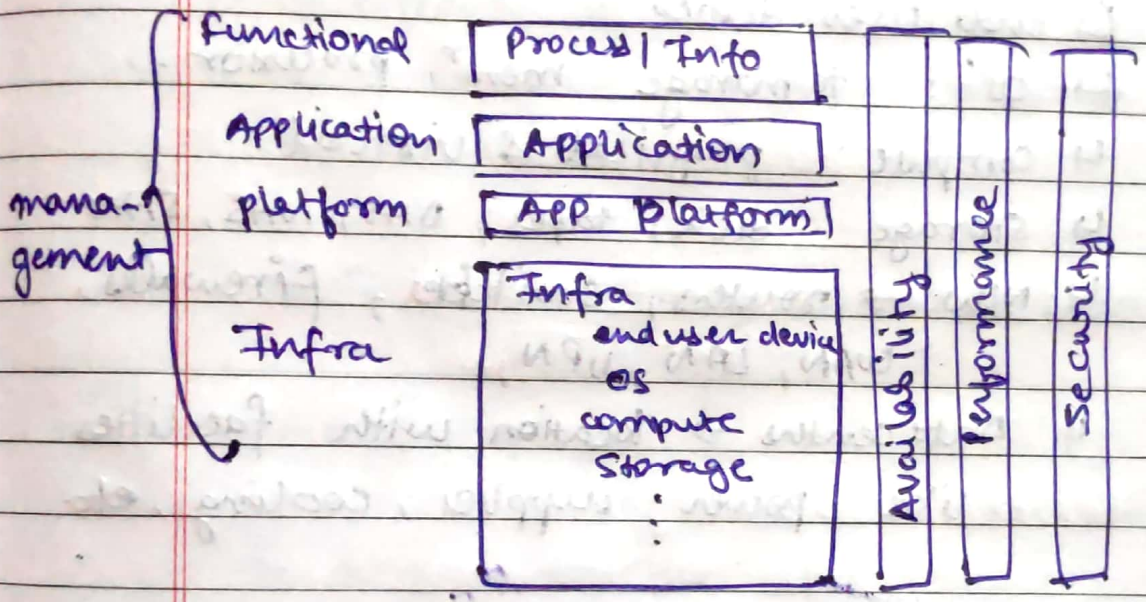
IT infrastructure

↳ All h/w, s/w, n/w facilities etc that are reqd to dev, test, deliver, monitor.

↳ includes all IT but not associated people, proces & documentation.

IT model



1) Process
   ↳ organisation specific
   ↳ create / use info

2) Application
   ↳ client → end user device
   ↳ office → emails, portals, colab tools
   ↳ Business specific → custom built (CRM)
   ERP → Enterprise Resource planning
   SCADA → Supervisory control and data acquisition.

3) Application platform
- Frontend servers → interaction
- Application servers → run actual app
- connectivity → FTP servers, ETL
- Database

4) Infrastructure
- end user device
- OS : to manage mem^r, processor ...
- compute : physical & virtual
- storage : disk, tapes, DAS, NAS, SAN
- N/w → routers, switches, firewalls, WAN, LAN, VPN,
- Data centers → location with facilities like power supplies, cooling, etc.

NFR
- Non function Requirements (attribute)
- Availability, Scalability, Reliability, Security, Performance, testability etc.
- aka quality attributes
- System may have conflicting NFR like performance v security

Availability

↳ can not be calculated or guaranteed upfront, only reported afterwards after some years.

↳ expressed as percentage of uptime in given time period.

↳ agreement is nearly 99.9%. ∴ availability of underlying IT infr must be 99.99% to ensure this.

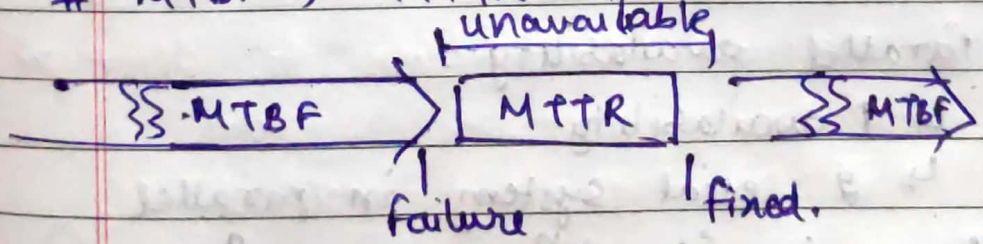
↳ 5 9s → extremely available → 99.999%

max frequency of unavailability

↳ unavailability (mins) vs frequency (year)

↳ better practice for showing stats & setting agreement.

