

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

Scalable Internet Services and Systems, Winter 2002

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

## 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

## Clusters

### Infrastructure services use clusters

- Absolute scalability
  - ◆ Nothing else scales to millions of users
- Cost/performance
  - ◆ Actually dwarfed by bandwidth and ops costs
- Independent components
  - ◆ Independent faults
- Incremental scalability

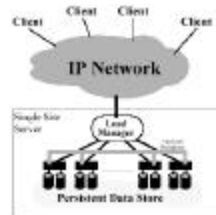
### Challenges

- Load management
- High availability
- Evolution and Growth

## 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

5

## 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

6

## 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

7

## 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	Weekly	Annually
2 9's	99%	1h40m	87.5 hr
3 9's	99.9%	10m	8.75hr
4 9's	99.99%	1m	52m
5 9's	99.999%	6s	5.25m

8

## High Avail. Metrics (cont.)

### Yield

- Fraction of queries completed
- Def:  $\text{yield} = \text{queries completed} / \text{queries offered}$
- Better than uptime
  - ◆ reflects that all seconds are not of equal value

### Harvest

- Fraction of database reflected in query results
- Def:  $\text{harvest} = \text{data available} / \text{complete data}$

## DQ Principle

### Data per query \* queries/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

### Corollaries

- Idea:
  - ◆ Faults reduce DQ
  - ◆ How does this affect uptime, yield, harvest?
- $\text{Harvest} * \text{capacity} \approx \text{constant}$ 
  - ◆  $\text{Harvest} \sim \text{data/query}$
  - ◆  $\text{Capacity} \approx \text{queries/sec}$
- $\text{Harvest} * \text{yield} \approx \text{constant}$  (@ high capacity)
  - ◆ Decrease in capacity results in decrease in yield

## 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:
  - ◆ 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 **Queries**, keep **Data/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

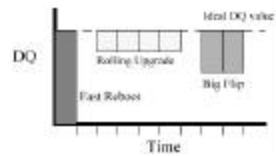
## Evolution & growth

### Upgrade granularity

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

### Upgrade strategies

- Fast reboot
- Rolling upgrade
- Big flip



13

## 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

14