Agenda

- Designing Orchestrater
- Multi Master

9pm — 11:30pm

10-15 mins

doubt session → 1pm

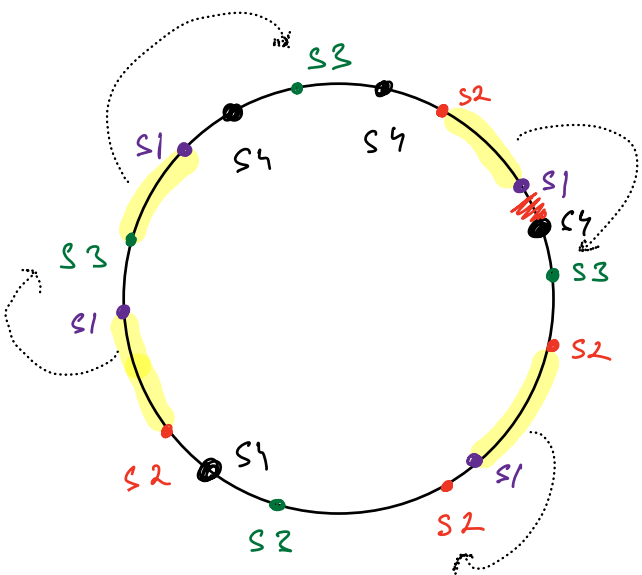
SQL / NoSQL

↳ sharding happens automatically

how?

Consistent Hashing → what happens when a server goes down → consistent hashing automatically redistributes load

But how is data copied?



Sharded based on some key → user-id

3 shards → S1, S2, S3

k hashes → 4

~~S1~~ down

Replication → how do we maintain replication?
↳ master-slave

Problem Statement

Design data orchestrator
- → NameNode (Hadoop)
- → JobTracker
- → Hbase Master

↳ automatically add/remove machines as needed
- ↳ if new servers are added they get assigned a shard
- ↳ creating new shards

↳ ensure equal load & storage distribution at all times → consistent hashing

↳ maintaining replication level

Given config
- ↳ sharding by → user_id
- ↳ replication level → 3
- ↳ 10 servers

} Orchestrator manages everything else

Suppose → 10 servers are already working

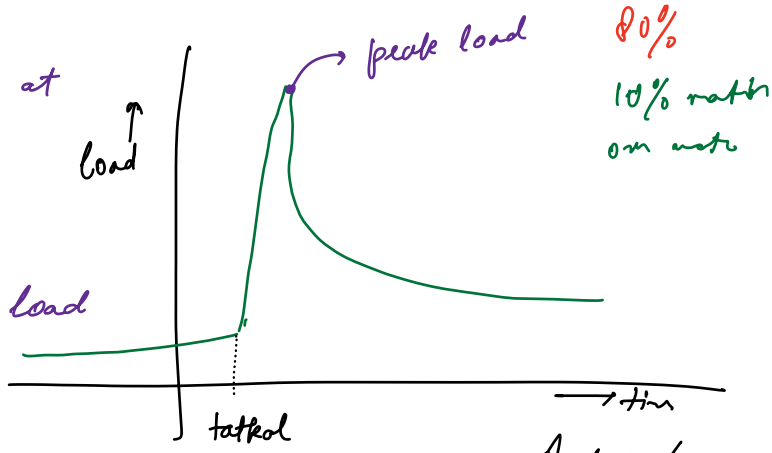
given 2 new servers
idle

Monitor memory/disk utilization for each shard
↳ cross a threshold
↳ move some data to new servers.

Reacting to load on a shard

Drawbacks:

If we let sharding happen at any time (reacting to load)
↓
create shard during peak load
↓
move data from 1 machine to another
↓
more traffic → uneven situation



80%
10% math on note

Activity/load graph

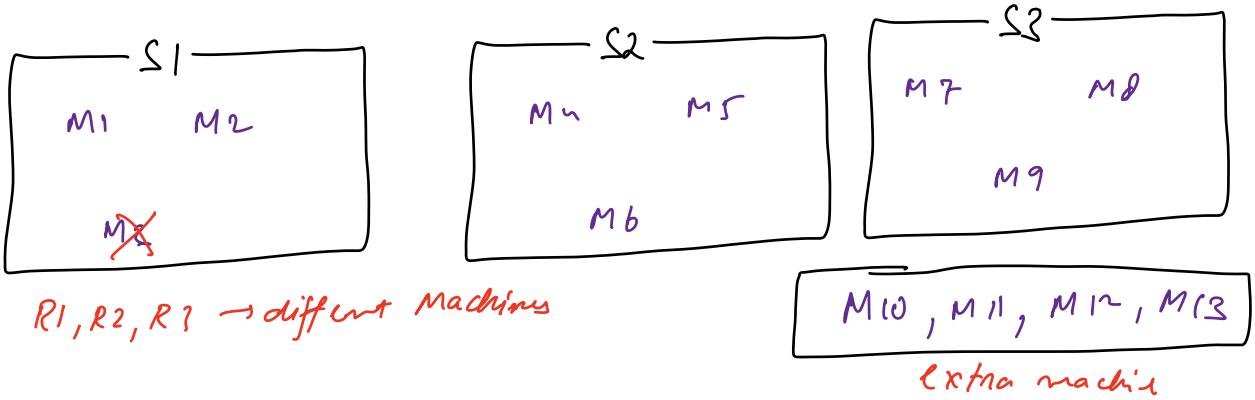
→ Let us not react to load at time of load
→ when we have extra servers available → assign a role to them

