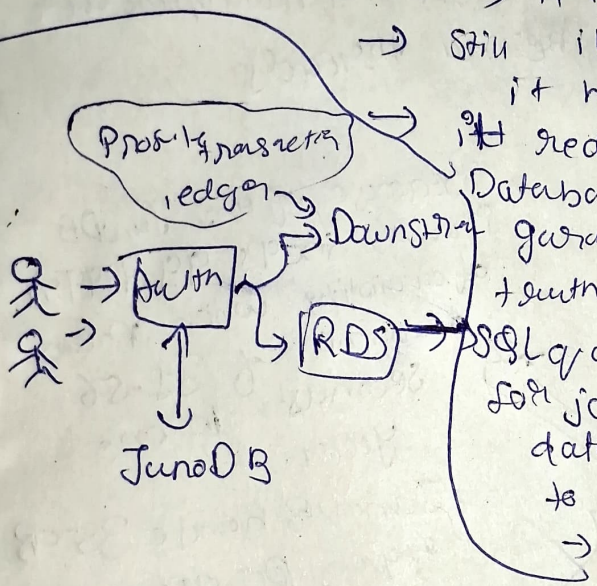
Graphite

↳ Git is not only the tool to manage code changes
↳ gregory szoc ⟹ mercurial
Best

V1 — All Basic

Juno → Distributed Key Value DB
↳ used in so key services at Paypal
↳ core backend service → auth, backend log in
↳ efficient store & cache data
→ it is not in memory all the time
→ still it can cache data for you
it has persistent storage as well
it reduces load of Relational Database because of its assets guarantees that is source of truth for successfull payments
SQL query can get very expensive for joining table and ~~order~~ rearranging data. Hence we use cache to reduce load on relational DB
→ in microservice architecture one service depend on another



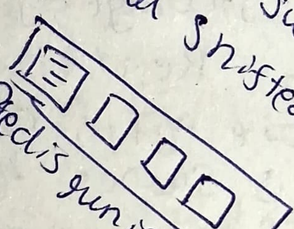
let say
Auth depend on another microservice (Downstream) - - - - Making large no. of call can make it go down ---- Juno DB can cache (But Downstream can have feature of caching) - - - - way to increase load on server - - ---- Juno DB can help in easy caching system

→ reduce help reduce SQL expensive like aggregation and joins.

redis was written in single threaded C program

Juno DB
Started in single threaded C++ program
rewrote in golang
↳ Highly efficient concurrent (Goroutines)
Goroutines are lightweight threads ⇒ fibers
→ we can create many fibers to getting done parallely
→ far more efficient than traditional posix threads which then multiplex over kernal threads.

↳ if we use single threaded code no matter how many cores are there in computer our program will execute in one CPU core of hardware. Hence this is a feature in Redis But Juno DB they do not suit this use case hence PayPal Shifted to Golang



Redis running only one core
↳ Because there selling factor is you cause your DB on single core machine
But it should have high memory/RAM
Redis is memory Bound But not CPU Bound
Juno DB is well suited for CPU Bound

Because of this JunoDB Provides six 9's 99.9999% of availability which means downtime ~ 31.56 seconds in one year.
JunoDB handle 350B request per day

Juno DB use Both → in memory & on disk cache.
↳ TTL
from few seconds to few days..

use cases
↳ user tokens → no need to hit auth on every request why to go to ~~db~~ JWT then do verify than do DB cross check why not do do cache of user details

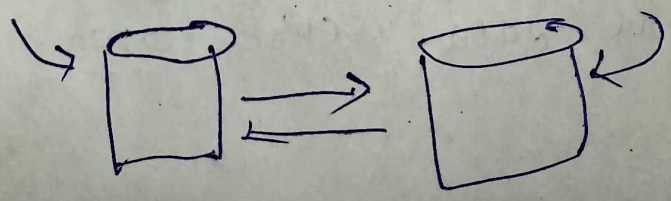
account details → no need to fetch account detail every time

