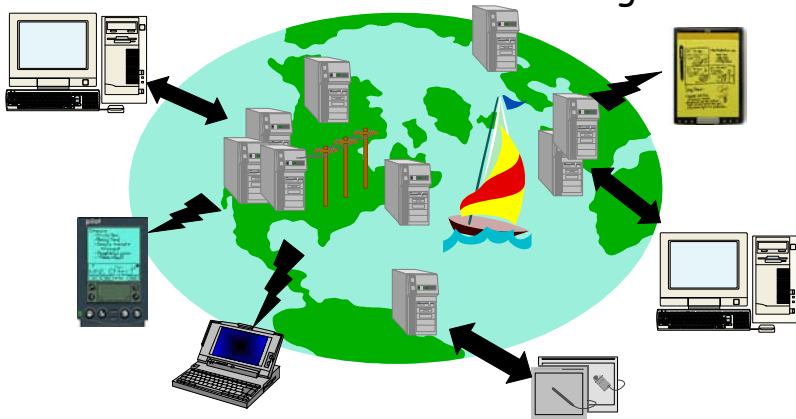


## OceanStore

Toward Global-Scale, Self-Repairing,  
Secure and Persistent Storage

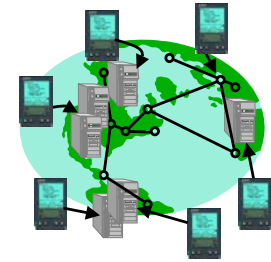


John Kubiawicz

University of California at Berkeley

## OceanStore Context: Ubiquitous Computing

- Computing everywhere:
  - Desktop, Laptop, Palmtop
  - Cars, Cellphones
  - Shoes? Clothing? Walls?
- Connectivity everywhere:
  - Rapid growth of bandwidth in the interior of the net
  - Broadband to the home and office
  - Wireless technologies such as CMDA, Satelite, laser
- Where is persistent data????

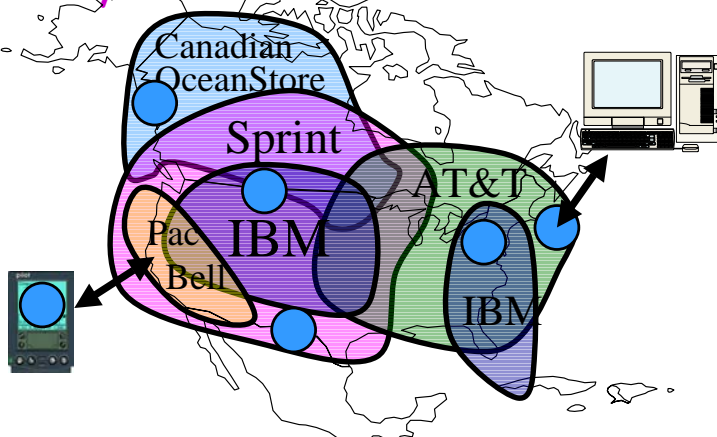


MIT Seminar

©2002 John Kubiawicz/UC Berkeley

OceanStore:2

## Utility-based Infrastructure?



- Data service provided by storage federation
- Cross-administrative domain
- Pay for Service

## OceanStore: Everyone's Data, One Big Utility "The data is just out there"

- How many files in the OceanStore?
  - Assume  $10^{10}$  people in world
  - Say 10,000 files/person (very conservative?)
  - So  $10^{14}$  files in OceanStore!
- If 1 gig files (ok, a stretch), get 1 mole of bytes!

Truly impressive number of elements...

... but small relative to physical constants

Aside: new results: 1.5 Exabytes/year ( $1.5 \times 10^{18}$ )

MIT Seminar

©2002 John Kubiawicz/UC Berkeley

OceanStore:3

MIT Seminar

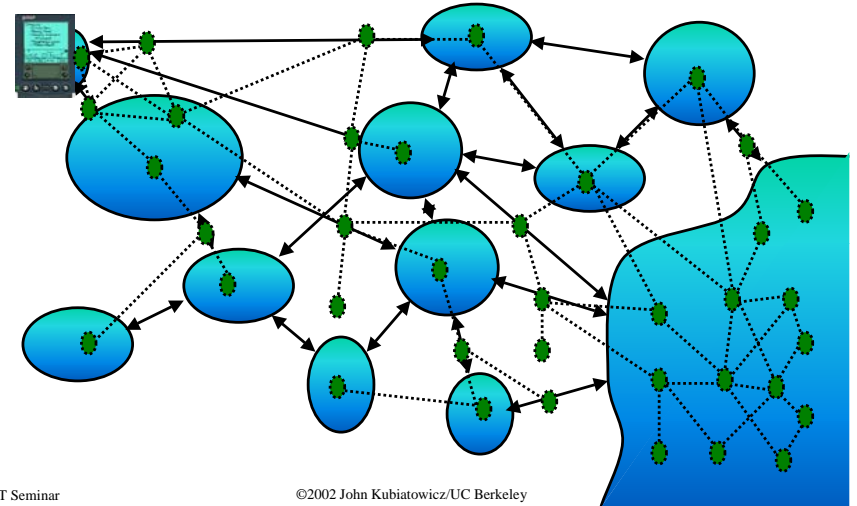
©2002 John Kubiawicz/UC Berkeley

OceanStore:4

## OceanStore Assumptions

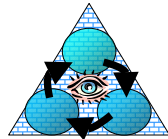
- **Untrusted Infrastructure:**
  - The OceanStore is comprised of untrusted components
  - Individual hardware has finite lifetimes
  - All data encrypted within the infrastructure
- **Responsible Party:**
  - Some organization (*i.e. service provider*) guarantees that your data is consistent and durable
  - Not trusted with *content* of data, merely its *integrity*
- **Mostly Well-Connected:**
  - Data producers and consumers are connected to a high-bandwidth network most of the time
  - Exploit multicast for quicker consistency when possible
- **Promiscuous Caching:**
  - Data may be cached anywhere, anytime