---

Replication levels → 3

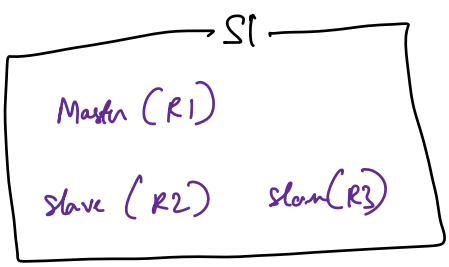
3 shards → S1, S2, S3

Rajeev DB



R1, R2, R3 → different machines

Master-Slave for replicas



highly available (inconsistent)
↳ write → Master
↳ read → any Slave

eventually consistent
↳ write → Master + 1 slave
↳ read → any slave

highly consistent
↳ write → all slaves + master.

Asid Ranking System
↓
highly available
& eventually consistent

Questions

How to add or remove machines?
- if extra machines are available then what should system do?
- what happens when a machine dies

Possibilities when you have extra machines

replication level = 3
# of shards = 3

→ create new shard ($N \geq$ replication level)
→ add machine to existing shard
→ leave them idle

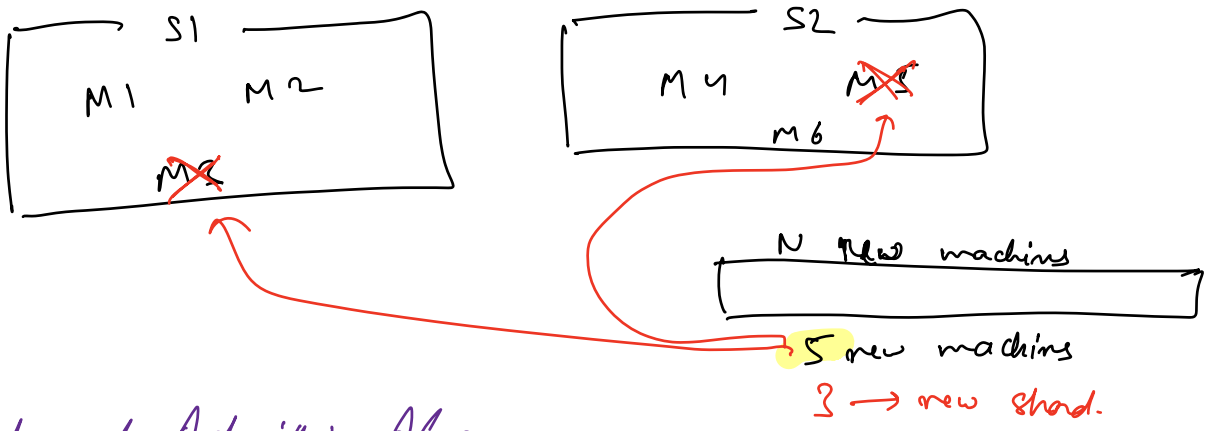
Ashwith's Algorithm

① N Extra machines

N // R new shards

N % R → idle

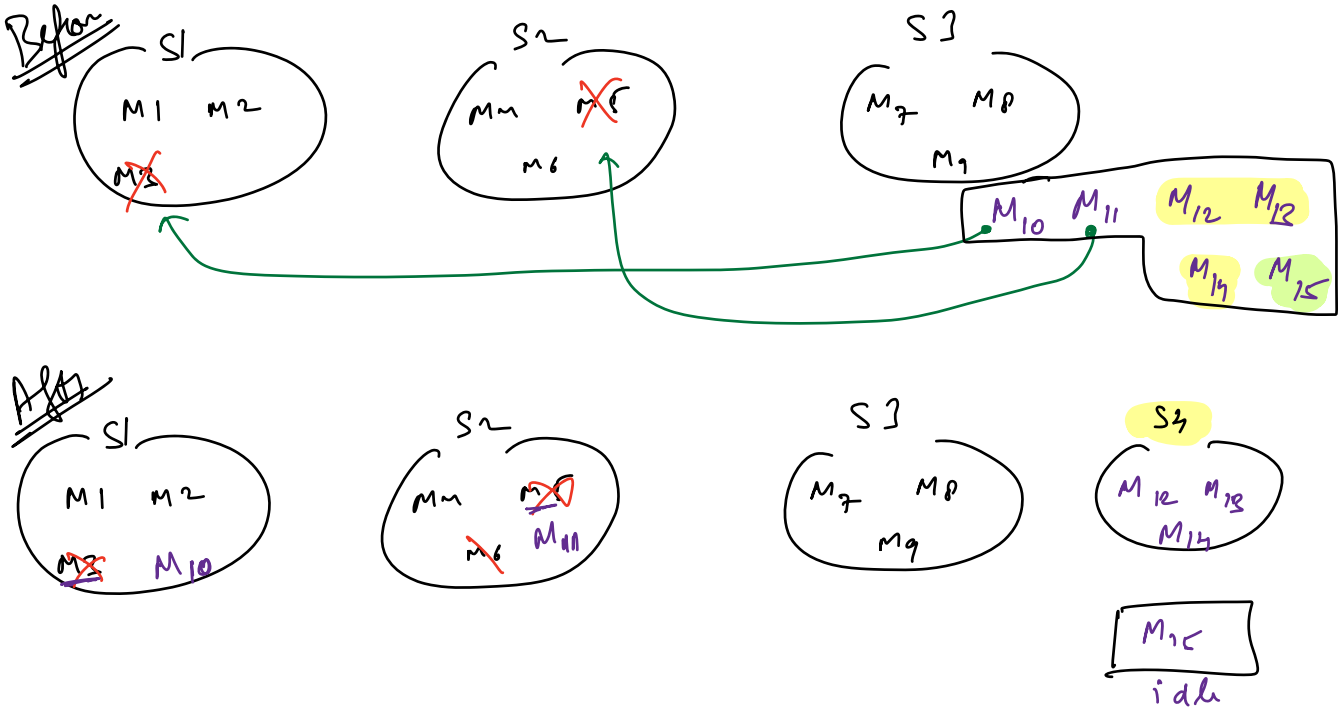
eg. = 1001
333 new shards
2 machines remain



