- Bandwidth
    - Ingress / Egress
- Any data eviction policy?
- Any spike in traffic?
- Storage
    - distributed file storage systems: hdfs, glusterfs
    - any data eviction policy?
    - Non volatile storage
- Cache
    - any cache eviction policy? LRU?
- Load Balancer -> SSL termination
- Data replication
- Monitoring
- Security
- Any Machine Learning pattern is usable?
- Lambda Architecture
- CDN
- Datacenter as a service
- 429 - too many requests
- Race condition
- Rate limiter: fixed, rolling
- UUID : Universally unique ID
- Session persistance
- Stale data
- Domino effect - we want the replicas to Have independent failure probability to Have uncorrelated failure

- !!! Data - communication - computation
- What about utilization?
- Quorum for read / write?
- For replication; replicated state machine (read this, write that); state transfer (transfer all database). state transfer more robust. RSM has a problem with parallelism.
- What is output rule for system design, mentioned in VMware paper?
- What is Split brain disaster?
- What is test and set server?
- Automated canary analyses?

- Autoscaling? for stateless and stateful machines? Immutable machine images?
- Consistent Naming?
- ELB Config?
- Healthcheck?
- Squeeze testing
- Staged, red/black deployments
- timeouts / retries / fallbacks
- circuit breakers, fallbacks, chaos
- multi-region failover
- failure-driven design
- Write back, write ahead caches???
- Serializability : concurrency control
- Linearizability : consistency
- Concurrency control : Pessimistic -> two phase locking, optimistic -> ?
- Web sockets - full duplex - allows communication in both direction - unlike half-duplex, allow this to happen simultanously.

- We begin with the problem statement, gather requirements, and iterate through designs that become increasingly sophisticated until we reach a viable solution. Ultimately, we arrive at a system that defends against many failure modes and satisfies both the initial requirements and additional details that emerged as we iterated.
- I will try to split my system into services that each one do one nothing and do it well.
    - Gives different parts of my system isolation from each other.
    - Makes it easier to troubleshoot problems.
    - Makes it easier for the system to evolve over time.
    - Simplifies performance analysis. No need to think about different traffic mixes.
- I will try to keep the state in specific components and keep the rest of the system stateless.
    - Stateless components easy to manage, because they are trivial to scale horizontally.
- Flow
    - First design for single DC, then multiple DC
    - Assumption, Constraints, SLOs

        - Authentication and Authorization already taken care of? So all users are already authenticated and trusted clients?
        - API
        - Data (Critical / Non-critical)
        - Do some calculations
            - Disk (1 TB SSD / HDD)
            - RAM (128 GB)
            - Bandwidth (10 Gbps -> 1 GBps)
            - CPU (32 cores)
            - Datacenters are connected via 100Gbps?
            - Assume the probability of a machine failing some number of times?
                - 0 -> 40 %
                - 1 -> 40 %
                - 2 -> 15 %
                - 3 -> 4 %
                - >3 -> 1%
        - Iterate & Refine (If I am unhappy with any aspect)
            - Is it feasible and does that work correctly?
            - Is it reliable?
            - What is the scariest thing that can happen to my system? And how my system defend against it?
                - Loss of a datacenter
                - Loss of important state / data
                - Load spikes
            - Efficient use of hardware? Utilization?
            - Are there places where I can improve by adding caching, indexing, lookup filters like bloom filters.
        - Monitoring
            - Detect outages (and alert) - Auto scalers, commodity computing
            - Give indications of growth and system capacity to meet future needs
            - Assist in troubleshooting outages and problems
    - Consistency
        - FaxtPaxos : Optimized consensus algorithm that uses a stable leader process to improve performance. Performance for FP is

    one round-trip between the leader and quorum of closest replicas to commit an update or a consistent read.

- Sometimes keys are not evenly distributed. It is always best to shard based on a hash of the key.
    - M shards over N servers. M >= 100 * N
    - shard = hash(key) % M
    - we have a map of shard -> server
- Sharded and replicated datastores should have automatic mechanisms to cross-check state with other replicas and load lost states from the peers. Maybe after a downtime or when new replicas are added. This needs to be rate-limited somehow avoid thundering-herd problems where many replicas are out of sync.

External consistency
At most one semantics

- Technical (expertise in features, systems, processes; technical discussions with the engineers)
- Ownership (scope of projects, ability to deliver projects, coordination)
- Business Insight (understanding team delivery; setting priorities for business value, vision)
- Continuous Improvement (mentorship and coaching, championing changes, well-being of the team)
- Leadership (communication across organisation, building communities, helping recruiting processes))

- Mainframe hash çalışması
- Google Hashcode hub world championship
- Slack group - data structures & algorithms
- son 5 dakika için ek sorular
- LinkedIn Common Questions - evernote