## The Peer-To-Peer View: Irregular Mesh of "Pools"



## Key Observation: Want Automatic Maintenance

- Can't possibly manage billions of servers by hand!
- System should automatically:
  - Adapt to failure
  - Exclude malicious elements
  - Repair itself
  - Incorporate new elements
- System should be secure and private
  - Encryption, authentication
- System should preserve data over the long term (*accessible for 1000 years*):
  - Geographic distribution of information
  - New servers added from time to time
  - Old servers removed from time to time
  - Everything just works



## Outline: Three Technologies and a Principle

### Principle: ThermoSpective Systems Design

- Redundancy and Repair everywhere
1. Structured, Self-Verifying Data
    - Let the Infrastructure Know What is important
  2. Decentralized Object Location and Routing
    - A new abstraction for routing
  3. Deep Archival Storage
    - Long Term Durability



## Goal: Stable, large-scale systems

- State of the art:
  - Chips:  $10^8$  transistors, 8 layers of metal
  - Internet:  $10^9$  hosts, terabytes of bisection bandwidth
  - Societies:  $10^8$  to  $10^9$  people, 6-degrees of separation
- Complexity is a liability!
  - More components  $\Rightarrow$  Higher failure rate
  - Chip verification > 50% of design team
  - Large societies unstable (especially when centralized)
  - **Small, simple, perfect components combine to generate complex emergent behavior!**
- Can complexity be a useful thing?
  - Redundancy and interaction can yield stable behavior
  - **Better figure out new ways to design things...**

Question: Can we exploit Complexity to our Advantage?

Moore's Law gains  $\Rightarrow$  Potential for Stability

## The Thermodynamic Analogy

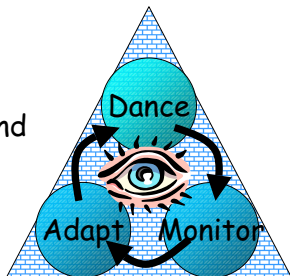
- Large Systems have a variety of *latent order*
  - Connections between elements
  - Mathematical structure (erasure coding, etc)
  - **Distributions peaked about some desired behavior**
- Permits "Stability through Statistics"
  - Exploit the behavior of aggregates (redundancy)
- Subject to Entropy
  - Servers fail, attacks happen, system changes
- Requires continuous repair
  - Apply energy (i.e. through servers) to reduce entropy

## The Biological Inspiration

- Biological Systems are built from (extremely) faulty components, yet:
  - They operate with a variety of component failures ⇒ Redundancy of function and representation
  - They have stable behavior ⇒ Negative feedback
  - They are self-tuning ⇒ Optimization of common case

- **Introspective (Autonomic) Computing:**

- Components for performing
- Components for monitoring and model building
- Components for continuous adaptation



## ThermoSpective

- Many Redundant Components (Fault Tolerance)
- Continuous Repair (Entropy Reduction)

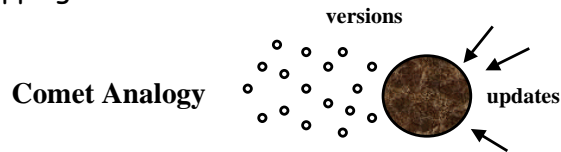


## Let the Infrastructure help You!

- End-to-end and everywhere else:
  - Must distribute responsibility to guarantee: QoS, Latency, Availability, Durability
- Let the infrastructure understand the *vocabulary or semantics* of the application
  - Rules of correct interaction?

# OceanStore Data Model

- **Versioned Objects**
  - Every update generates a new version
  - Can always go back in time (Time Travel)
- **Each Version is Read-Only**
  - Can have permanent name
  - Much easier to repair
- **An Object is a signed mapping between permanent name and latest version**
  - Write access control/integrity involves managing these mappings