API responses → no need to re compute responses every time
eg transaction list, recent activity

user preference → Notification preferences

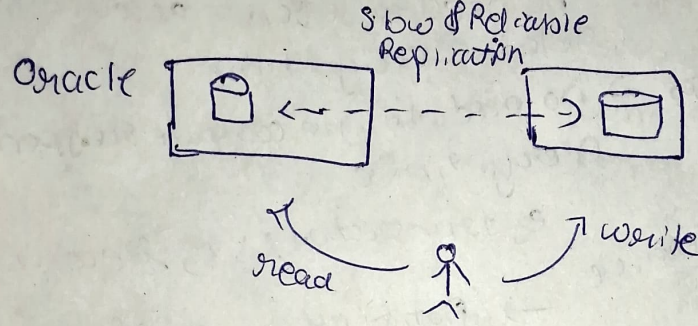
* Idempotency → avoid duplicate processing no matter how many time the request come
eg Suppose while transaction state did not got updated B that why another request and some time you done processing But you want amount should get transfer again hence no matter how many time request come you do only one processing
eg you can't drop notification twice ~~thing~~ will confuse user that payment got transfered twice

* Latency Bridging : Juno DB has really fast inter cluster replication
nearly instant Backup in another cluster
Because

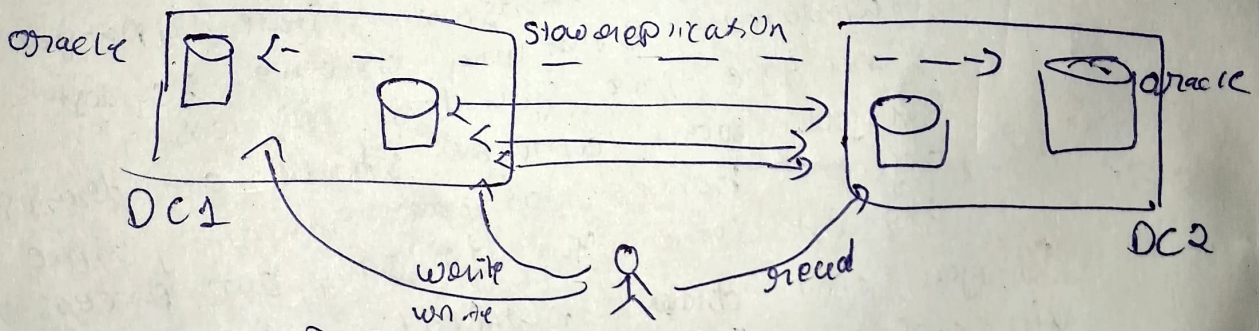


fast replication required when suppose
read go to DB cluster 1
write go to DB cluster 2 near consistent
we will have inconsistent / Broken data

PayPal uses Oracle as Primary DB
they have active active setup across datacenters



replication lag B/w 2 slow But reliable Oracle's DB is pain point so they do



Video 2 System Design

High level architecture & System Design of Storage Server

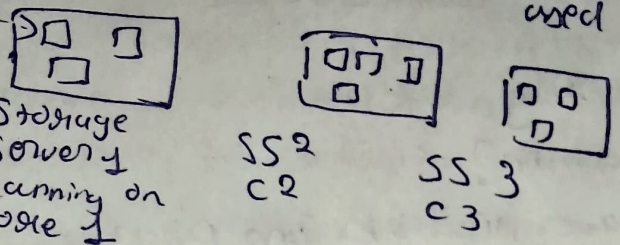
Storage Server → Inside DB -- Inside the DB the server which is used to do get put post Delete operation
Hence Juno DB is not purely implemented if they persistence - This can store data in disk or on memory

Data is split into Partition (also called Shards)

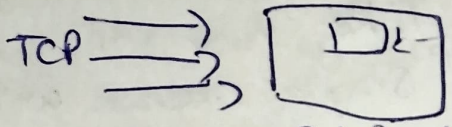
this Shards converted in → Rock DB

(very famous embeded DB used to embed data)