MTBF & MTTR



MTBF → mean time b/w failures

↳ expressed in hrs

To calc → calc disk failed in year = a

↳ calc total disk tested in year

= disk × 365 × 24 hours = b

ans = $\frac{b}{a}$

↳ says abt chance of failure

MTTR → mean time to replace / repair
↳ spare parts are kept to lower this
↳ replace first then try to analyse the situation.
↳ to keep it low → automated redundancy & failover

↑ MTBF ↓ MTTR ⇒ ↑ availability

$$\text{availability} = \frac{MTBF}{MTBF + MTTR} \times 100$$

Serial availability
↳ one fails ⇒ all fails
↳ lower availability
↳ Π availability of single component
→ Power → Fan → board → CPU → NoC →

Parallel availability
↳ ↑ availability
↳ 2 serial system in parallel

$$A = 1 - (1 - A_T)^n$$

↳ assuming identical A_T
↳ important to make sure that no single point of failure exist for this

n → total systems
A_T → A of single

Sources of unavailability

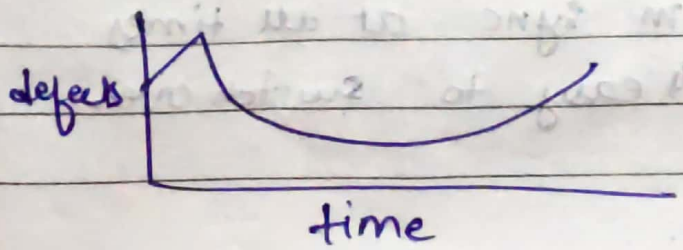
(1) ↳ Human errors
↳ can be because of technicians, users (generate huge data at backend) or system managers

errors :
↳ perform test in production env.
↳ switching off wrong component
↳ Restoring wrong backup
↳ Accidently removing files
↳ making typo in sys command
↳ avoided using standard procedures & templates.

(2) ↳ S/w bugs → impossible to create bug free system (large)

(3) ↳ Planned maintenance
↳ sys is vulnerable, mistake can be made, upgrade in cluster may create inconsistency ~~with~~ with other member

(4) ↳ Physical defects
↳ fans, disk, tapes
↳ temperature, pressure, vibration



(5) ↳ environmental issues
↳ earthquake, flood, power cut

(6) ↳ complexity of infra
↳ complicate infra → more unavailability issue instead using simple extra spare parts

Availability patterns

↳ SPOF → single point of failure

To eliminate SPOF

→ (1) Redundancy
↳ In power supply, n/w, SAN NBA

2) Failover
↳ (Semi) automatic switchover to stand by system (component)

(3) Fallback
↳ manual switchover to identical standby computer system
↳ disaster recovery method.

Types
↳ hot site → fully configured.
↳ h/w, s/w, data etc must be in sync at all times
↳ easy to switch over

↳ warm site
  ↳ infra present, data & application must be configured
  ↳ take a day to shift

↳ cold site
  ↳ just place with cooling $, ventilation etc.
  ↳ PCs, n/w etc + data & app must be brought in & configured.

4) Business continuity (BC) ~~management~~
↳ IT availability not guaranteed ∴ response must be provided in case.
↳ identifies threats
(i) BCM (~~mag~~ management)
  ↳ IT + people + work + processes
  ↳ measures to be taken when critical incident occurs to continue critical opn running

2) DRP (disaster recovery planning)
  ↳ measures to take in case of disaster
  ↳ fallout method.
  ↳ CERT decides how to hande crisis
    ↳ computer Emergency response team
      ↳ team of senior managers

RTO & RPO

↳ RTO → recovery time objective
  ↳ time (max) within which a business process must be restored after a disaster.
↳ failover + ~~fallout~~ fallback

↳ RPO → Recovery point objective
  ↳ point in time to which data must be recovered
  ↳ amt of data loss → willingly accept
  ↳ backup techniques.

Perceived performance

↳ how quickly system appears to perform
↳ may be different than actual
↳ like sys performs bad once a week
  ⇒ poor performans for people

ways to increase this
↳ improve actual
↳ use of splash screen, progress bar

designing for performance

↳ sys must meet performance reqm even during increasing load.
↳ must meet & when parts have failed, sys under maintenance,

during backup etc.

ways to calculate performance in design phase

(1) Benchmarking
- used to assess the performance characteristics of comp h/w
  eg: FLOPS, MIPS
- provide method to compare across subsystem
- raw performance measure

