**Performance**

* **How fast a response responds**
* **It is measured relative to** 
  + **workload (request volume or data volume)**
  + **hardware (capacity)**

**What is performance problem?**

* **Queue of request building somewhere**
  + **Network socket queue**
  + **OS run queue**
  + **Db IO queue**

**Causes of performance issues**

* **Slow processing of request compared to what resource capacity allows (UNDERPERFORMANCE)**
* **Resource capacity limited(resources are not enough compared to work load but system is using all it got)**
* **Serial resource access (lacking concurrent or parallel processing)**

**Performance Principles**

* **Efficiency**
  + **Efficient use of resources (CPU and memory)**
  + **Efficient algorithms**
  + **Efficient data storage**
* **Concurrency**
  + **Hardware should allow for concurrency**
  + **Software needs to be developed to run concurrency**
* **Capacity**

**Object of a system to perform better?**

* **Minimize request response latency**
* **Maximize throughput**

**Latency**

* **Time taken to respond to a request**
* **Latency = wait time for request to start processing + Processing time**

**Throughout**

* **Rate of request processing. How much we can process in a second**
* **Depends on** 
  + **Latency (minimize it)**
  + **CAPACITY (maximize it)**

**Performance Metrics**

* **Throughout (maximize). More users supported**
* **Latency (minimize). Relevant for live systems.**
* **Resource utilization – CPU and memory** 
  + **low means wasted resource,**
  + **high means saturation,**
  + **ideal is middle**
* **Errors (ideally None)**

**Tail Latency**

* **Slowest response time of a system**
* **It is indication of queuing of requests (requests are forced to wait before processing even starts)**

**Network Latency**

* **Data Transfer: latency when data is moved over in wires**
* **TCP connection: requests sent for connection to server and acknowledgements by server take time**
* **SSL/TLS connection: sits on top of TCP to secure the data transfer. Overhead for connection adds to latency**

**Minimizing Network Latency**

* **Data Transfer:** 
  + **Reduce size of data with compression. Compression may increase overhead**
  + **cache information to reduce data transfer**
  + **Use binary formatted data to transfer. Google grpc protocol has less latency due to binary format**
* **Connections: Connections between server and browser can be saved in a pool for future use instead of creating new connections. Same for ssl connnections where server can save information exchanged when last connection was setup.**

**Memory Latency**

* **Finite Heap memory – sits on RAM**
  + **Condition where heap memory is filled**
  + **Program specific**
  + **Garbage collectors act to reduce heap memory but it ends up slowing application adding memory latency**
* **Large Heap memory**
  + **Condition where heap memory is large with more objects**
  + **Garbage collector scans more objects and algorithms runs slow due to sheer size of heap**
* **Database Buffer memory**
  + **Frequently accessed info is cached in buffer memory**
  + **Sits on Ram**
* **Garbage collector algorithms**
  + **Some of them are slower than others**
  + **Generational Garbage collector**

**Minimizing memory latency**

* **Avoid memory bloat**
  + **Memory bloat: excessive memory use due to inefficient allocation, memory leaks, aggressive caching(without eviction policy)**
* **Weak/Soft reference**
  + **Memory allocations that can be cleaned by garbage collector if memory is close to full**
  + **Programmer defines references to be weak in Java**
* **Smaller processes**
  + **Break batch processes onto smaller processes with worker nodes. Each process takes less memory, and quicker GC cycles.**
* **Pick good Garbage collector** 
  + **G1 garbage collector divides heap into smaller regions (young objects, old objects, free space) and focuses collection region wise. It is fast as collection localized to region**
  + **Generation garbage collector: divides collections into y**
    - **Young objects: newly created objects are often frequently looked upon**
    - **Old objects: they have survived collection attempts so they are important and hence less frequently accessed for collection**
* **Normalization of databases**
  + **Reduces duplication and hence less memory and less latency**
* **Compute Over Cost**
  + **Compute info on top of raw data instead of storing computation in memory**

**Disk Latency**

* **Faced where disk is accessed.**
* **Hard Disk Drive has moving parts that delay data access**
  + **Database query is sent to disk if not stored in RAM, leading to disk latency**
  + **Serving web content files like CSS are accessed from disk**

**Minimizing Disk Latency**

* **Sequential IO** 
  + **IO in an order from start to end. Like reading a book**
  + **Faster than random io because HDD head moves in one direction**
  + **Random io: HDD head jumps around on disk in diff directions leading to time waste**
* **Batch IO**
  + **Write or read many files all at once**
  + **Reduces context switching between io and other tasks**
* **Asynchronous Logging**
  + **Delicate IO work to child process**
  + **Reduces load on parent process**
* **Cache**
  + **Store web content like css in cache. Reverse proxy(nginx)**
* **Zero Copy**
  + **Copy data from disk to kernel and then to network**
  + **Eliminates copying data to user space before being sent over network**
* **Schema Optimization**
  + **Denormalization of databases**
    - **Keep all data in one table instead of 10 tables**
    - **Reduces jumping around tables to find information and hence reduces disk latency**
    - **But increases memory latency**
  + **Indexing**
    - **Instead of full table scan, map values in XYZ column to row id where they are found**
    - **Column id has values [“hari”, “alex”, “xay”]. We can map “hari” id to row 1 which means information about “hari” is in row 1**
    - **Reduces disk latency from millions of scans to few.**
* **Raids (Redundant Array Of Independent Disks)**
  + **several physical disks are grouped under a logical unit**
  + **Parallelize data access across disks**
    - **Faster performance**
    - **Fault tolerant**

**CPU Latency**

* **Inefficient algorithms**
* **Context Switching**
  + **Switch time between process 1 and process 2**
  + **As process 1 is pending, its state is saved in memory. Process 2 is loaded into CPU from memory**
  + **This loading onto CPU and eviction from CPU takes time.**

**Minimizing CPU latency**

* **Batch/Async IO** 
  + **reduces context switching because we process all at once**
  + **Use separate thread with async io for logging to reduce load on main thread**
* **Single Threaded Model**
  + **One main thread so no context switch**
  + **Main thread delegates tasks to worker threads who run asynch**
* **Thread Pool Size: keep small**
* **Processes in dedicated Virtual Environment**
  + **Run each process in their own environment with fixed CPU/memory**
  + **Prevents processes exhausting CPU resource for other processes**