
  
Inktomi

# Towards Robust Distributed Systems

**Dr. Eric A. Brewer**  
 Professor, UC Berkeley  
 Co-Founder & Chief Scientist, Inktomi

PODC Keynote, July 19, 2000


  
Inktomi

## Inktomi at a Glance

|   |  |
|---|--|
| <b>Company Overview</b> <ul style="list-style-type: none"> <li>◆ “INKT” on NASDAQ</li> <li>◆ Founded 1996 out of UC Berkeley</li> <li>◆ ~700 Employees</li> </ul> | <b>Applications</b> <ul style="list-style-type: none"> <li>◆ Search Technology</li> <li>◆ Network Products</li> <li>◆ Online Shopping</li> <li>◆ Wireless Systems</li> </ul> |
|---|--|










PODC Keynote, July 19, 2000



  
Inktomi

## Our Perspective

- ◆ Inktomi builds two distributed systems:
  - Global Search Engines
  - Distributed Web Caches
- ◆ Based on scalable cluster & parallel computing technology
- ◆ But very little use of classic DS research...



PODC Keynote, July 19, 2000


  
Inktomi

## “Distributed Systems” don’t work...

- ◆ There exist working DS:
  - Simple protocols: DNS, WWW
  - Inktomi search, Content Delivery Networks
  - Napster, Verisign, AOL
- ◆ But these are not classic DS:
  - Not distributed objects
  - No RPC
  - No modularity
  - Complex ones are single owner (except phones)

PODC Keynote, July 19, 2000

## Three Basic Issues



- ◆ Where is the state?
- ◆ Consistency vs. Availability
- ◆ Understanding Boundaries

PODC Keynote, July 19, 2000

## Where's the state?

(not all locations are equal)



PODC Keynote, July 19, 2000

## Santa Clara Cluster



- Very uniform
- No monitors
- No people
- No cables
- Working power
- Working A/C
- Working BW



000

## Delivering High Availability



We kept up the service through:

- ◆ Crashes & disk failures (weekly)
- ◆ Database upgrades (daily)
- ◆ Software upgrades (weekly to monthly)
- ◆ OS upgrades (twice)
- ◆ Power outage (several)
- ◆ Network outages (now have 11 connections)
- ◆ Physical move of all equipment (twice)

PODC Keynote, July 19, 2000

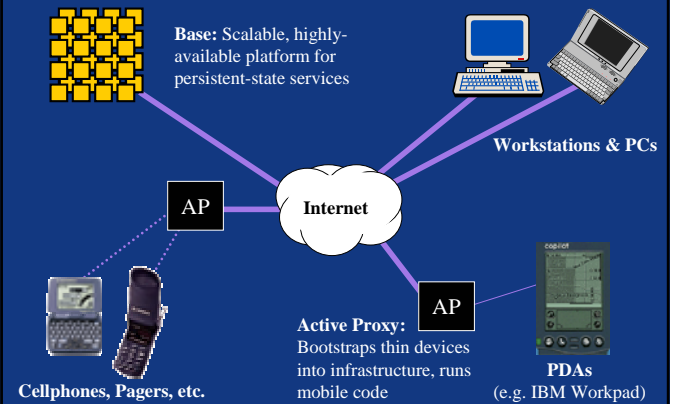
## Persistent State is HARD



- ◆ Classic DS focus on the computation, not the data
  - this is WRONG, computation is the easy part
- ◆ Data centers exist for a reason
  - can't have consistency or availability without them
- ◆ Other locations are for caching only:
  - proxies, basestations, set-top boxes, desktops
  - phones, PDAs, ...
- ◆ Distributed systems can't ignore location distinctions

PODC Keynote, July 19, 2000

## Berkeley Ninja Architecture



## Consistency vs. Availability (ACID vs. BASE)



PODC Keynote, July 19, 2000

## ACID vs. BASE



- ◆ DBMS research is about ACID (mostly)
- ◆ But we forfeit "C" and "T" for availability, graceful degradation, and performance

**This tradeoff is fundamental.**

**BASE:**

- **B**asically **A**vailable
- **S**oft-state
- **E**ventual consistency

PODC Keynote, July 19, 2000

## ACID vs. BASE

### ACID

- ◆ Strong consistency
- ◆ Isolation
- ◆ Focus on “commit”
- ◆ Nested transactions
- ◆ Availability?
- ◆ Conservative (pessimistic)
- ◆ Difficult evolution (e.g. schema)