Improved Ashwith's Algo

- priority → maintain data consistency (don't want to lose data)
  - → replace downed machines
- if there are machine left over after replacing downed machines
  - create N'//3 new shards
  - N'%3 → idle

Situation



Insight

↳ don't immediately create new shards if you have enough machines

↳ ∵ some machines might break in future

at all times we should have X machines available just in case some machine in a shard breaks.

How to calculate how many machines should we have as backup?

1000 shards → 1000 machines available as backup

Machine → goes down once every 2 months
↳ 2 days to repair

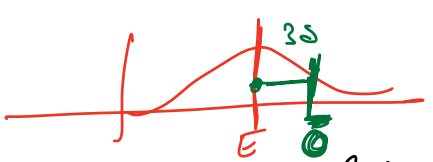
Big Table
NoSQL

in a period of 60 days → available for 58 days

Prob that a machine is down = $\frac{2}{60} = 1/30$

S shards each with Replication level of R
↳ total S * R machines

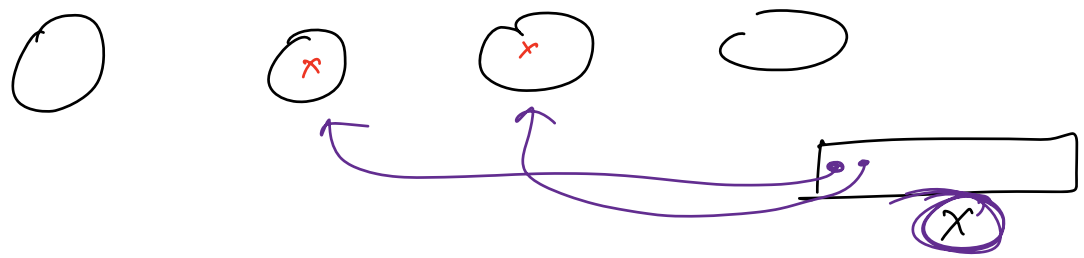
Calculate X → F (machines that down at a given time) → avg.



1.5 * E

↳ How many backup machines should I have so that with 99% prob I have sufficient backups

whatever remains → create a new shard if possible
(N - X)



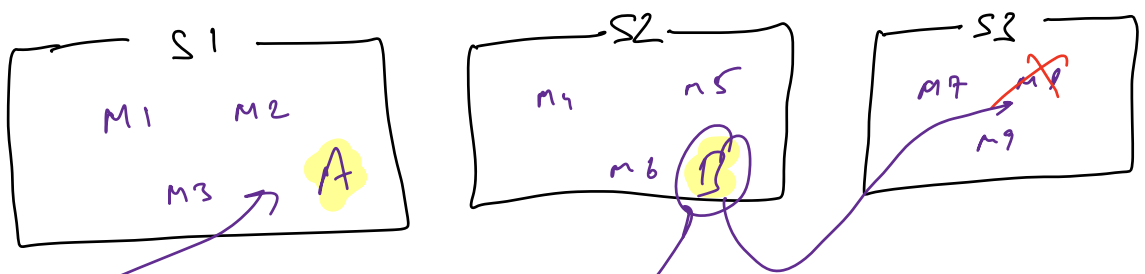
idle machines?

$(N - X) / 3$ → shards

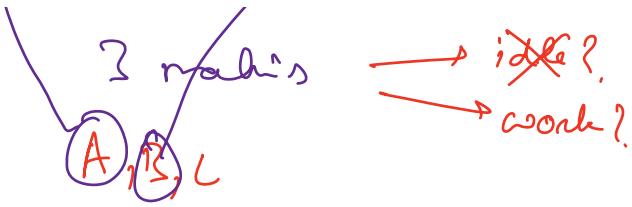
new backup

$(N+X)\%3$ → remainder

X machines in backup → idle?



Replication factor = 3
X (backup) = 2 machines
15 extra machines
13 ext → 4 new shards ⇒ 12



Instead of having idle machines

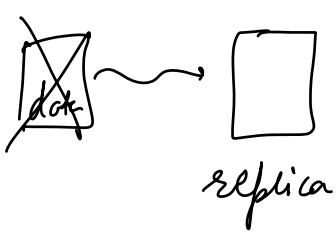
↳ use them as extra replicas

↳ contract: if a machine goes down in some other shard this extra replica will be reclaimed.