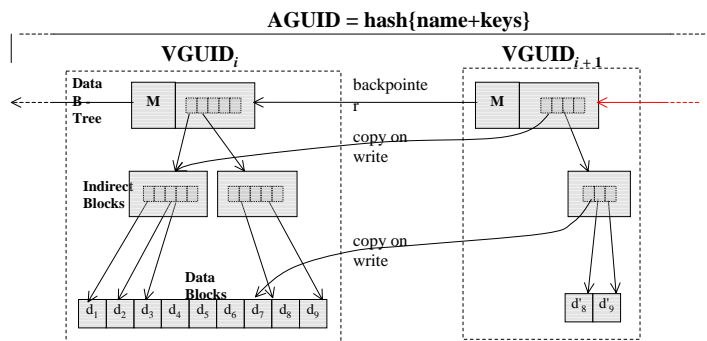


# Secure Hashing

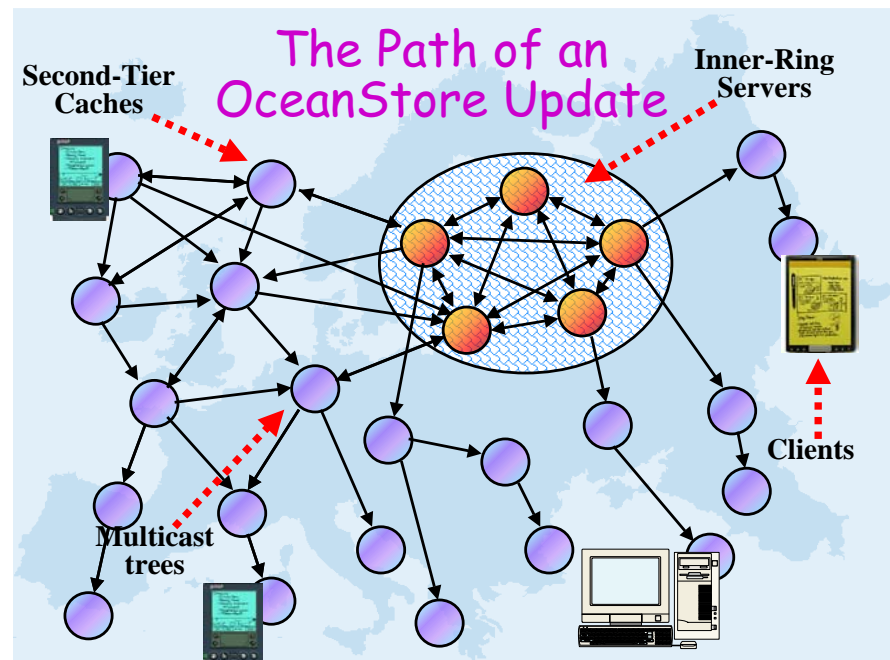


- **Read-only data:** GUID is hash over actual data
  - Uniqueness and Unforgeability: the data is what it is!
  - Verification: check hash over data
- **Changeable data:** GUID is combined hash over a human-readable name + public key
  - Uniqueness: GUID space selected by public key
  - Unforgeability: public key is indelibly bound to GUID
- **Thermodynamic insight:** Hashing makes "data particles" unique, simplifying interactions

# Self-Verifying Objects



♥ **Heartbeat:** {AGUID, VGUID, Timestamp}<sub>signed</sub>

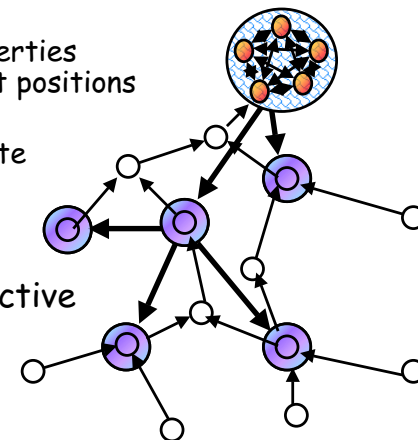


## OceanStore Consistency via Conflict Resolution

- Consistency is form of optimistic concurrency
  - An update packet contains a series of *predicate-action* pairs which operate on encrypted data
  - Each predicate tried in turn:
    - If none match, the update is *aborted*
    - Otherwise, action of first true predicate is *applied*
- Role of Responsible Party
  - All updates submitted to Responsible Party which chooses a final total order
  - Byzantine agreement with threshold signatures
- This is powerful enough to synthesize:
  - ACID database semantics
  - release consistency (build and use MCS-style locks)
  - Extremely loose (weak) consistency

## Self-Organizing Soft-State Replication

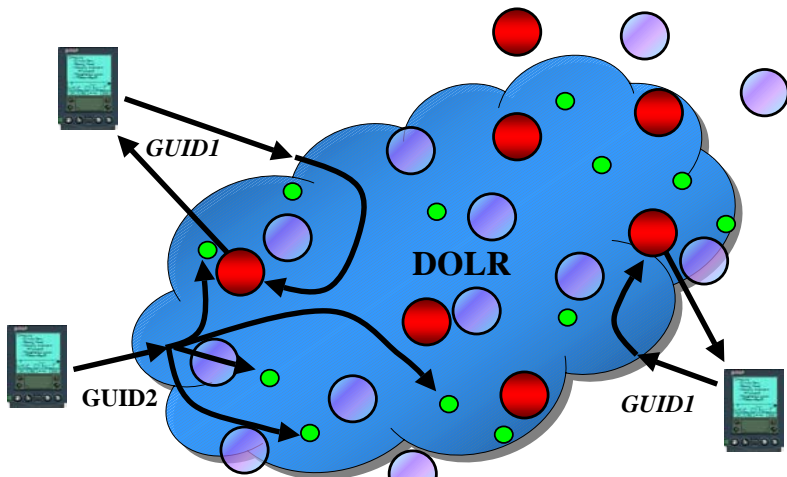
- Simple algorithms for placing replicas on nodes in the interior
  - Intuition: locality properties of Tapestry help select positions for replicas
  - Tapestry helps associate parents and children to build multicast tree
- Preliminary results show that this is effective