Shards

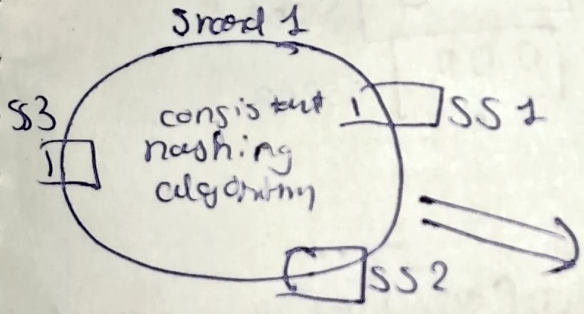


and given for get, put, post & delete operation on server



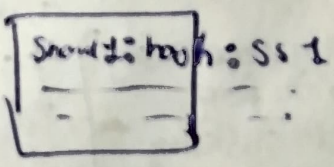
→ Request came to storage server
→ It figures out which Shard occurs it
→ It writes / read data from Shard

Data ownership to check which Shard owned By which Storage server core

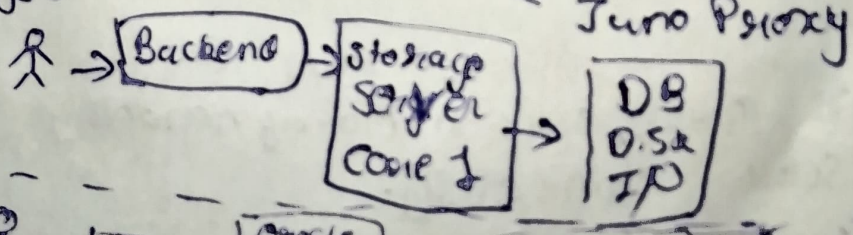


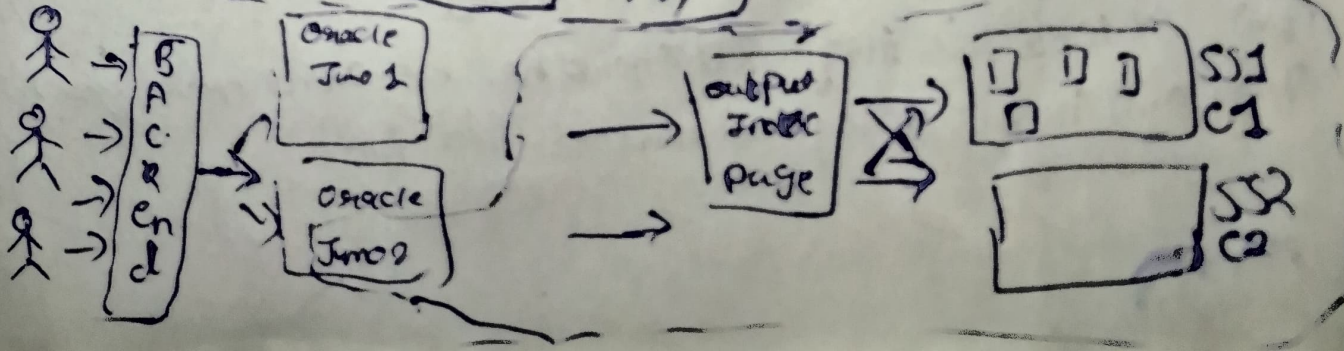
+ minimal data movement
+ linearly elastic

