System Design Techniques

[Srinivas and Keshav]
[Butler Lampson's paper]

System Design

- The art and science of integrating (distributed) resources into a harmonious whole.
- Not a clear cut science
- A lot depends on good judgment and experience
  - Cannot easily quantify simplicity, scalability, modularity, usability, extensibility, elegance.
  - Yet tradeoffs are necessary among these
- But we can identify some general principles of good design.

Design Space = Performance Metrics + Resource Constraints

- Design space is constrained by performance metrics and resource constraints.

- Performance metrics specify performance targets of the system being designed
  - Throughput, delay, cost, development time, failure rate
  - One metric may affect another
  - E.g. lower cost may increase failure rate

- Resource constraints specify limits on available resources
  - Some resources are more constrained than others
  - E.g. computational power vs. I/O bandwidth
  - Former is unconstrained (almost!), while latter is constrained
  - Trade unconstrained resources for constrained ones to maximize the utility.
  - E.g. use computational power to compress data so that less bandwidth is required.

- A metric in one situation could be a constraint in another.
  - Example: Latency as a metric or resource constraint
  - Metric = Latency of a stock trading transaction should be no more than 10ms.
  - Resource constraint = Minimum packet latency of a satellite link is 250ms.

Common metrics/resources

- Time
  - Latency, development time, mean time between failures, clock cycles
- Space
  - Memory, storage, packet size, task size (instructions, lines of code)
- Money
  - Project cost
- Labor
  - Person hours of work
- Bandwidth
  - Space per unit time (space/time)
  - bits/second, bytes/second, instructions/second, lines of code written per hour, hourly cost
- Social constraints
  - Standards (compatibility), market requirements
- Scaling
  - Number of machines in a cluster, number of users supported, amount of data processed

Balanced Systems

- Bottleneck resource
  - One which is most constrained and hence limits overall performance

- System performance improves only if we devote additional resources to the bottleneck.

- Conversely, decreasing the unconstrained resource lowers cost without reducing performance.

- Balanced system
  - All resources are equally constrained, while meeting performance targets.
  - No more or no less than needed.

- Henry Ford's Model T
  - A balanced cheap car!
  - Ideally no part of the car outlives any other part.

Common design techniques

- Multiplexing
  - Time vs. space and money
- Pipelining and Parallelism
  - Compute units vs. time
- Batching
  - Response time vs. Throughput
- Exploiting Locality
  - Space vs. time
- Speedup the common case
- Hierarchy
  - Scaling
- Binding and Indirection
- Virtualization
- Randomization
- Soft State
- Explicit State Exchange
- Hysteresis
- Seperating Control and Data
- Extensibility

Multiplexing

Trading time for space and money

- Sharing one resource among many users
- E.g.
    - Teller at a bank : Space over waiting time
    - Long Distance Trunks : Bandwidth over queuing delay.
    - One CPU shared by multiple processes: Space and money for waiting time

- Temporal vs. spatial multiplexing
    - Temporal = Share time (e.g. CPU or network link)
    - Spatial = Share space (e.g. memory)

- Server (scheduler) controls access to the resource
    - Boarding the plane
    - CPU scheduler
    - Network link scheduling
    - Memory manager
    - Office hours

- Multiplexing might virtualize the shared physical resource.

- Statistical Multiplexing
    - Overcommitting (overbooking) a resource given some probability that not all allocations are fully utilized
    - Doctor's appointment schedule
    - Airplane seats
    - Virtual memory

Pipelining and Parallelism

Trading computation for time

- Parallelism
  - Use N processors for N independent sub tasks
- Pipelining
  - Use N stages for serially dependent tasks
- E.g. used extensively in data forwarding path of routers.
- Linear speedup: if throughput increases by a factor of N for N compute units. Smaller otherwise.
- In both cases, speedup limited by the slowest processor or stage.

Batching

Trading response time for throughput

- Accumulate a number of tasks, then execute.
- Effective when
  1. Task overhead increases sub-linearly with number of tasks
  2. Accumulation time is not significant
- Example:
  - Interrupt coalescing in network adaptors
  - Character batching in remote login sessions

Exploiting Locality

Trading space for time

- Also called caching

- Spatial vs. temporal locality

- Examples
  - Instruction and data caches
  - Web caches
  - Route lookup
  - File system buffering
  - Virtual Memory Paging

Optimizing the common case

- The 80/20 rule
  - 80% of time is spent in 20% of code
- Challenge: How to identify the 20%?
  - Instrument and measure
- Once you do, optimize the heck out of 20%
- Examples
  - RISC machines
  - Router data path : Process common case in hardware.

Hierarchy, Binding, Indirection

- Hierarchy
  - Common technique to scale
  - Loose vs. strict hierarchy
    - E.g. Local ISPs may directly connect to each other
- Binding
  - Mapping from abstraction to specifics
- Indirection
  - Reading the binding translation from a well known location
- Examples
  - Machine name ==> IP address
  - Alias ==> Email address
  - Virtual memory: Virtual page # ==> Physical page #
  - Mobile communication: Phone number ==> device

Virtualization, Randomization

- Virtualization
  - Combines multiplexing and indirection
  - E.g. Names of call center reps., CPU sharing, Virtual memory, Virtual Machines, VPNs, VONs, Web hosting.

- Randomization
  - To break a tie without knowing number of contenders.
  - E.g. CSMA/CD, routing (??), multicast NACK implosion.

Soft State

- Hard state
  - once installed, needs to be explicitly removed
  - Complicates recovery upon failure
- Soft state
  - State removed unless its periodically refreshed
  - Trade bandwidth and computation for robustness and simplicity
  - Challenge: How to choose deletion time?

Hysteresis

- To prevent rapid oscillation of a threshold value.
  - E.g. Thermostat: With a single temperature threshold, the heating system would rapidly turn on or off.

- Solution: Make the threshold state-dependent
- Example:
  - 0.1 threshold in state A and –0.1 threshold in state B.
  - So value must change at least 0.2 for state change.

- E.g.
  - Memory: Start swapping pages when memory usage is above 90% and stop swapping when usage falls below 80%.
  - Network connection handover between base stations

- Lesson: Whenever you must design a threshold for some event, consider using two (or more) thresholds instead of one.

Separating Data and Control, Extensibility

- Data vs. Control
  - Separate one-time actions vs. repetitive ones
  - Pros: Helps make the data plane fast.
  - Cons: More state needed in the network
  - E.g. connection establishment vs. data forwarding in Virtual Circuit networks
    - Packets only carry VCI. Control plane is separate.
    - How about datagram networks (IP)?
- Extensibility
  - Allow hooks for future growth
  - E.g. IP version field, HTTP version field, data rate exchange among modems, kernel modules.

Summary

- A repertoire of techniques to apply in different situations.
- Good judgement and experience is important
- Use a good idea more than once, but only when appropriate.
- Like a system designer’s toolbox
    - Not all tools may be applicable or appropriate in all situations.