## Locality, Locality, Locality One of the defining principles

- *The ability to exploit local resources over remote ones whenever possible*
- "-Centric" approach
  - Client-centric, server-centric, data source-centric
- Requirements:
  - Find data quickly, wherever it might reside
    - Locate nearby object without global communication
    - Permit rapid object migration
  - Verifiable: can't be sidetracked
    - Data name cryptographically related to data

## Enabling Technology: DOLR (Decentralized Object Location and Routing)

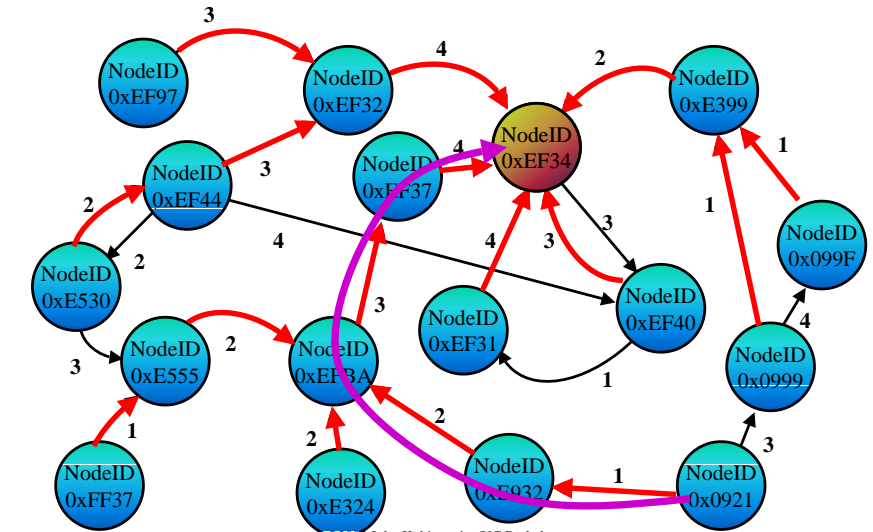


MIT Seminar

©2002 John Kubiatowicz/UC Berkeley

OceanStore:25

## Basic Tapestry Mesh Incremental Prefix-based Routing

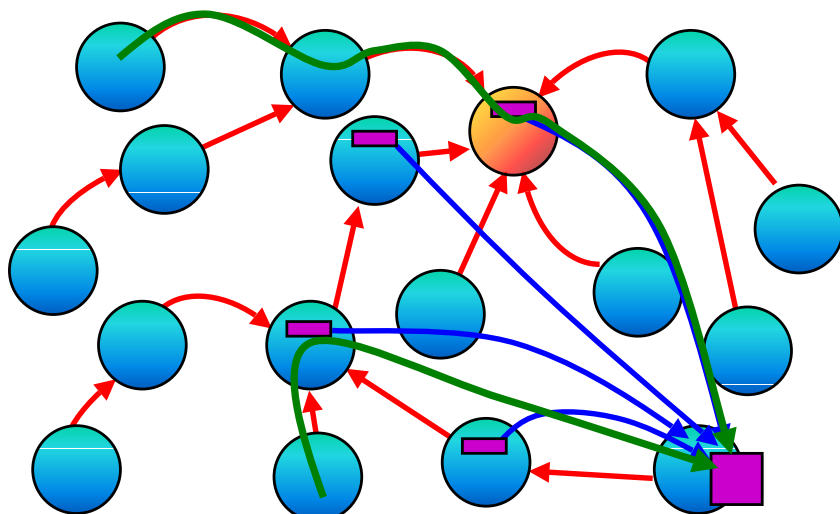


MIT Seminar

©2002 John Kubiatowicz/UC Berkeley

OceanStore:26

## Use of Tapestry Mesh Randomization and Locality



MIT Seminar

©2002 John Kubiatowicz/UC Berkeley

OceanStore:27

## Stability under Faults

- Instability is the common case....!
  - Small half-life for P2P apps (1 hour????)
  - Congestion, flash crowds, misconfiguration, faults
- **Must Use DOLR under instability!**
  - The right thing must just happen
- Tapestry is natural framework to exploit redundant elements and connections
  - Multiple Roots, Links, etc.
  - Easy to reconstruct routing and location information
  - Stable, repairable layer
- **Thermodynamic analogies:**
  - Heat Capacity of DOLR network
  - Entropy of Links (decay of underlying order)

