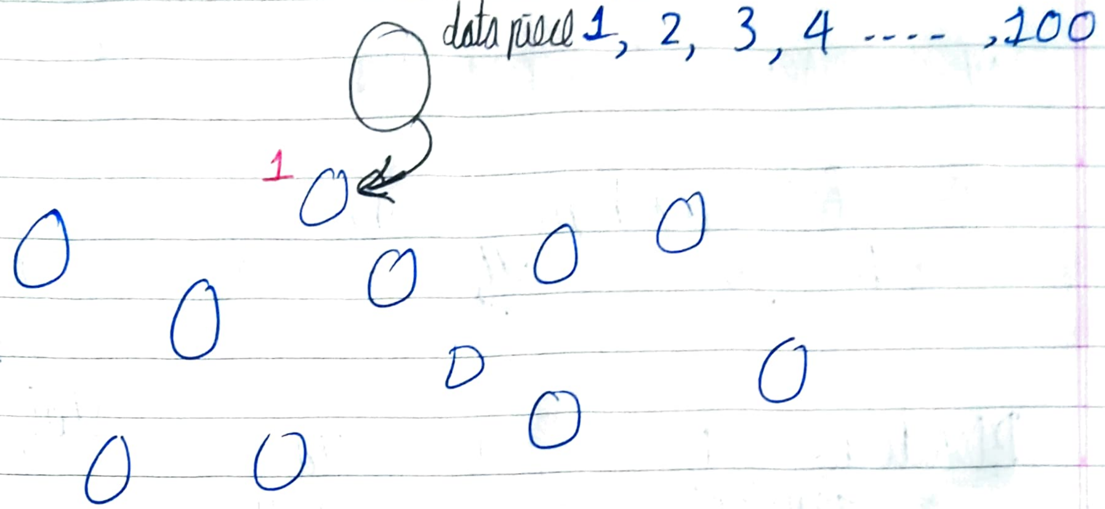
eg:- number each part

P2P network → 1) Data from main file split into UIDable parts and given to diff peers

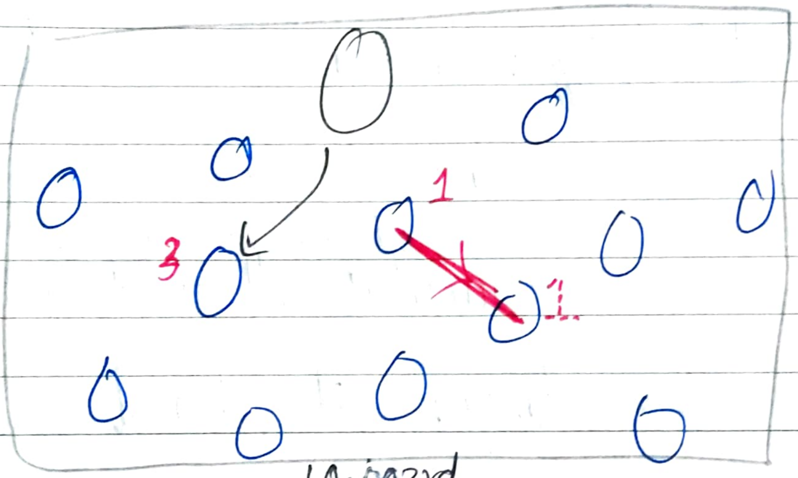
2) Whilst some peers is gettin data from main server, other peers talk to each other to share & acquire pieces of data that they already don't have to piece togetha entire data.

eg: 0.001st sec

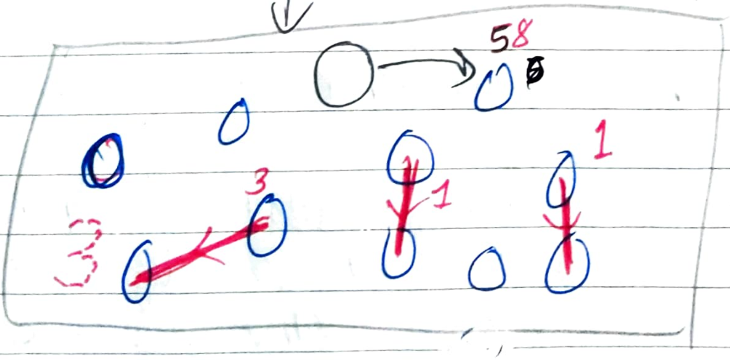
each data piece = 5MB



0.002nd sec

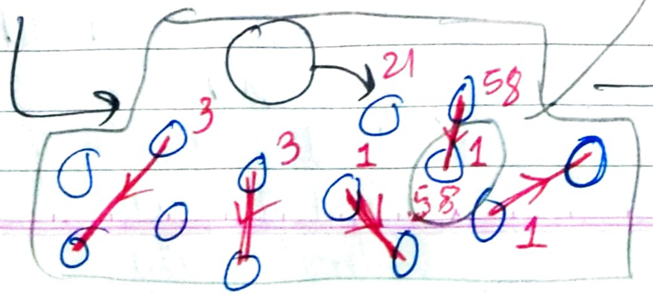


0.003rd sec



now these peer has piece 1, 58 i.e. 2 pieces outta 100

0.004th sec



flows exponentially each sec

For P2P networks to f^n optimally → peers gotta know which peers to talk to next

→ to give them data

→ to get ~~[illegible]~~ back data to build up rest of the file that they're missin

Peer discovery / selec^n

Ways by which peers know what peers to comm. with / transfer data to / get data from next.

M-1 : Central DB/machine a.k.a tracker

Whilst peers are talkin to each other, they also comm. with this central machine which tells em which peer to ~~[illegible]~~ comm. with next

M-2 : Gossip a.k.a. Epidemic Protocol

① Peers talk b/w themselves and figure out b/w themselves.

eg:- Peer 1 talkin to Peer 2

Peer 2 to Peer 1 : Thanks for piece 1!
Oooh btw you should totally talk to Peer 47
She got ~~[illegible]~~ some big chunks you might wanna get a load of

② Analogy
→ people gossipin, sharin things b/w themselves as well as abt others ppl they've talked to
→ People comin in contact, durin an epidemic, and spreadin virus (for us → a data piece) like wildfire

Implementaⁿ ↓

③ Every peer carries
→ certain chunks
→ mappings that map certain peers to certain pieces of data

eg:- Peer 2 maps Peer 33 to Peer 99 to help Peer 33 get a piece frm Peer 99

④ After each gossip
↓
knowledge ~~of~~ that a peer has w.r.t. "what peer holds what piece of info" ↑ i.e its hash table of IP addresses of other peers that it needs to go to retrieve missin data pieces

⑤ Idea of havin this knowledge viz effectively a mappin, viz effectively pieces of a hash table (which could've contained all (IP address of peer) → (piece it has)) is a.k.a a $\underbrace{\text{distributed hash table}}_{\text{D.H.T.}}$

(6) P2P networks often operate usin the concept of a D.H.T. to figure out which peers hold what pieces of data

eg:- Kraken (dev.ed by Uber) → at its peak, it distri.s ~~20~~ 2×10^4 100MB - 1GB blobs in under 30s

(7) P2P is fast af as contrary to other meths., here, all machines talk to, share with all machines

eg:- Torrenting

1 machine → 1 large file (eg:- movie)

spread this file in chunks to machines (peers) all over the region / world

then these peers work together to obtain all missin pieces i.e- puzzle em back together → each peer gets full file.