Order Index Page



current Redis System





Advantage

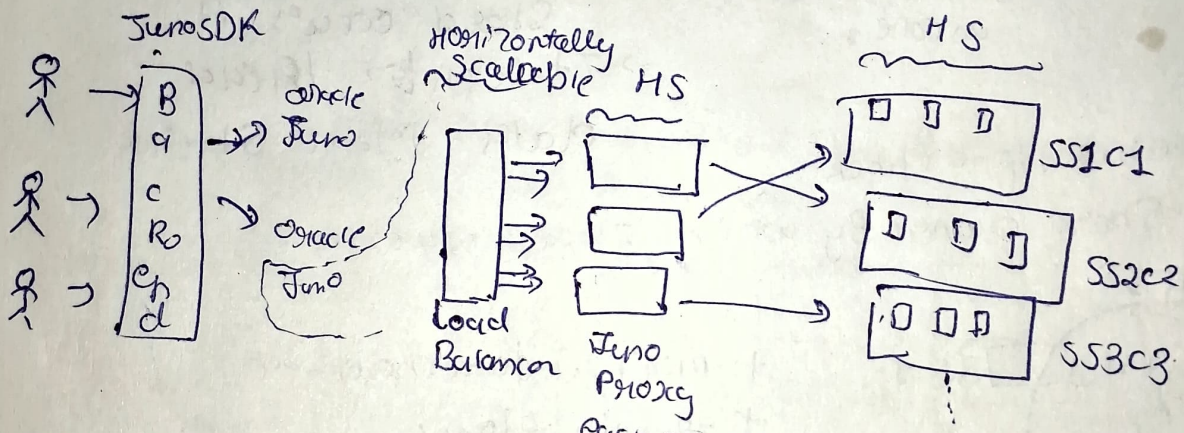
→ Storage topology is abstracted
→ client need not connect to all storage server
→ minimal config changes

is Proxy a single machine?

No juno can have multiple Juno proxy which can be put behind LB.

over all architecture look like this.

this feature is given because single machine will not be able to scale



Each Juno Proxy has copy of index page and they run ETCD [distributed store, reliable to store key value mostly critical data for distributed systems] + consistent hashing all the time for each request

Juno Proxy is Stateless, equal, equal in no. of handelling connection they can scale seamlessly.

it is Stateless ⇒ they all should be same replicas of each other and have same information all the time.

V:3 Scaling

why Juno DB needed to scale

as no. of microservice increase → network
↳ the no. of inbound connection increase

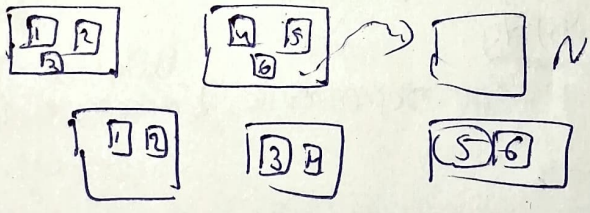
to get max performance there is a persistant connection maintaine between client to Juno Proxy & Juno proxy to storage server

an no. of microservice ↑ no. of concurrent connection ↑

↳ the amount of data to be stored increase → storage
↳ the no. of read write increase → compute

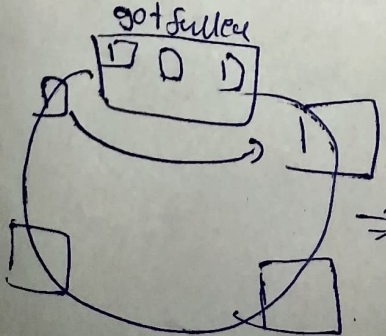
as data grows it is essential to add more storage server Juno DB fixes the number of Partition (say 1024 and never change after that) 1024 shards

this 1024 shard are distributed to other server by consistent hashing



→ this is done by consistant hashing
→ when distribution of shard done control reaches to ETCD and updates the index page, mapping

ETCD shard 1 → storage server.
↓ shard
minimal data movement
truely elastic



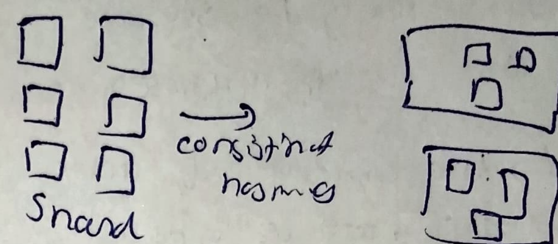
⇒ hash table → ETCD → remapping updated

Each shard has a bunch of microshare

when a shard is moved from one node to another
all micro shards are moved as a shard moved
micro shard are building Block of data

