

Consistency, Harvest, Yield (or how to get away with cheating)

Scalable Internet Services and Systems, Spring 2001

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

Lessons from Giant-Scale Services

† Eric Brewer

- † CTO & founder of Inktomi

† Overview

- † Giant-Scale services
- † Internet service model
- † Load management
- † High Availability
 - † Metrics
- † Online evolution and growth

† Also...

- † Harvest, Yield, and Scalable Tolerant Systems
 - † Eric Brewer & Armando Fox

2

Where to compute?

† Evolution of computing

- † Mainframe 60's: central building
- † Mini 70's: department-level computer
- † PC 80's: desktop
- † Internet 90's: Web services

† Advantages of services:

- † Access anywhere, anytime
- † Enables groupware
- † Cheaper overall cost
- † Upgrade&add service in-place
- † Convergence with other infrastructure
- † Available via portable/low-cost devices

3

Clusters

† Infrastructure services use clusters

- † Absolute scalability
- † Cost/performance
- † Independent components
- † Incremental scalability

† Challenges

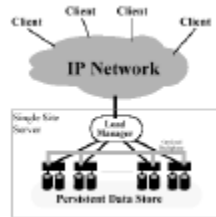
- † Load management
- † High availability
- † Evolution and Growth

4

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
- † 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

- † Symmetric
- † Asymmetric
- † Symmetric with affinity

† Load bal. Properties

- † Database aggregation: adding nodes increases the DB size
- † Single-query t-put: speed of single query scales with nodes
- † Query locality: working set of node < working set of cluster
- † Even utilization: effectiveness of load balancing

High Availability Metrics

† NB: People are highest source of failures

† Uptime

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

2 9's	99%	1h40m
3 9's	99.9%	10m
4 9's	99.99%	1m
5 9's	99.999%	6s

† Yield

- † Fraction of queries completed
- † Def: yield = queries completed / queries offered
- † Better than uptime, but not all seconds are of equal value

† Harvest

- † Fraction of database reflected in query results
- † Def: harvest = data available / complete data

DQ Principle

† Data per query * queries/sec ? constant

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

† Corollaries

- † Harvest * capacity ? constant
 - ‡ Harvest ~ data/query, Capacity ? queries/sec
- † Harvest * yield ? constant (@ high capacity)
 - ‡ Decrease in capacity results in decrease in yield

DQ Applied

† Replication vs. partitioning

- † 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:
 - † Partition 'til size is right, then replicate
 - † Cost is in providing DQ

DQ Applied (cont.)

† Graceful degradation

- † Observations
 - † Peak load 1.6x to 6x average load
 - † Single event bursts at 10x peak
 - † Massive faults (e.g. router)
- † Degraded service
 - † Admission control: reduce Q, keep D
 - † 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

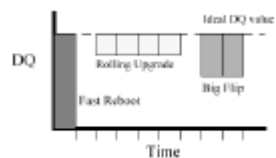
Evolution & growth

† Upgrade granularity

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

† Upgrade strategies

- † Fast reboot
- † Rolling upgrade
- † Big flip



CAP principle

† Pick at most 2 of:

- † Strong Consistency
- † High Availability
- † Partition 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