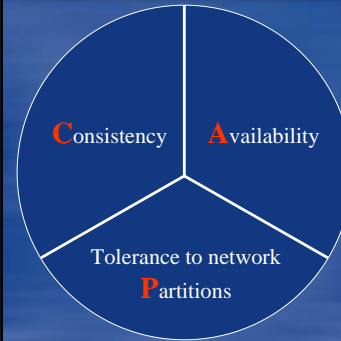
### BASE

- ◆ Weak consistency
  - stale data OK
- ◆ Availability first
- ◆ Best effort
- ◆ Approximate answers OK
- ◆ Aggressive (optimistic)
- ◆ Simpler!
- ◆ Faster
- ◆ Easier evolution

← But I think it's a spectrum →

PODC Keynote, July 19, 2000

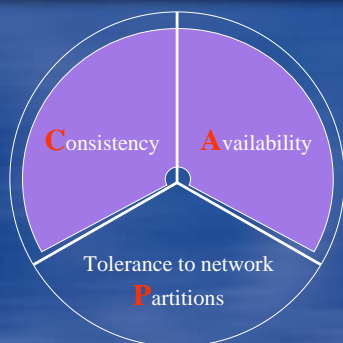
## The CAP Theorem



Theorem: You can have **at most two** of these properties for any shared-data system

PODC Keynote, July 19, 2000

## Forfeit Partitions



### Examples

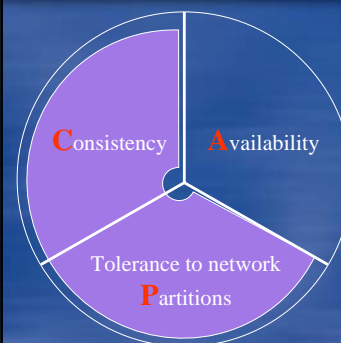
- ◆ Single-site databases
- ◆ Cluster databases
- ◆ LDAP
- ◆ xFS file system

### Traits

- ◆ 2-phase commit
- ◆ cache validation protocols

PODC Keynote, July 19, 2000

## Forfeit Availability



### Examples

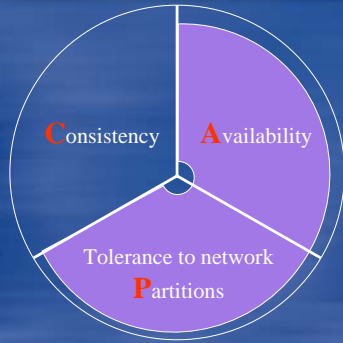
- ◆ Distributed databases
- ◆ Distributed locking
- ◆ Majority protocols

### Traits

- ◆ Pessimistic locking
- ◆ Make minority partitions unavailable

PODC Keynote, July 19, 2000

## Forfeit Consistency



### Examples

- ◆ Coda
- ◆ Web caching
- ◆ DNS

### Traits

- ◆ expirations/leases
- ◆ conflict resolution
- ◆ optimistic

PODC Keynote, July 19, 2000

## These Tradeoffs are Real



- ◆ The *whole* space is useful
- ◆ Real internet systems are a careful *mixture* of ACID and BASE subsystems
  - We use ACID for user profiles and logging (for revenue)
- ◆ But there is almost no work in this area
- ◆ Symptom of a deeper problem: systems and database communities are separate but overlapping (with distinct vocabulary)

PODC Keynote, July 19, 2000

## CAP Take Homes



- ◆ Can have consistency & availability within a cluster (foundation of Ninja), but it is still hard in practice
- ◆ OS/Networking good at BASE/Availability, but terrible at consistency
- ◆ Databases better at C than Availability
- ◆ Wide-area databases can't have both
- ◆ Disconnected clients can't have both
- ◆ All systems are probabilistic...

PODC Keynote, July 19, 2000

## Understanding Boundaries (the RPC hangover)



PODC Keynote, July 19, 2000

## The Boundary



- ◆ The interface between two modules
  - client/server, peers, libraries, etc...
- ◆ Basic boundary = the procedure call



- thread traverses the boundary
- two sides are in the same address space

PODC Keynote, July 19, 2000

## Different Address Spaces



- ◆ What if the two sides are NOT in the same address space?
  - IPC or LRPC
- ◆ Can't do pass-by-reference (pointers)
  - Most IPC screws this up: pass by value-result
  - There are TWO copies of args not one
- ◆ What if they share some memory?
  - Can pass pointers, but...
  - Need synchronization between client/server
  - Not all pointers can be passed

