

CS290f - Lecture 16 Consistency, Harvest, Yield (or how to get away with cheating)

Paper: *Lessons from Giant-Scale Services*, Eric Brewer

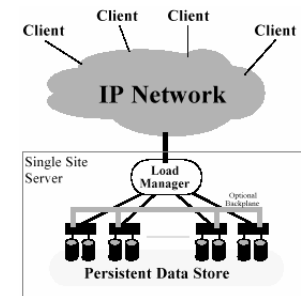
Scalable Internet Services and Systems, Fall 2006

Thorsten von Eicken
Department of Computer Science
University of California at Santa Barbara

Basic Internet Model

■ Components

- Client
- IP network
- Load manager
- Nodes
- Persistent store



Fault Model & End-to-End Semantics

■ End-to-end availability vs. service availability

- Assume no client failover
- $e2e\ avail < srv\ avail$ (e2e adds internet + client faults)
- focus on service availability, use many connections to mitigate internet faults

■ Reload semantics

- Dropping a request is ok, if retry succeeds with high probability
 - ◆ End-to-end fault tolerance depends on the user retrying

■ At-least-once semantics

- Can use transaction id in URL to avoid more-than-once

Load Management

■ Responsibilities

- Provide external name (DNS)
- Load balance traffic
- Isolate faults from clients

■ Data distribution options

- ◆ I.e., does load manager understand distribution of data?
- Symmetric
 - ◆ No: all nodes functionally equal
- Asymmetric
 - ◆ Yes: nodes vary, load manager must map correctly
- Symmetric with affinity
 - ◆ Maybe: all nodes equal, but mapping affects performance
 - ◆ Often: load manager is allowed to make mistakes

Load Management (cont.)

■ Load bal. Properties

- Database aggregation
 - ◆ Does adding nodes increase the DB size?
 - ◆ Generally T for asymmetric, F for symmetric
- Single-query t-put
 - ◆ Does the speed of single type of query scale with nodes?
 - ◆ Generally T for symmetric, F for asymmetric
- Query locality
 - ◆ Is the working set of node < working set of cluster?
 - ◆ Generally T for asymmetric, goal for symmetric with affinity
- Even utilization
 - ◆ How effective is the load balancing?
 - ◆ Generally easier in symmetric

High Availability Metrics

■ Basics

- Burn-in: useful due to “bathtub” failure curve
- People are highest source of failures
- Cables are bad, especially external ones
- Temperature significantly affects longevity

■ Uptime

- Measured in 9's, e.g., per week:
- Uptime = (MTBF - MTTR) / MTBF
- Focus on MTTR:
 - ◆ easier to measure & improve

Nines	Percent	Monthly	Annually
2 9's	99%	7.2hr	87.5 hr
3 9's	99.9%	43m	8.75hr
4 9's	99.99%	4.3m	52m
5 9's	99.999%	26s	5.25m

High Avail. Metrics (cont.)

■ Yield

- Fraction of queries completed
- Def: yield = queries completed / queries offered
- Better than uptime
 - ◆ reflects that all seconds are not of equal value
 - ◆ how could one measure queries not issued due to downtime?

■ Harvest

- Fraction of database reflected in query results
- Def: harvest = data available / complete data
- Examples of fractional harvest beyond search engines?

DQ Principle

■ Data/query x queries/sec = data/sec » constant

- I.e. total data movement in system
- Assumes data movement is bottleneck
 - ◆ CPUs process data as it moves through them
 - ◆ Not: Data moves to/from CPU as it needs it

■ How does DQ change?

- Faults reduce DQ linearly at best
- DQ tends to scale linearly with nodes
- Evaluate hw & sq changes by DQ impact
- Express traffic & feature predictions in terms of DQ

DQ Applied

■ Replication vs. partitioning

Harvest: frac of database
Yield: frac of queries

- Example: 2-node cluster under failure
 - ◆ Replicated: 100% harvest, 50% yield/capacity
 - ◆ Partition: 50% harvest, 100% yield
 - ◆ DQ: reduced in half
 - ? Replication in disk is cheap, but accessing the data isn't

■ Lessons:

- Cost is in providing DQ
 - ◆ Replication on disk is cheap, but *accessing* that data requires DQ points
 - ◆ True replication: need twice the DQ value (not just 2x data)
 - There is no real savings to partitioning over replication
- Partition 'til size is right, then replicate
 - ◆ Repartitioning overhead -> flexibility gained through replicas

9

DQ Applied (cont.)

■ Graceful degradation

- Observations
 - ◆ Peak load 1.6x to 6x average load
 - ◆ Single event bursts (due to media) at 10x peak
 - ◆ Massive faults (e.g. router) degrade DQ a lot
- Degraded service
 - ◆ Admission control: reduce **Q**eries, keep **D**ata/query
 - ◆ Inexact results: reduce **D**, keep **Q**
 - ◆ Admission control based on cost
 - ? Reduces **D** (run more simple queries)
 - ? Increases **Q** (one denied, many accepted)
 - ? Probabilistic: hard queries eventually go through

10

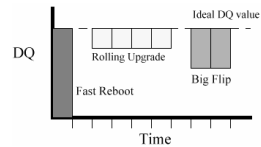
Evolution & growth

■ Upgrade granularity

- Space by the rack
- Repartitioning is hard
- -> Grow by racks or clusters

■ Upgrade strategies

- Fast reboot
- Rolling upgrade
- Big flip



11

CAP principle

■ Pick at most 2 of:

- Strong **C**onsistency
- High **A**vailability
- **P**artition resilience

■ Examples:

- CA no P: DB with distributed transactions
 - ◆ Can't tolerate network partitions
- CP no A: DB without updates if partitioned
 - ◆ Can't update to preserve consistency
- AP no C: System with reduced consistency, e.g. cached/stale data
 - ◆ Can't verify some data under partition

12