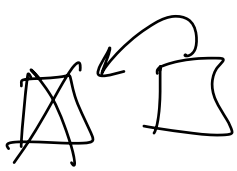
Replicas / Hot copy

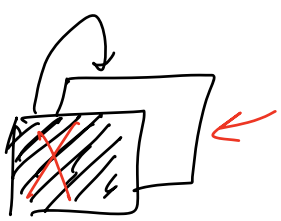
{

data



load balancer
App server
Orchestrator
} Stateless



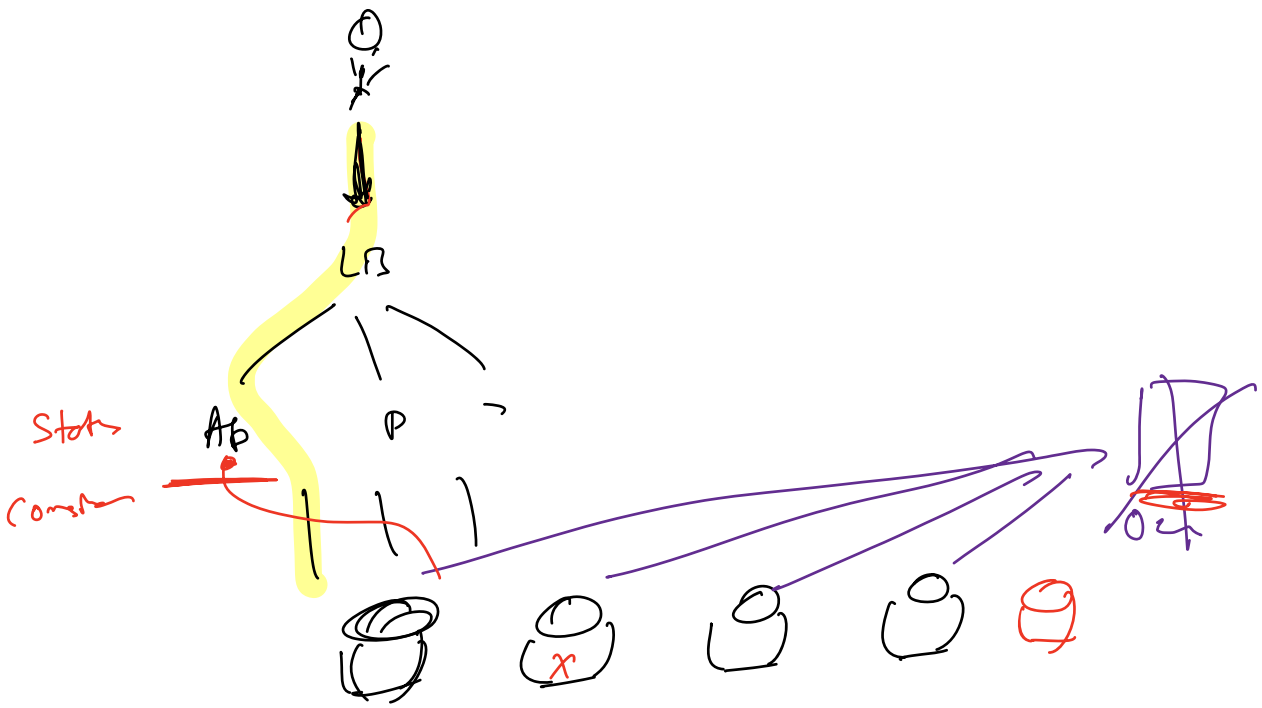


10:40 → 10:50

How is data moved

↳ new shard

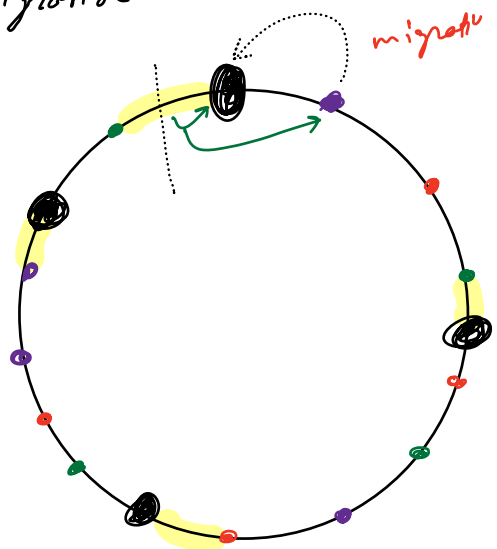
↳ a new read replica is added?



Seamless Shard Creation

Cold start

- new shard → 0 data
- data migration → 2 step



Sh shard has to be added.

Staging Phase

→ don't really add shard Sh to the consistent hashing
↳ upper layers (db client) no idea that new shard will be added.

→ run Simulation
determining what keys (user_id) → range of hashes will be directed to new shard once it gets added.

→ start copying data → starts at 11 pm
↳ 15 min

↓
11:15

→ shard is warmed up.

However any write requests b/w 11 pm & 11:15 pm still go to old shard

So new shard is missing that data.

Real Phase

↳ shard actually goes online (db clients are made aware of a shard)

→ we need to catch up with the data can from 11:0pm to 11:15 pm

Small data

↳ Available → new shard start serving requests

copying this will take around 1 min

↳ Consistent → 1 min → reject any requests

Timeline

T1 (11 pm) → Orchestrator decides that new shard will be added

Simulation Phase

Start copying data.

