Replicaⁿ and Sharding

1) A dB is critical to sys.
2) ∴ Sys.'s performance ∝ dB's performance.
3)

	dB	Sys
eg 1)	unavail.	unavail.
eg 2)	low thruput	low thruput

4) ∴ While des-in a sys and makin it performin, we gotta ensure its dB is performant, else sys will fall apart

I] Replicaⁿ

The act of duplicatin data frm 1 dB server to others

Use case (i) : To act as backup if main dB server fails

(1) When main dB goes down, clients can't perform any opns. (read, write, etc.) on the data

(2) To prev. clients frm experiencin this we establish a secondary dB (backup) i.e. a replica of the main dB.

(3) For this to happen the main dB, on receivin req.
- → performs whatev opn (read, write, etc) the req. is for
- → updates the replica s.t. replica === main dB i.e. in 'sync' with dB at any given pt. in time

④ We do this to ensure that replica DB can take over in case main DB fails

⑤ main DB fails → replica takes over as new main DB → after some time → main DB comes back up → now, main DB is updated by replica ← now, they swap roles, i.e. main DB take over again as main DB → replica goes back to acting as a standby.

⑥ Write opns to main DB take a lil longer as you gotta write data in ~~main~~ DB and replica to ensure replica is always 'in sync' (i.e. up-to-date) w.r.t. main DB
"Each write opn of main DB should sync. by be done in replica. If write opn. fails on replica due to some reason (eg:- network partn"), then the write opn. should also not be completed on main DB"

⑦ Replica should NEVER have stale data.

⑧ Redundancy of sys ↑

Use Case (ii) : To move data closer to clients, thereby ↓ latency of accessing ~~data~~ specific data.

① Consen" : When you save data to a DB, it gets

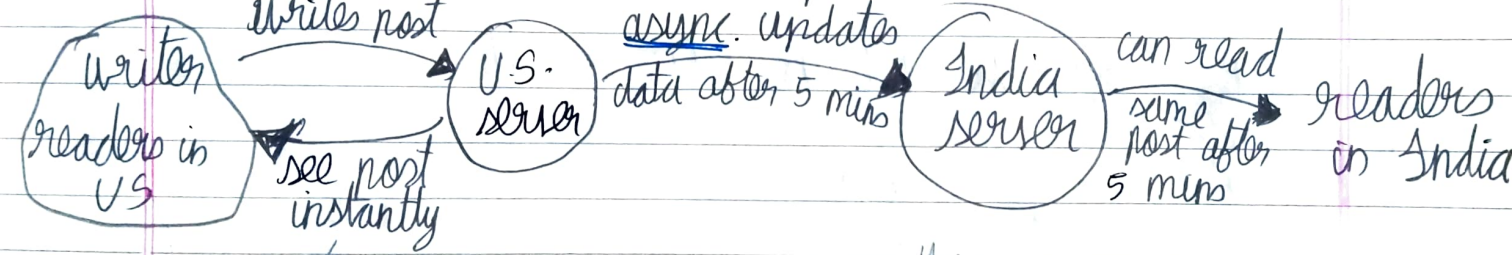
stored but time taken & dist. of server from user.

(2) To bring server closer to user, i.e. ↓ latency, we use a replica server close to user and user stores his/her writes to it. (takes lesser time as server is closer)

(3) We async. ly update data (say once/5min) present in all servers to have consistent data everywhere

(4) Used only in sys.s where we can afford the delay this async. update causes

eg:- 1 user in US writes post on LinkedIn. 1 user in India reads this post



reason → server is closer (not like the cases where server is on other side of world so you gotta make a time consuming round trip to 1st write data (writer) and then read data (reader))

NOTE: vice versa Indian writers' posts are also read by US readers 5 mins after post is published

eg:- If you don't need result of a deployment of new feature in your app to be seen all over the world instantly.

Date:
M T W T F S S

II Sharding

① It may so happen that main dB is handling way more reqs (say, 1 billion/sec) than it should be and thus, a bottleneck starts forming at main dB server

② ∴ We split up main dB data into small portⁿs of data and store em in separate small servers (known as shards) and use em to provide data to diff. kinds of reqs.

hori. scaling of a server

③ Shards a.k.a. 'data portiⁿs'
Sharding : Act of splitting a dB into 2 or more pieces (shards)
Advantage : Throughput ↑ (as no bottleneck formed)

eg:- say we got an rdB → table

- certain rows stored in shard 1 (eg:- user's name starts frm A-M)
- other rows stored in shard 2 (user's name starts frm N-Z)

bad sharding strategy as no. of user's with names frm A,B,C >> X,Y,Z

④ Popular sharding strategies

i) Based on clients' regions
ii) Based on type of data being stored
eg:- data → split → shard 1 → user data
→ shard 2 → payments data
iii) Sharding based on the hash of a column (only done for structured data)

⑤ Hotspots :- Certain shards that get more traffic than other shards cuz of the kinda data they store.