Key —f→ hash → %shard
murmur hash

because of no. of shard is fixed we can do mod shard



Total no. of shard are fixed and specified cluster creation ← Number should be large enough

these shard will be distributed across storage nodes

Paypal has 200 storage nodes to process 100 B ~~req~~ req every day

V4 99.9999% Availability

Paypal cant have poor experience because it deals with money

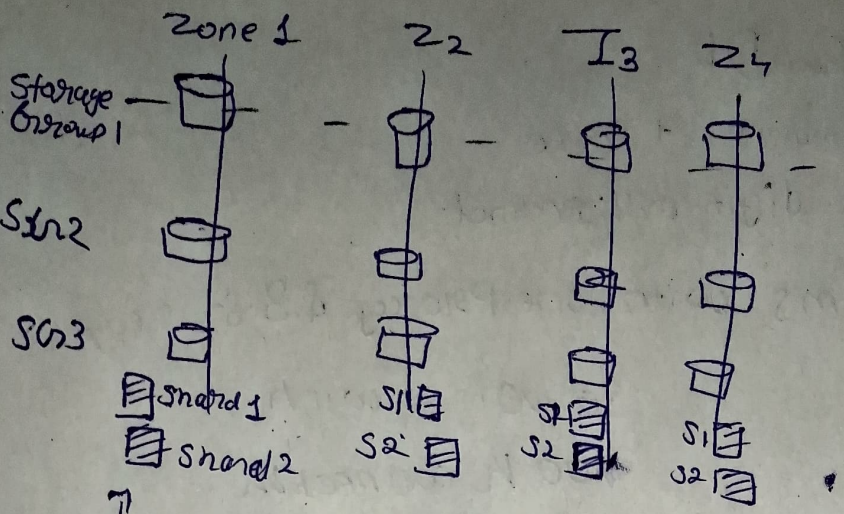
redundancy (compute + storage) + Highly available