T2 (11:15 pm) → copy done

New shard → live

T3 (11:16 pm) → △ updates are completed

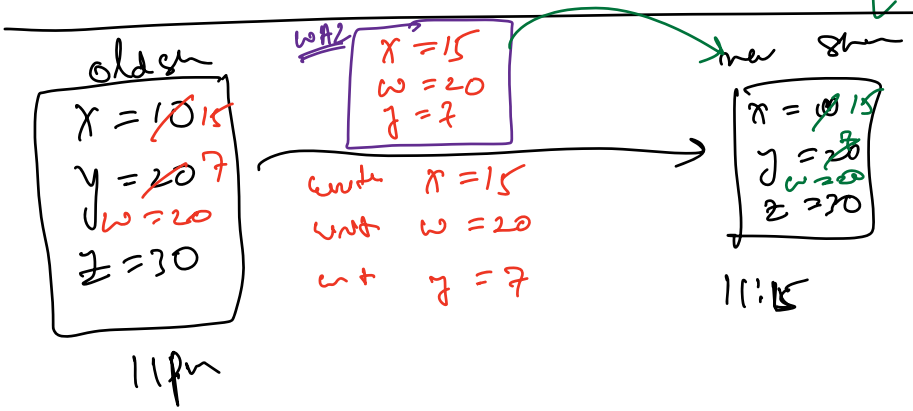
full old shard to new data

SQL / No SQL

↳ Write ahead Log (WAL)

x = 10 W

x += 10 W (R)



WAL → general DB operation to provide Durability, Atomicity

Roll back in case of power failure

→ improve write speed (SQLite)

X backup machine → estimation X can be challenging.

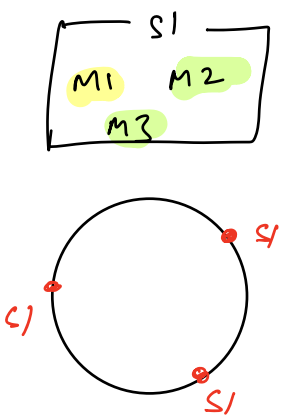
Dynamo DB / Cassandra

Multi - Master

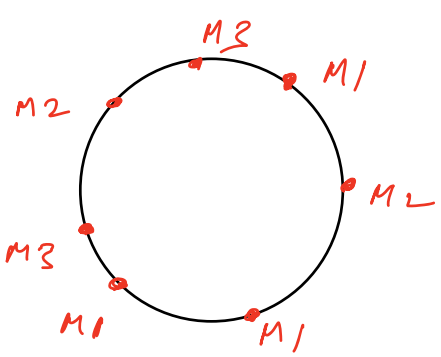
→ no slaves

→ every machine is a master

Cassandra

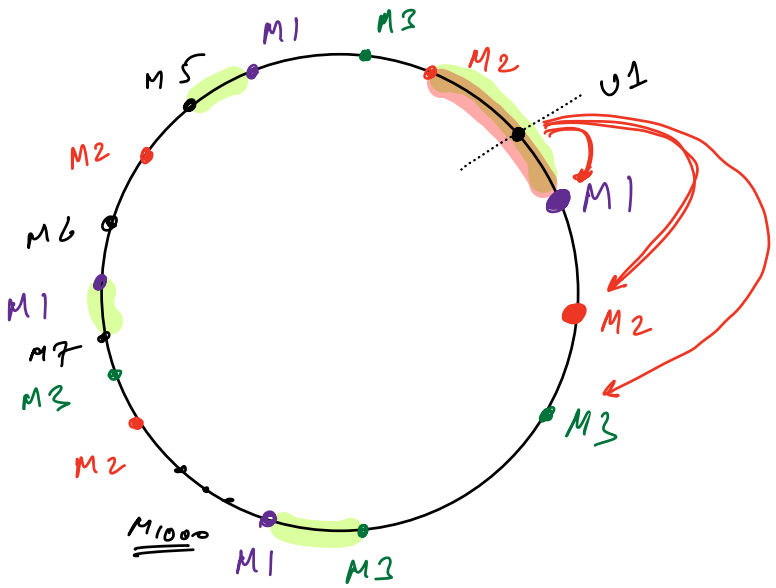


multi-master



machines are not grouped into shards → each shard is just 1 machine

→ Assume replication factor is 3.



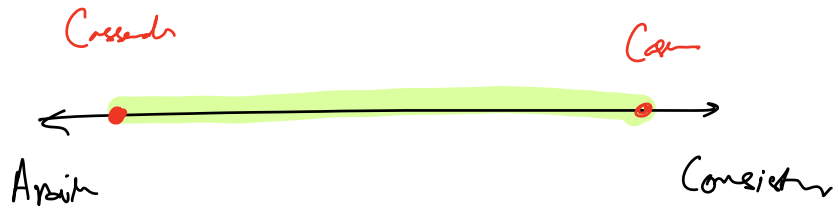
U1's data is stored in M1 if M1 crashes

→ Optimal to store replicas in servers which are next in line to serve U1's requests

→ Note: M2 is not a replica of M1 ✗

it stores replicas for any users whose req will go to M2 in case M1 dies.