(2) vendor experience
- provide tools, configuration & best practices

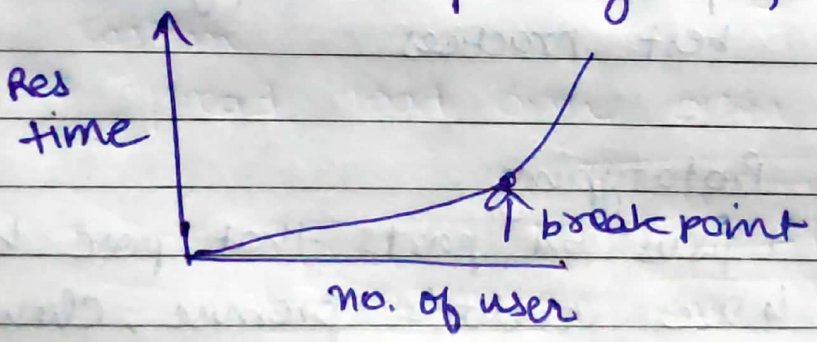
(3) Prototyping
- focus on parts that pose high risk.
- use vendor's premise, Cloud computing
- use POC

(4) User profiling
- predict load of a new s/w in sys
- create performance test scripts that put representative load.

Performance of running systems

(1) Bottlenecks
- ↳ performance = Perf (all components)
- ↳ 1 component limits total perf
- ↳ no bottleneck = impossible

(2) Performance testing
- (a) load testing → normal circumstance
- (b) stress ↳ → extreme load ↳
  - ↳ finds threshold
- (c) endurance ↳ → normal load for large time
  - ↳ issue arise in memʳ leaks. expanding dB, disks



Perf testing uses:
(1) injectors → servers emulating users
(2) test conductor → ↳ coordinate tasks, gather metrics & collecting perf data.

perf testing should be done in prod-like env.

Date / /
Page No. 12

Perf patterns
↳ diff ways to increase perf : caching, scaling, load balancing, high perf cluster, grid, designing for perf, capacity management.

◎ perf issue in upper layers
↳ 80% perf issue bcos of badly behaving app.,
↳ ~~first~~ optimise upper layer
↳ prioritise task, work from memr, good use of queues & schedulers

(1) caching → general defn
↳ disk caching → disk drives contain cache memr, look ahead caching
↳ web proxies → store earlier ~~accessed~~ fetched data

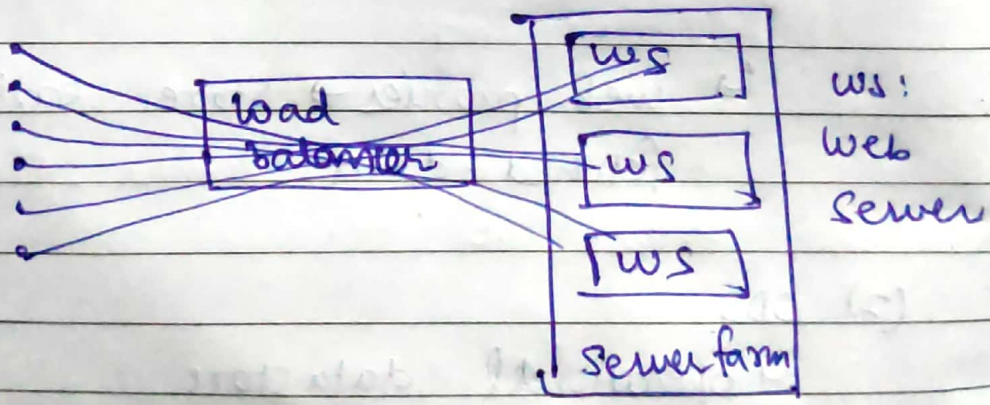
(2) ODS
↳ operational data store
↳ small read only replica of main DB

(3) frontend servers : Reverse proxy
↳ store most freq web pages on frontend servers

4) In memory database
↳ run entire db from memr.

(5) Scalability
↳ 2ways
↳ vertical → add more resource to single components
↳ limitations
↳ horizontal → add more component to infra
↳ more complex to manage

(6) load balancing
↳ for optimal use of horizontal scaled system
↳ spread load over other machines



load balancer
↳ checks load, availability of other components
↳ servers in farm must be identical

app must be stateless to work

(7) High perf cluster (HPC)
- pool of computing power
- all sys must be doing useful work at all times, ↓ wasting resource, ↓ comm^n.

(8) Grid
- HPC but sys are geographically separated
- bottleneck → limited b.w.
- security imp., PC's must be protected and data must be protected.

(9) Design for use:
- know what sys will be used for
- follow vendors recommended implem^n
- spread load over available time
- move rarely used data
- use special products like (pt 4), Real time OS etc.

(10) Capacity management
- perf of sys is monitored
- perf issues can be predicted.