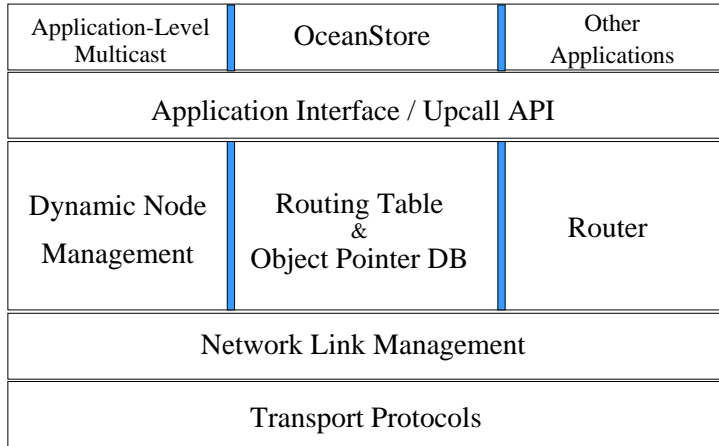


MIT Seminar

©2002 John Kubiatowicz/UC Berkeley

OceanStore:28

# Single Node Tapestry

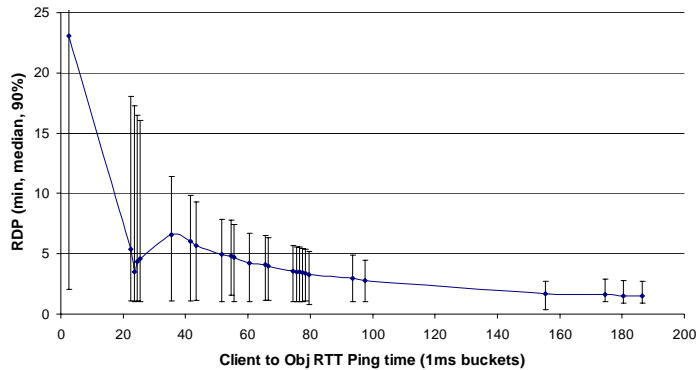


# It's Alive!

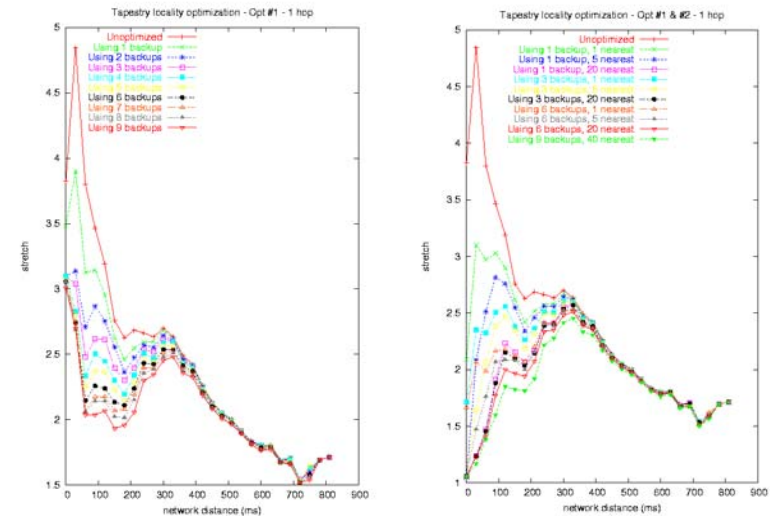


- Planet Lab global network
  - 98 machines at 42 institutions, in North America, Europe, Australia (~ 60 machines utilized)
  - 1.26Ghz PIII (1GB RAM), 1.8Ghz PIV (2GB RAM)
  - North American machines (2/3) on Internet2
- Tapestry Java deployment
  - 6-7 nodes on each physical machine
  - IBM Java JDK 1.30
  - Node virtualization inside JVM and SEDA
  - Scheduling between virtual nodes increases latency