Tunable Consistency



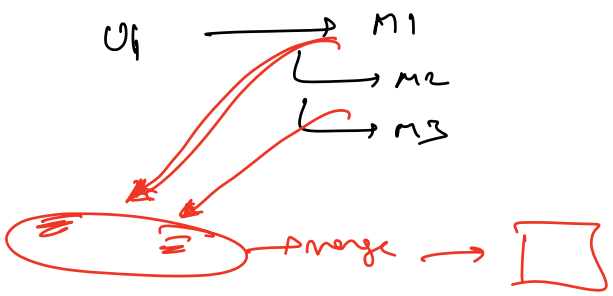
Configure two variables R & W → Min # writes

↓

Min # reads

X → replication level

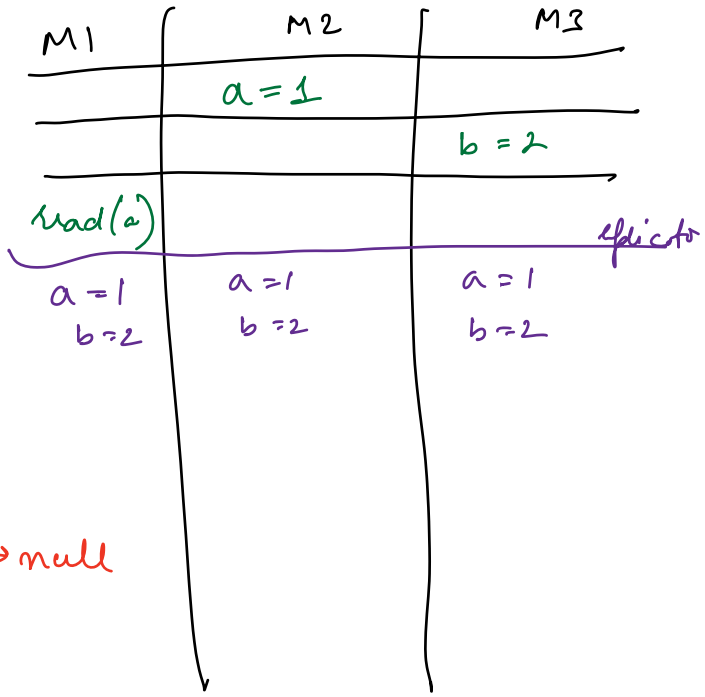
X = 3 R = 2 W = 2



$N = 3$

$R = 1$ how many copies must be read

$W = 1$ how many copies must be written



U1 'data' → write a = 1

→ write b = 2

→ read (a) → null

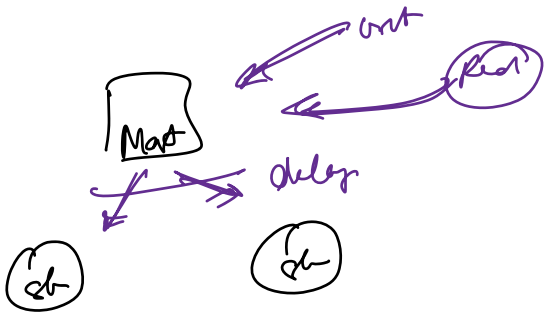
Consistent / Available?

$R \uparrow$ $W \uparrow$ $\Rightarrow$ higher consistency

If R is high then befor succesful read
for read to access multiple copies
if copy is down → read unsuccesful
More consistent but less availa

R+W → the higher this is, the more consistent your syst will be.

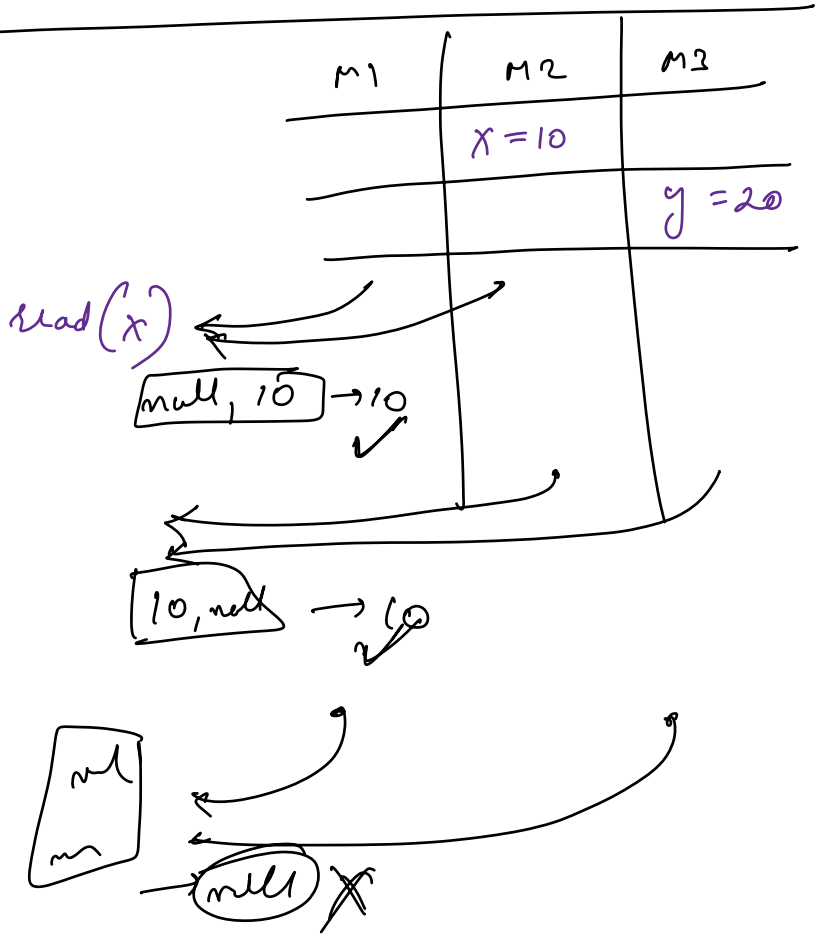
CAP → if you make a read → data is the latest data