Cause → shardin key or hashin f^n is sub-optimal
Effect → uneven distri. of workload across shards

⑥ Minimizin hotspots i.e. evenly splittin up data

i) Use a hashin f^n (H.F.) that guarantees uniformity in determinin what data goes to which shard.

ii) Also, we need to ensure this H.F. doesn't change much to ensure that a particular type of data always goes to the same server

iii) Consistent hashin is kinda useful (H.F is const)

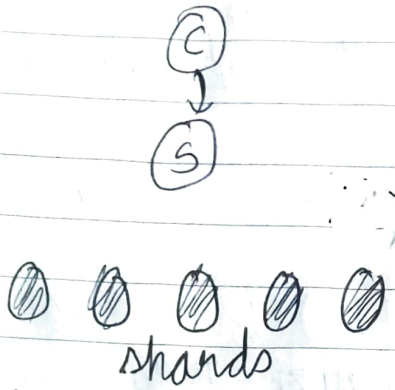
- Addin a new shard → consistent hashin minimizes the pieces of data we gotta migrate from existin shards to new shard
- Shard goes down → consistent hashin can't do anythin
  - soln :- replicate each shard.

∴ Give a lotta thought to comin up with a good logic to split up data while designin a sys.

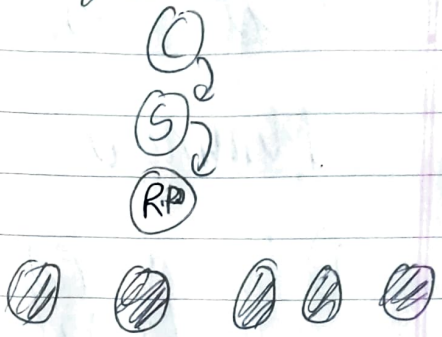
Date:
M T W T F S S

(7) Storage of shardin strategy logic

M-1 ⇒ stored in server



M-2 → stored in reverse proxy b/w server & shards



Place of storage basically decides that based on clients req which shard is it → gonna req. data frm
OR
→ write data to

```
//REVERSE PROXY - STORES LOGIC TO SPLIT INCOMING DATA (WRITE REQUEST) INTO SHARDS

const axios = require('axios');
const express = require('express');
const SHARD_ADDRESSES = ['http://localhost:3000', 'http://localhost:3001'];
const SHARD_COUNT = SHARD_ADDRESSES.length;

function getShardEndpoint(key) {
  const shardNumber = key.charCode(0) % SHARD_COUNT;
  const shardAddress = SHARD_ADDRESSES[shardNumber];
  return `${shardAddress}/${key}`;
}

app.post('/:key', (req, res) => {
  const shardEndpoint = getShardEndpoint(req.params.key);
  console.log(`Forwarding to ${shardEndpoint}`);
  axios.post(shardEndpoint, req.body).then((innerRes) => res.send());
});

app.get('/:key', (req, res) => {
  const shardEndpoint = getShardEndpoint(req.params.key);
  console.log(`Forwarding to ${shardEndpoint}`);
  axios.get(shardEndpoint).then((innerRes) => {
    if (innerRes.data === null) {     res.send('null');     return;    }
    res.send(innerRes.data);
  });
});

app.listen(8000, () => {  console.log(`Listening on port 8000`);});
```

```
//TO BASICALLY FORWARD DATA TO REVERSE PROXY
const express = require('express');
const fs = require('fs');
const app = express();
const PORT = process.env.PORT;
const DATA_DIR = process.env.DATA_DIR;
app.use(express.json());

app.post('/:key', (req, res) => {
  const { key } = req.params;
  console.log(`Storing data at key ${key}`);
  fs.writeFileSync(destinationFile, req.body.data);
  res.send();
});

app.get('/:key', (req, res) => {
  const { key } = req.params;
  console.log(`Storing data at key${key}`);
  const destinationFile = `${DATA_DIR}/${key}`;
  try {    const data = fs.readFileSync(destinationFile);    res.send(data);  }
  catch (e) {    res.send('null');  }
});

app.listen(PORT, () => {  console.log(`Listening on port ${PORT}`);});
```