PODC Keynote, July 19, 2000

## Trust the other side?



- ◆ What if we don't trust the other side?
- ◆ Have to check args, no pointer passing
- ◆ Kernels get this right:
  - copy/check args
  - use opaque references (e.g. File Descriptors)
- ◆ Most systems do not:
  - TCP
  - Napster
  - web browsers

PODC Keynote, July 19, 2000

## Partial Failure



- ◆ Can the two sides fail independently?
  - RPC, IPC, LRPC
- ◆ Can't be transparent (like RPC) !!
- ◆ New exceptions (other side gone)
- ◆ Reclaim local resources
  - e.g. kernels leak sockets over time => reboot
- ◆ Can use leases?
  - Different new exceptions: lease expired
- ◆ RPC tries to hide these issues (but fails)

PODC Keynote, July 19, 2000



## Multiplexing clients?



- ◆ Does the server have to:
  - deal with high concurrency?
  - Say “no” sometimes (graceful degradation)
  - Treat clients equally (fairness)
  - Bill for resources (and have audit trail)
  - Isolate clients performance, data, ....
- ◆ These all affect the boundary definition

PODC Keynote, July 19, 2000

## Boundary evolution?



- ◆ Can the two sides be updated independently?  
(NO)
- ◆ The DLL problem...
- ◆ Boundaries need versions
- ◆ Negotiation protocol for upgrade?
- ◆ Promises of backward compatibility?
- ◆ Affects naming too (version number)

PODC Keynote, July 19, 2000

## Example: protocols vs. APIs



- ◆ Protocols have been more successful than APIs
- ◆ Some reasons:
  - protocols are pass by value
  - protocols designed for partial failure
  - not trying to look like local procedure calls
  - explicit state machine, rather than call/return (this exposes exceptions well)
- ◆ Protocols still not good at trust, billing, evolution

PODC Keynote, July 19, 2000

## Example: XML



- ◆ XML doesn't solve any of these issues
- ◆ It is RPC with an extensible type system
- ◆ It makes evolution better?
  - two sides need to agree on schema
  - can ignore stuff you don't understand
- ◆ Can mislead us to ignore the real issues

PODC Keynote, July 19, 2000

## Boundary Summary



- ◆ We have been very sloppy about boundaries
- ◆ Leads to fragile systems
- ◆ Root cause is false transparency: trying to look like local procedure calls
- ◆ Relatively little work in evolution, federation, client-based resource allocation, failure recovery

PODC Keynote, July 19, 2000

## Conclusions



- ◆ Classic Distributed Systems are fragile
- ◆ Some of the causes:
  - focus on computation, not data
  - ignoring location distinctions
  - poor definitions of consistency/availability goals
  - poor understanding of boundaries (RPC in particular)
- ◆ These are all fixable, but need to be far more common

PODC Keynote, July 19, 2000

## The DQ Principle



**Data/query \* Queries/sec = constant = DQ**

- for a given node
- for a given app/OS release
- ◆ A fault can reduce the capacity (Q), completeness (D) or both
- ◆ Faults reduce this constant linearly (at best)

PODC Keynote, July 19, 2000

## Harvest & Yield



- ◆ **Yield: Fraction of Answered Queries**
  - Related to uptime but measured by queries, not by time
  - Drop 1 out of 10 connections => 90% yield
  - At full utilization: **yield ~ capacity ~ Q**
- ◆ **Harvest: Fraction of the Complete Result**
  - Reflects that some of the data may be missing due to faults
  - Replication: maintain D under faults
- ◆ **DQ corollary: **harvest \* yield ~ constant****
  - ACID => choose 100% harvest (reduce Q but 100% D)
  - Internet => choose 100% yield (available but reduced D)

PODC Keynote, July 19, 2000



## Harvest Options



### 1) Ignore lost nodes

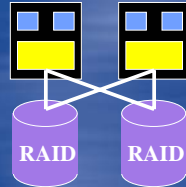
- RPC gives up
- forfeit small part of the database
- reduce D, keep Q

### 2) Pair up nodes

- RPC tries alternate
- survives one fault per pair
- reduce Q, keep D

### 3) n-member replica groups

Decide *when* you care...



