System Design Techniques

[Srinivas and Keshav]
[Butler Lampson’s paper]
System Design

- The art and science of putting together (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.
Performance Metrics and Constraints

- Some resources are more constrained than others
  - E.g. computational power vs. I/O bandwidth
  - Former is unconstrained (almost!), while latter is constrained

- Performance metric measures some aspect of system performance
  - Throughput, delay, cost, development time, failure rate

- Design space defined by performance metrics and resource constraints.

- Trade unconstrained resources for constrained ones to maximize the utility.
  - E.g. use computational power to compress data so that less bandwidth is required.
Common resources

- **Time**
  - Latency, development time, mean time between failures
- **Space**
  - Memory, bandwidth (?)
- **Computation**
  - Less of an issue these days
- **Money**
- **Labor**
- **Social constraints**
  - Standards, market requirements
- **Scaling**
  - Design constraint rather than resource
Balanced Systems

- Bottleneck resource
  - One which is most constrained

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

- Conversely, decreasing the unconstrained resource does not impact system performance.

- Balanced system: All resources are equally constrained.

- Henry Ford’s Model T
  - A balanced car! No part 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
- Separating Control and Data
- Extensibility
Multiplexing

Trading time for space and money

- Sharing single resource among many users
- E.g.
  - Teller at a bank: Space over waiting time
  - Long Distance Trunks: Space (capacity) over queuing delay.

- Multiplexing virtualizes the shared physical resource.

- Server controls access to the resource
  - Boarding the plane
  - Link scheduling

- Statistical Multiplexing
  - Overcommitting a given some probability that not all allocations are fully utilized
  - Temporal vs. spatial
    - Doctor's appointment schedule
    - Airplane seats
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
  o Combines multiplexing and indirection
  o E.g. Names of call center reps., CPU sharing, Virtual memory, Virtual Machines, VPNs, VONs, Web hosting.

● Randomization
  o To break a tie without knowing number of contenders.
  o 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

- Hysteresis
  - To prevent rapid oscillation of a value around a threshold.
  - Soln: Make threshold state-dependent
  - E.g. 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. Handover between base stations
Separating Data and Control, Extensibility

● Data vs. Control
  o Separate one-time actions vs. repetitive ones
  o Pros: Helps make the data plane fast.
  o Cons: More state needed in the network
  o 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
  o Allow hooks for future growth
  o 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.

- Not all may be applicable or appropriate.

- Use a good idea more than once, but only when appropriate.