replication_and_sharding — node aedb.js — 93×24

~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — node aedb.js

```
Clements-MBP:replication_and_sharding clementmihailescu$ DATA_DIR=aedb_data_0 PORT=3000 node aedb.js
Listening on port 3000!
```

replication_and_sharding — node aedb.js — 93×24

~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — node aedb.js

```
Clements-MBP:replication_and_sharding clementmihailescu$ DATA_DIR=aedb_data_1 PORT=3001 node aedb.js
Listening on port 3001!
Storing data at key a.
```

replication_and_sharding — node aedb_proxy.js — 93×24

~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — node aedb_proxy.js

```
Clements-MBP:replication_and_sharding clementmihailescu$ node aedb_proxy.js
Listening on port 8000!
Forwarding to: http://localhost:3001/a
```

replication_and_sharding — -bash — 93×24

~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — -bash

```
Clements-MBP:replication_and_sharding clementmihailescu$ curl --header 'Content-Type: application/json' --data '{"data": "This is some data."}' localhost:8000/a
Clements-MBP:replication_and_sharding clementmihailescu$ curl -w "\n" localhost:8000/a
```

replication_and_sharding — node aedb.js — 93×24

~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — node aedb.js

```
Clements-MBP:replication_and_sharding clementmihailescu$ DATA_DIR=aedb_data_0 PORT=3000 node aedb.js
Listening on port 3000!
Storing data at key b.
Retrieving data from key b.
Retrieving data from key b.
Storing data at key bar.
Storing data at key baz.
Storing data at key foobar.
Storing data at key foobaz.
```

replication_and_sharding — node aedb.js — 93×24

~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — node aedb.js

```
Clements-MBP:replication_and_sharding clementmihailescu$ DATA_DIR=aedb_data_1 PORT=3001 node aedb.js
Listening on port 3001!
Storing data at key a.
Retrieving data from key a.
Retrieving data from key a.
```

replication_and_sharding — node aedb_proxy.js — 93×24

~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — node aedb_proxy.js

```
Clements-MBP:replication_and_sharding clementmihailescu$ node aedb_proxy.js
Listening on port 8000!
Forwarding to: http://localhost:3001/a
Forwarding to: http://localhost:3001/a
Forwarding to: http://localhost:3000/b
Forwarding to: http://localhost:3000/b
Forwarding to: http://localhost:3001/a
Forwarding to: http://localhost:3000/b
Forwarding to: http://localhost:3000/bar
Forwarding to: http://localhost:3000/baz
Forwarding to: http://localhost:3000/foobar
Forwarding to: http://localhost:3000/foobaz
```

replication_and_sharding — -bash — 93×24

~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — -bash

```
Clements-MBP:replication_and_sharding clementmihailescu$ curl --header 'Content-Type: application/json' --data '{"data": "This is some data."}' localhost:8000/a
Clements-MBP:replication_and_sharding clementmihailescu$ curl -w "\n" localhost:8000/a
This is some data.
Clements-MBP:replication_and_sharding clementmihailescu$ curl --header 'Content-Type: application/json' --data '{"data": "This is some data."}' localhost:8000/b
Clements-MBP:replication_and_sharding clementmihailescu$ curl -w "\n" localhost:8000/b
This is some data.
Clements-MBP:replication_and_sharding clementmihailescu$ curl -w "\n" localhost:8000/a
This is some data.
Clements-MBP:replication_and_sharding clementmihailescu$ curl -w "\n" localhost:8000/b
This is some data.
Clements-MBP:replication_and_sharding clementmihailescu$ curl --header 'Content-Type: application/json' --data '{"data": "This is some data."}' localhost:8000/bar
Clements-MBP:replication_and_sharding clementmihailescu$ curl --header 'Content-Type: application/json' --data '{"data": "This is some data."}' localhost:8000/baz
Clements-MBP:replication_and_sharding clementmihailescu$ curl --header 'Content-Type: application/json' --data '{"data": "This is some data."}' localhost:8000/foobar
Clements-MBP:replication_and_sharding clementmihailescu$ curl --header 'Content-Type: application/json' --data '{"data": "This is some data."}' localhost:8000/foobaz
Clements-MBP:replication_and_sharding clementmihailescu$
```