It arranges the storage nodes in a grid

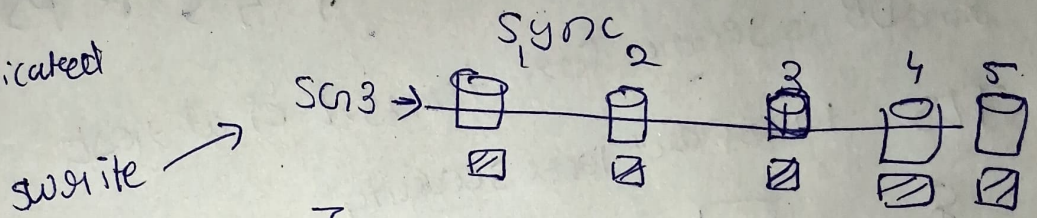
rows → Storage Groups 1 SG2

column → Zone Z1 Z2

let suppose Zone as rack 1
rack 2

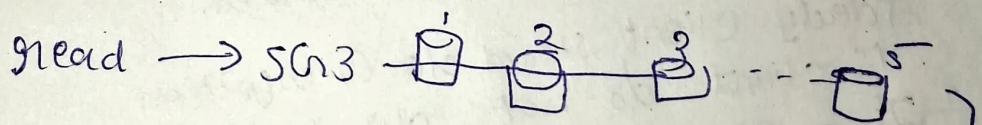


this is how Shards are replicated



Juno proxy will wait for acknowledgment from majority

write successful

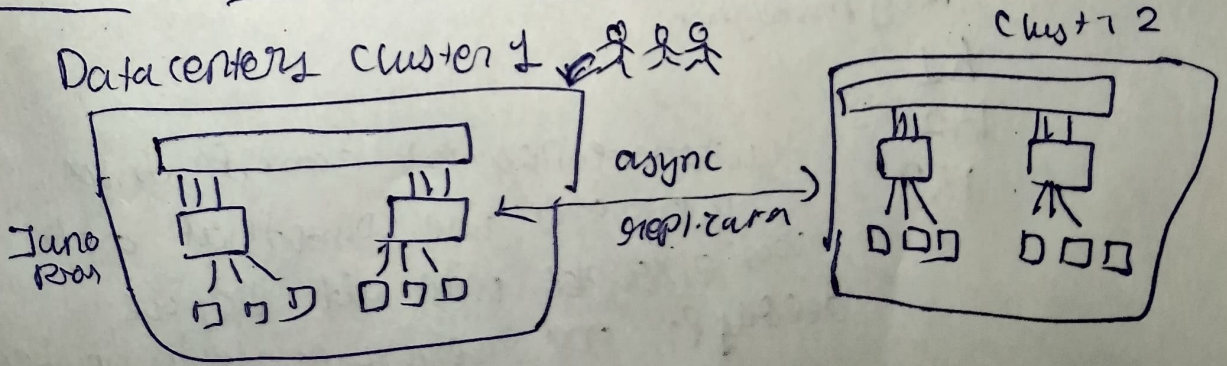


Juno proxy will wait from majority

always return latest value

because of this beautiful architecture PayPal can take whole zone for routine health check, maintainance

Now Cross Data center Replication



Performance of JunoDB

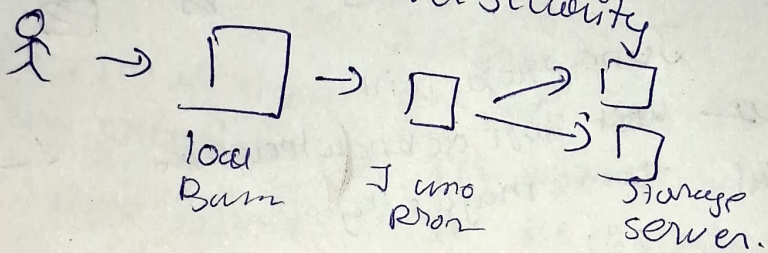
↳ performance + Scale

single digit millisecond.

P99 of 4 ms with one Proxy & 3 Storage Server with 100 K connection & 45 K RPS

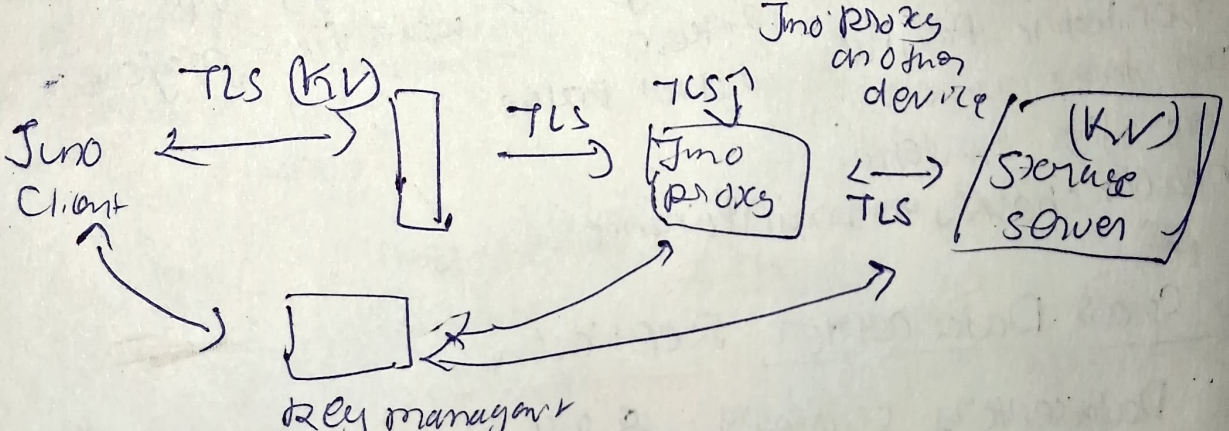
JunoDB is secure for transit & at rest

TLS transport level security



ideally client should do encryption

if client do not do encryption Juno Proxy does it



K1

K2

K3

whenever request comes for that data and we find that data was encrypted with old key we decrypt and reencrypt with newkey