# Object Location

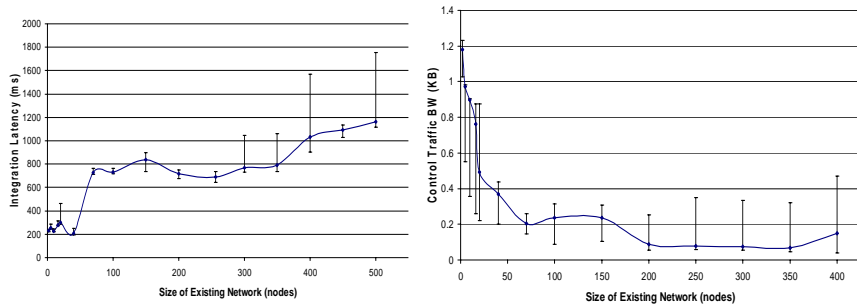


# Tradeoff: Storage vs Locality





## Management Behavior

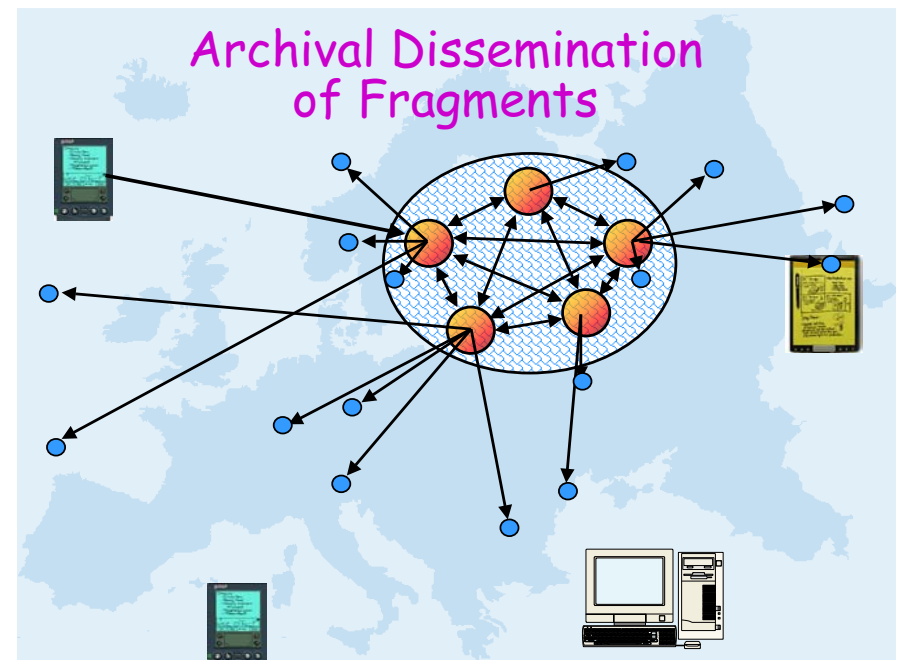


- Integration Latency (Left)
  - Humps = additional levels
- Cost/node Integration Bandwidth (right)
  - Localized!
- Continuous, multi-node insertion/deletion works!

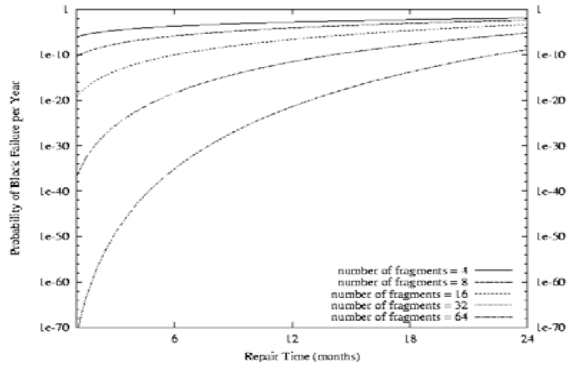


## Two Types of OceanStore Data

- **Active Data:** "Floating Replicas"
  - Per object virtual server
  - Interaction with other replicas for consistency
  - May appear and disappear like bubbles
- **Archival Data:** **OceanStore's Stable Store**
  - m-of-n coding: Like hologram
    - Data coded into  $n$  fragments, any  $m$  of which are sufficient to reconstruct (e.g  $m=16, n=64$ )
    - Coding overhead is proportional to  $n \div m$  (e.g 4)
    - Other parameter, *rate*, is  $1/\text{overhead}$
  - Fragments are cryptographically self-verifying
- **Most data in the OceanStore is archival!**

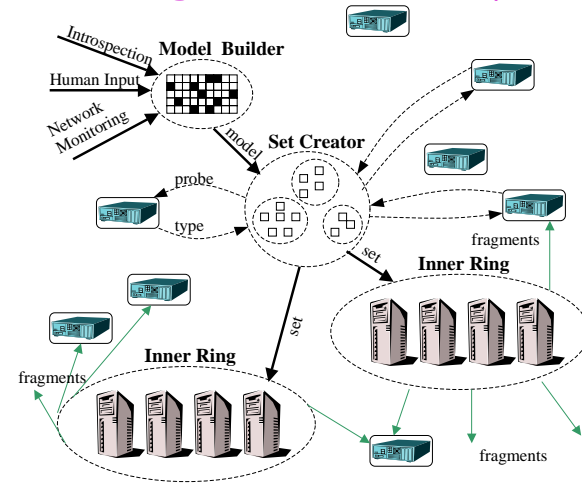


# Fraction of Blocks Lost per Year (FBLPY)

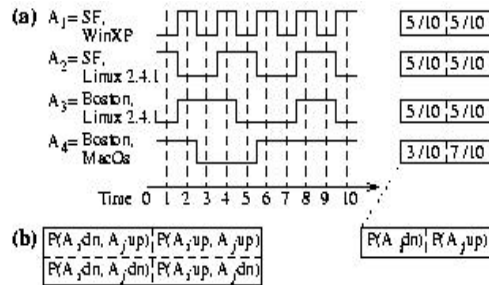


- Exploit law of large numbers for durability!
- 6 month repair, FBLPY:
  - Replication: 0.03
  - Fragmentation: 10<sup>-35</sup>

# The Dissemination Process: Achieving Failure Independence

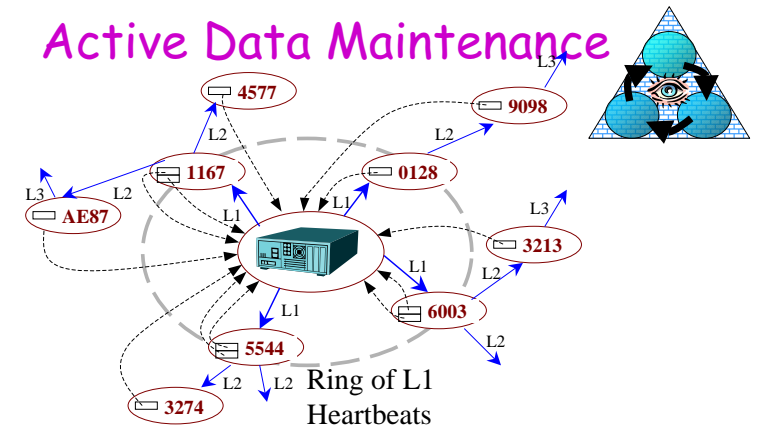


# Independence Analysis



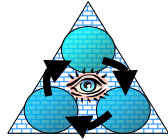
- Information gathering:
  - State of fragment servers (up/down/etc)
- Correlation analysis:
  - Use metric such as mutual information
  - Cluster via that metric
  - Result partitions servers into *uncorrelated clusters*

