**Advanced Topics in Computer Systems, CS262B**
**Prof Eric A. Brewer**

# Lessons from Giant-Scale Services

**February 19, 2004**

Experience paper on how to build and operate very large Internet sites...

Key ideas:

- Load management
- Partitioning vs. Replication, load redirection
- Availability metrics: yield and harvest, MTTR emphasis
- Online evolution
- Graceful degradation
- the DQ principle

Basics:

- Data center: power, AC, networking, no people, minimal cables
- Extreme symmetry simplifies everything
- FRU = “field replaceable unit” equals whole node (worry about subpieces offline)
- management backplane is a good idea for staging, management under duress
- data center contract limits temperature, power, networking variations

Load Management

- key idea: need a **highly available *name***
- DNS: map names to IP addresses
  - remap when IP address disappears
  - mediocre solution: DNS expiration takes a while, not all browsers obey expiration correctly
- L4 switches
  - forward incoming TCP connections to the “up” (IP, port) addresses
  - load balance based on number of open connections
  - manage the “up” set automatically (by detecting dropped connections/resets)
  - (but better to be proactive! why?)
  - come in pairs with automatic hot fail-over (avoid single point of failure)
  - L7 switches switch based on URLs
- smart clients: clients manages the name mapping and failover directly
  - this is the best solution, as it handles disaster recovery well (redirect to new data center)

- not built into HTTP, but can do it if the client is an applet or program

Availability Metrics

- focus on MTTR not MTBF -- faster debugging cycle, more stable
- yield = better version of uptime
- harvest reflects potential for incomplete data used for answers
- good designer plans how faults affect MTTR, yield and harvest!

DQ Principle

- claim: data per query * queries/sec == constant
- represents the total data flowing through the system (MB/s like bandwidth)
- only true is system is running near capacity, otherwise you can increase D or Q
- Lots of uses: spec hardware, eval software changes, capacity planning, failover planning, etc.

Replication vs. Partitioning

- replication typically viewed as "better": maintains 100% harvest during a fault, but 50% yield (if it was at capacity!)
- ...but partition maintains 100% yield, 50% harvest
- There is no "better" -- just different optimizations
- Load redirection problem: not enough to replicate data, must have replicated **capacity** (sufficient DQ points)
- See Table 1 for the overload factors\
- For write-intensive traffic replication costs more than partitioning, but for read mostly they are essentially the same!
- => partition until you reach a convenient size, and then replicate the whole set. AOL Caches partition within a rack (used to be 5 machines), and then replicate racks for capacity
- Can replicate only the important data, and then ensure that lost harvest does not include the important stuff
- Randomization makes worst case data loss same as the average case.

Graceful Degradation

- large sites on open networks will get overloaded
  - Schwab uses managers to handle the overload of phone calls during a market event
- correlated failures, although rare, can reduce capacity and cause overload
- key insight: to handle overload we can either limit Q (admission control), or reduce D

(to increase Q) to handle more capacity at some loss in quality

- best answer is ususally a combination! deny the expensive queries and increase caching and reduce harvest
  - cost-based admission control
  - priority or value-based AC (for good customers or financially important transactions)

Disaster Tolerance

- key idea: estimate correlated failures due to a disaster, typically one whole data center
- figure out replication/partitioning for that set of failures
- need load redirection to *outside* the data center => can't use L4 switch (both of them fail as well); ideally smart clients, else DNS
- plan on overload due to redirected load, and handle via graceful degradation!

Online evolution

- quality: standards go down under fast evolution! also lower for online services (vs normal software) why?
- staging: extra space to store two versions of the software (or data) makes it easy to switch back and forth; important to automate "revert"!
- three ways to upgrade
  - fast upgrade
  - rolling upgrade
  - big flip

Moving sites!

- Inktomi moved the data center twice while it was online!
- Many upgrades to hardware, OS, schema, protocols, ...