Perspectives from Operating a Large Scale Website

Dennis Lee

Amazon.com

Some General Lessons

- Decouple – allow for independent evolution
  - Key Question: can I deploy the pieces independently without breaking the “whole system”
  - Avoid structures that have to be the same across different systems
    - Pass messages not objects;
    - Avoid shared schemas across different databases – database engines don’t deal well with exceptions and schema changes forces downtime
  - Run-time versioning is the first step in the road to hell
    - Key symptom: running multiple instances of service to support different API’s
    - Better: one instance of service supporting multiple versions of API
  - Always prefer many small things to a few big things especially if you don’t fully control the big things.
    - e.g., databases, multicast, code-base
- Test, test, test - insist on Unit Tests and automated regression suites
  - Human nature will tend to concentrate on the “happy” case – need both discipline to write the tests and investment in improving test infrastructure
  - Watch out for unexpected points of failure
    - Failure mode testing is critical – know what your system will do if one of the “non-critical” systems fail. e.g., calls to systems without “fast fail”

Some General Lessons

- Computer systems model the real world – and the real world might be imperfect.
  - For complicated data structures, getting to 100% consistent state across large distributed systems is very expensive and we probably don't need it
  - Time-bounded eventual consistency is usually good enough
    - Example: Deployment, inventory levels
  - Token passing across systems works better than paying for cross system consistent state
    - Example: order state in "stale" caches

Designing for Availability

- Design for recovery - availability = MTBF x **MTTR**
  - Fast undo, fast restart
  - Can your system start from scratch? i.e., deal with the cold start problem
    - Avoid impulse functions
  - Reduce operations that require system restarts
    - E.g., software upgrades, configuration changes
- Watch out for positive feedback loops
  - Retries can be deadly
  - Fast-fail is critical for availability
    - Don't wait for a dead system
    - Don't beat up a system in trouble
  - Per client access control and throttling are important - defend against DOS attacks
- Breaking up a monolithic system can reduce availability if you don't do it right
  - e.g., Break up a monolithic system with 99% availablility:
    - Two systems have to be up: Availability = 99% x 99% = 98%
    - One of two systems have to be up: Availability = 1 – (1% x 1%) = 99.99%

Operating for Availability (technical)

- Latency (not just throughput) is critical
  - Customers care about latency
  - Latency and availability are related via capacity. Capacity calculations depend on:

    (1) average latency of dependent systems

    (2) average offered load by client systems

    Caches change these in dramatic ways so systems will probably not survive a cache disappearing => caches may need to be Tier-1.
  - Poor latency usually indicates a badly scaled or badly designed system
- Details matter
  - Single machines failures can be costly
    - e.g., least connections and the ‘bad’ machine
  - Tuning the system is the full-time job – for a team!
    - e.g., configuring the system for new business, watching performance, dealing with implications of upgrades, dealing with scaling limits of your providers
  - Memory leaks can kill
  - Getting machines up-to-speed fast simplifies scaling

Operating for Availability (management)

- Deployment rules
  - Deploy small changes frequently
    - Eases debugging and improves availability
    - Keeps your political options freer (allows you to say no more often)
  - Clean up all the way after failed deployments (don't let these interfere with non-related deployments)
  - Critical to having both correctness, performance, and "quality of software" standards enforced before software gets deployed
    - Final check should go through an automated system
      - Guarantee to run all checks
      - Arguments about exceptions become arguments about goodness of principle rather than 'just let this one through'
- Migration – moving from legacy to the "best new system"
  - Always actively manage migration and budget for it
    - Out of the box, most new systems need to be tuned
    - Legacy systems do not just die – they have to be shot
      - e.g., search system, presentation tier
  - Legacy systems have a cost: they have to be managed and have to be scaled
    - Teams rapidly lose expertise in the old system especially as migration to new system progress

Contact Info

- I’m around most days: dlee@cs.washington.edu