# Active Data Maintenance



- Tapestry enables "data-driven multicast"
  - Mechanism for local servers to watch each other
  - Efficient use of bandwidth (locality)

## 1000-Year Durability?



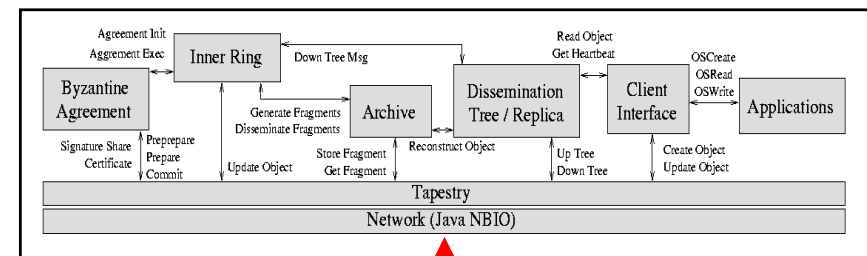
- Exploiting Infrastructure for Repair
  - DOLR permits efficient heartbeat mechanism to notice:
    - Servers going away for a while
    - Or, going away forever!
  - Continuous sweep through data also possible
  - Erasure Code provides Flexibility in Timing
- Data continuously transferred from physical medium to physical medium
  - No "tapes decaying in basement"
  - Information becomes fully Virtualized
- **Thermodynamic Analogy:** Use of Energy (supplied by servers) to Suppress Entropy



## First Implementation [Java]:

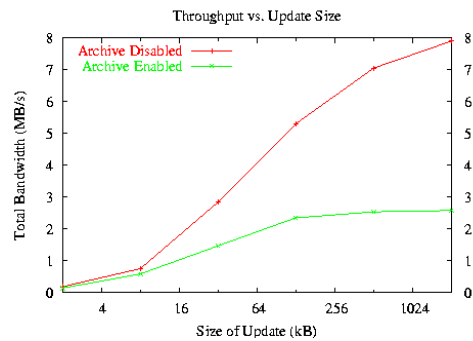
- Event-driven state-machine model
  - 150,000 lines of Java code and growing
- Included Components
  - ✓ DOLR Network (Tapestry)
    - Object location with Locality
    - Self Configuring, Self Repairing
  - ✓ Full Write path
    - Conflict resolution and Byzantine agreement
  - ✓ Self-Organizing Second Tier
    - Replica Placement and Multicast Tree Construction
  - ✓ Introspective gathering of tacit info and adaptation
    - Clustering, prefetching, adaptation of network routing
  - ✓ Archival facilities
    - Interleaved Reed-Solomon codes for fragmentation
    - Independence Monitoring
    - Data-Driven Repair
- Downloads available from [www.oceanstore.org](http://www.oceanstore.org)

## Event-Driven Architecture of an OceanStore Node



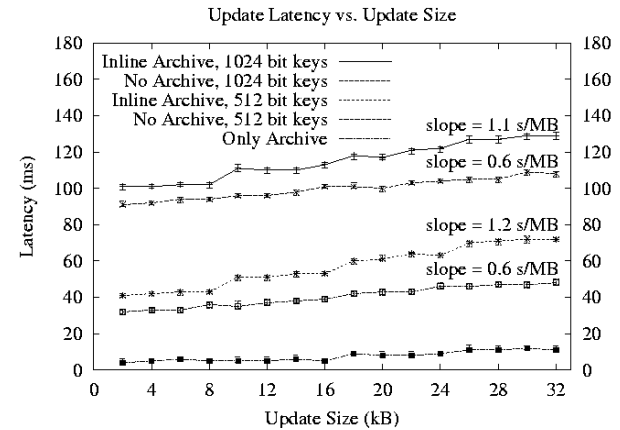
- Data-flow style
  - Arrows Indicate flow of messages
- Potential to exploit small multiprocessors at each physical node

# First Prototype Works!



- Latest: it is up to 8MB/sec (local area network)
  - Biggest constraint: Threshold Signatures
- Still a ways to go, but working

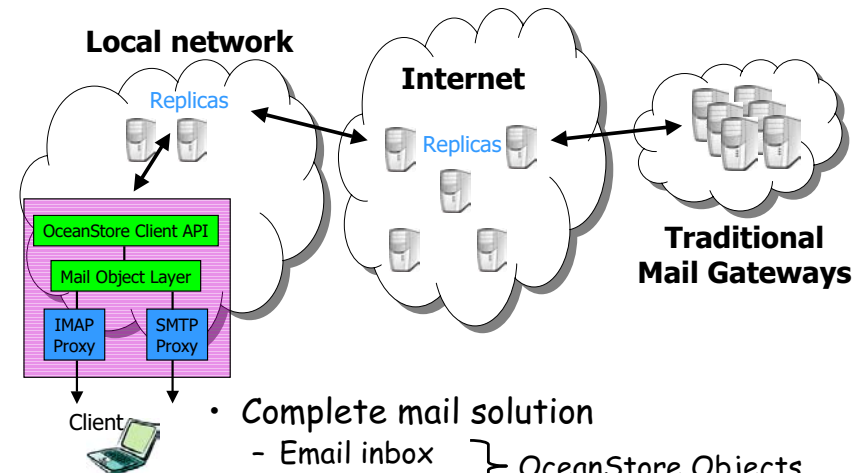
# Update Latency



- Cryptography in critical path (not surprising!)