PODC Keynote, July 19, 2000

## Replica Groups



With  $n$  members:

- ◆ Each fault reduces Q by  $1/n$
- ◆ D stable until  $n$ th fault
- ◆ Added load is  $1/(n-1)$  per fault
  - $n=2 \Rightarrow$  double load or 50% capacity
  - $n=4 \Rightarrow$  133% load or 75% capacity
  - “load redirection problem”
- ◆ Disaster tolerance: better have  $>3$  mirrors

PODC Keynote, July 19, 2000

## Graceful Degradation



- ◆ Goal: smooth decrease in harvest/yield proportional to faults
  - we know DQ drops linearly
- ◆ Saturation will occur
  - high peak/average ratios...
  - must reduce harvest or yield (or both)
  - must do admission control!!!
- ◆ One answer: reduce D dynamically
  - disaster  $\Rightarrow$  redirect load, then reduce D to compensate for extra load

PODC Keynote, July 19, 2000

## Thinking Probabilistically



- ◆ Maximize symmetry
  - SPMD + simple replication schemes
- ◆ Make faults independent
  - requires thought
  - avoid cascading errors/faults
  - understand redirected load
  - KISS
- ◆ Use randomness
  - makes worst-case and average case the same
  - ex: Inktomi spreads data & queries randomly
  - Node loss implies a random 1% harvest reduction

PODC Keynote, July 19, 2000

## Server Pollution



- ◆ Can't fix all memory leaks
- ◆ Third-party software leaks memory and sockets
  - so does the OS sometimes
- ◆ Some failures tie up local resources

### Solution: planned periodic “bounce”

- Not worth the stress to do any better
- Bounce time is less than 10 seconds
- Nice to remove load first...

PODC Keynote, July 19, 2000

## Evolution



### Three Approaches:

- ◆ Flash Upgrade
  - Fast reboot into new version
  - Focus on MTTR (< 10 sec)
  - Reduces yield (and uptime)
- ◆ Rolling Upgrade
  - Upgrade nodes one at time in a “wave”
  - Temporary 1/n harvest reduction, 100% yield
  - Requires co-existing versions
- ◆ “Big Flip”

PODC Keynote, July 19, 2000

## The Big Flip



- ◆ Steps:
  - 1) take down 1/2 the nodes
  - 2) upgrade that half
  - 3) flip the “active half” (site upgraded)
  - 4) upgrade second half
  - 5) return to 100%

- ◆ 50% Harvest, 100% Yield
  - or inverse?

- ◆ No mixed versions
  - can replace schema, protocols, ...

- ◆ Twice used to change physical location

PODC Keynote, July 19, 2000

## Key New Problems



- ◆ Unknown but large growth
  - Incremental & Absolute scalability
  - 1000's of components
- ◆ Must be truly highly available
  - Hot swap everything (no recovery time allowed)
  - No “night”
  - Graceful degradation under faults & saturation
- ◆ Constant evolution (internet time)
  - Software will be buggy
  - Hardware will fail
  - These can't be emergencies...

PODC Keynote, July 19, 2000

## Conclusions



### ◆ Parallel Programming is very relevant, except...

- historically avoids availability
- no notion of online evolution
- limited notions of graceful degradation (checkpointing)
- best for CPU-bound tasks

### ◆ Must think probabilistically about everything

- no such thing as a 100% working system
- no such thing as 100% fault tolerance
- partial results are often OK (and better than none)
- Capacity \* Completeness == Constant

PODC Keynote, July 19, 2000

## Conclusions



### ◆ Winning solution is message-passing clusters

- fine-grain communication => fine-grain exception handling
- don't want every load/store to deal with partial failure

### ◆ Key open problems:

- libraries & data structures for HA shared state
- support for replication and partial failure
- better understanding of probabilistic systems
- cleaner support for exceptions (graceful degradation)
- support for split-phase I/O and many concurrent threads
- support for 10,000 threads/node (to avoid FSMs)

PODC Keynote, July 19, 2000

## Backup slides



PODC Keynote, July 19, 2000

## New Hard Problems...



### ◆ Really need to manage **disks** well

- problems are I/O bound, not CPU bound

### ◆ Lots of **simultaneous connections**

- 50Kb/s => at least 2000 connections/node

### ◆ HAS to be **highly available**

- no maintenance window, even for upgrades

### ◆ Continuous **evolution**

- constant site changes, always small bugs...
- large but unpredictable traffic growth

### ◆ **Graceful degradation** under saturation

PODC Keynote, July 19, 2000

## Parallel Disk I/O



- ◆ **Want 50+ outstanding reads/disk**
  - Provides disk-head scheduler with many choices
  - Trades response time for **throughput**
- ◆ **Pushes towards a **split-phase approach** to disks**
- ◆ **General trend: each query is a finite-state machine**
  - split-phase disk/network operations are state transitions
  - multiplex many FSMs over small number of threads
  - FSM handles state rather than thread stack

PODC Keynote, July 19, 2000