$X = 3$

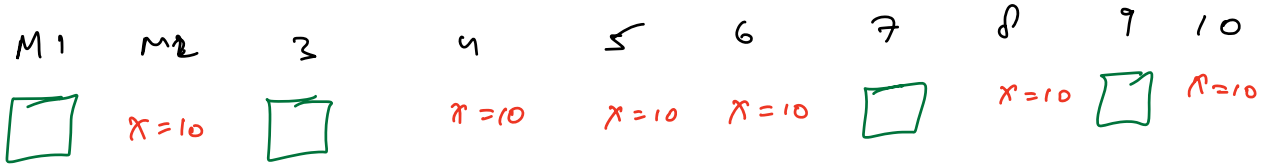
$R = 2$

$W = 1$



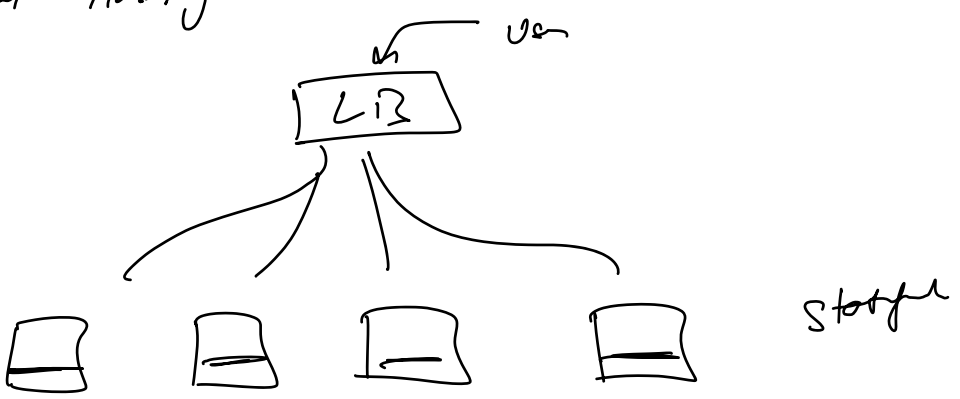
$R + W > X$ ⟶ highly consistent syst.

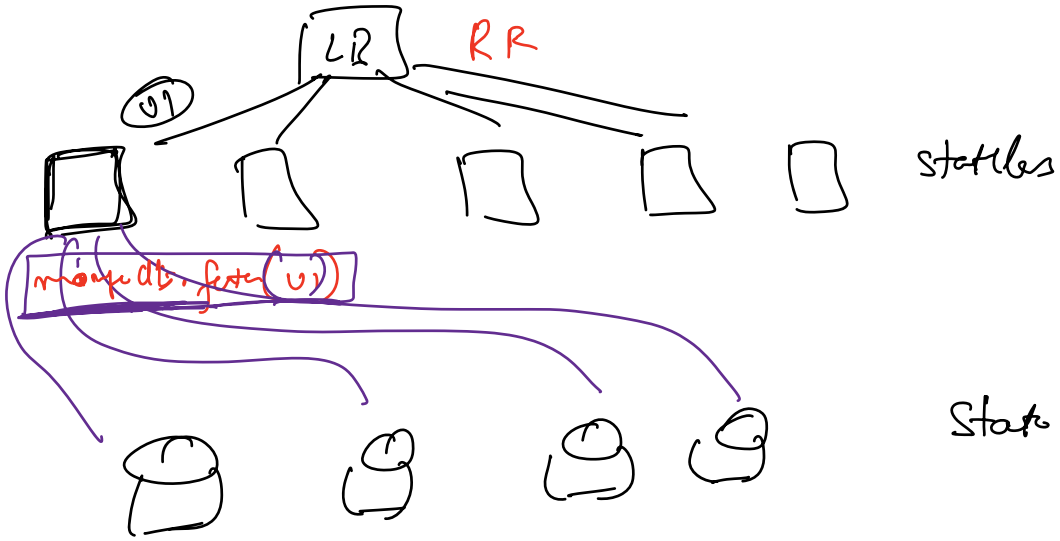
$X = 10$ $R = 5$ $W = 6$



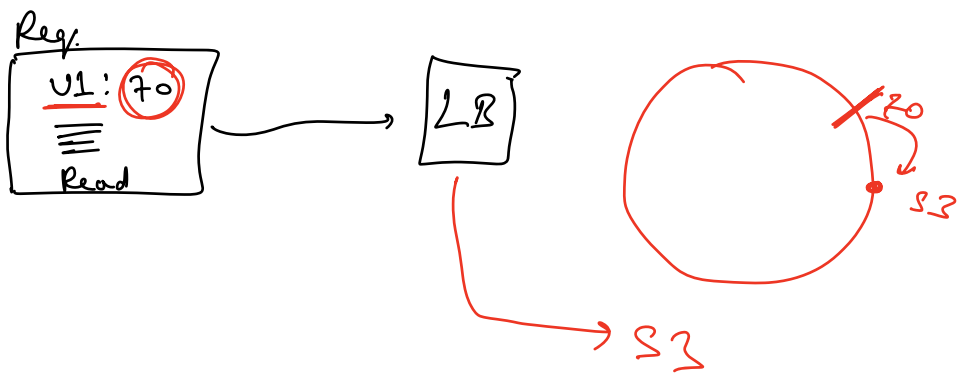
8th node → Wrd

Consistent Hashing ⟶ Stat

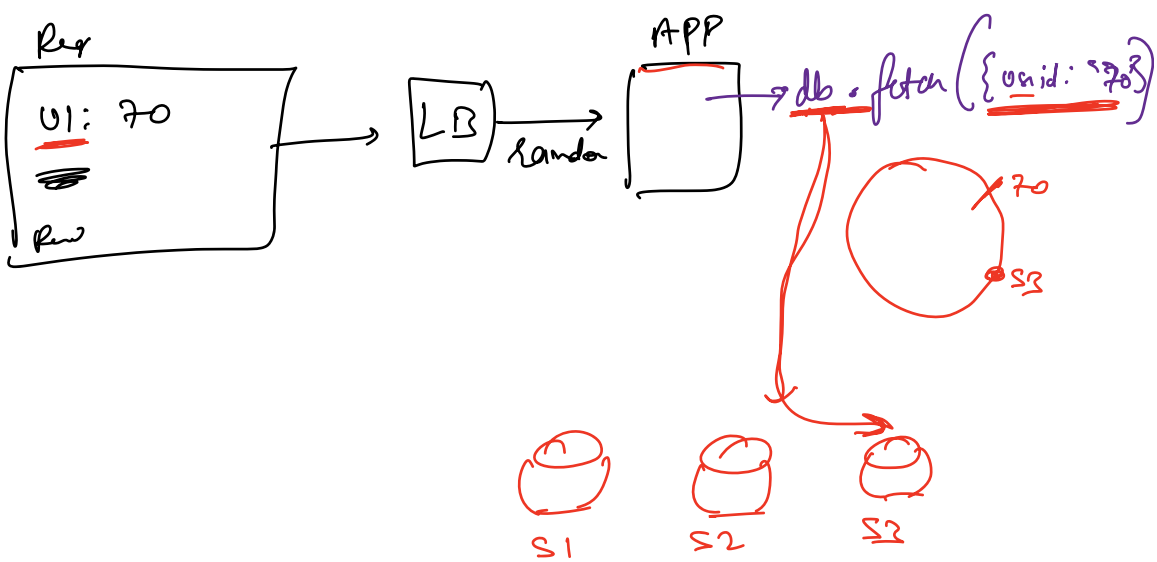


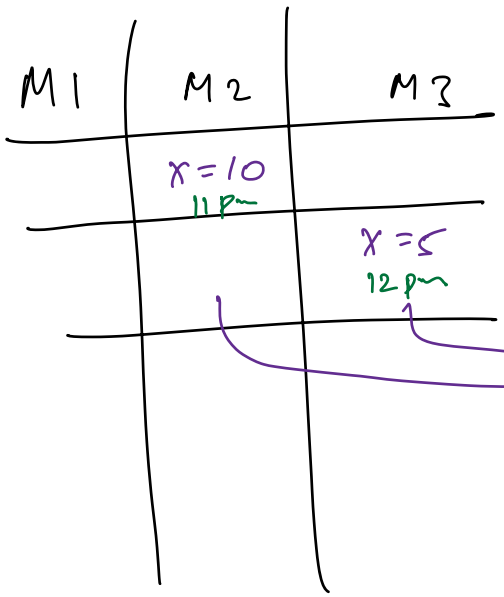

LB
RR
U1
statfiles
mongodb.find(U1)
Stats

Earlier


Req
U1: 70
Read
LB
70
S3
S3

Now


Req
U1: 70
Read
LB
Random
APP
db.find({uuid: "70"})
70
S3
S1
S2
S3



$X = 3$

$W = 1$

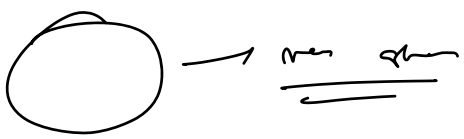
$R = 2$

Read

M2 → x=10 11 pm

M3 ~~10~~ 5 12 pm

N new machines → X books



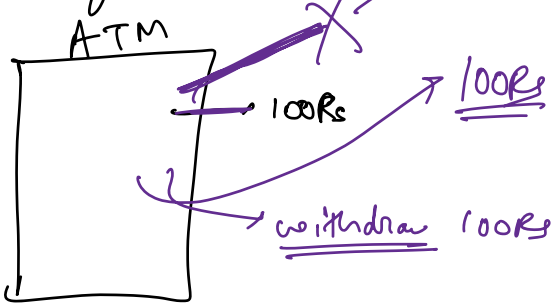
---

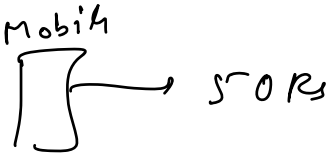
100 Rs → 50

Demand draft

for 10Rs

Banks log all data





Server push app → websocket connection with server.

whatsapp → 2 million+ WS per server.