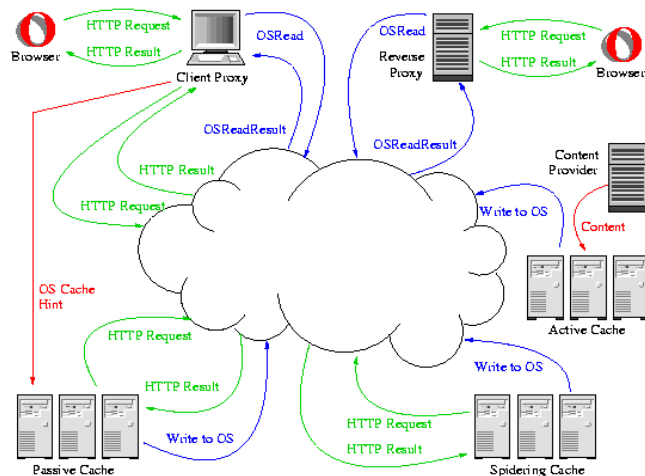


# MINO: Wide-Area E-Mail Service



- Complete mail solution
  - Email inbox
  - Imap folders
 } OceanStore Objects

## Riptide: Caching the Web with OceanStore



## Other Apps

- Long-running archive
  - Project Segull
- File system support
  - NFS with time travel (like VMS)
  - Windows Installable file system (soon)
- Anonymous file storage:
  - Nemosyne uses Tapestry by itself
- Palm-pilot synchronization
  - Palm data base as an OceanStore DB

## Conclusions

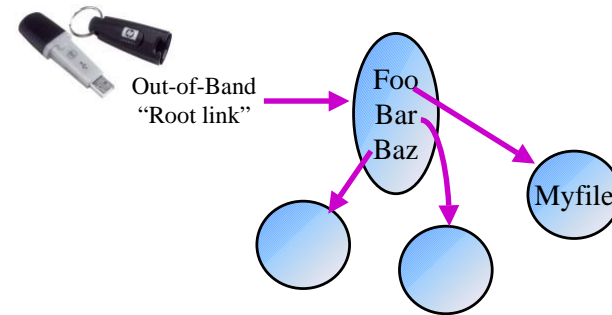
- **Exploitation of Complexity**
  - Large amounts of redundancy and connectivity
  - Thermodynamics of systems: "Stability through Statistics"
  - Continuous Introspection
- **Help the Infrastructure to Help you**
  - Decentralized Object Location and Routing (DOLR)
  - Object-based Storage
  - Self-Organizing redundancy
  - Continuous Repair
- **OceanStore properties:**
  - Provides security, privacy, and integrity
  - Provides extreme durability
  - Lower maintenance cost through redundancy, continuous adaptation, self-diagnosis and repair

For more info:  
<http://oceanstore.org>

- OceanStore vision paper for ASPLOS 2000  
"OceanStore: An Architecture for Global-Scale Persistent Storage"
- Tapestry algorithms paper (SPAA 2002):  
"Distributed Object Location in a Dynamic Network"
- Bloom Filters for Probabilistic Routing (INFOCOM 2002):  
"Probabilistic Location and Routing"
- Upcoming CACM paper (not until February):  
- "Extracting Guarantees from Chaos"

# Backup Slides

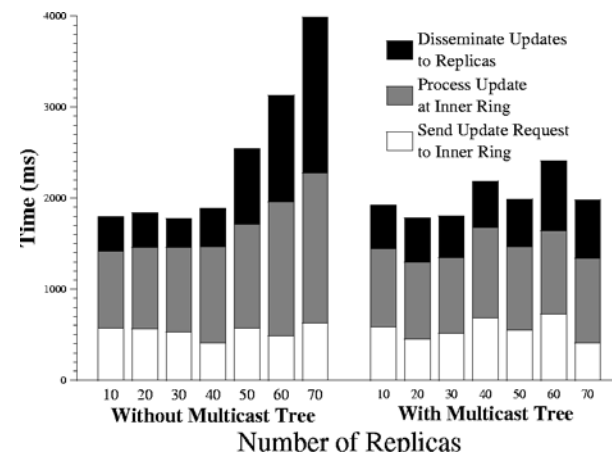
# Secure Naming



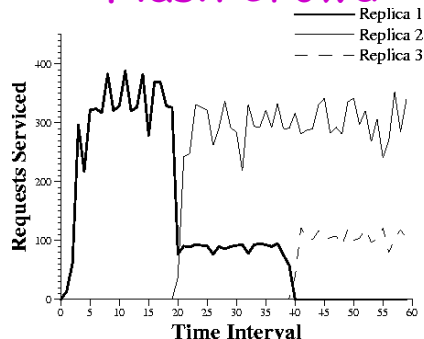
- Naming hierarchy:
  - Users map from names to GUIDs via hierarchy of OceanStore objects (*ala SDSI*)
  - Requires set of "root keys" to be acquired by user



# Effectiveness of second tier



## Second Tier Adaptation: Flash Crowd



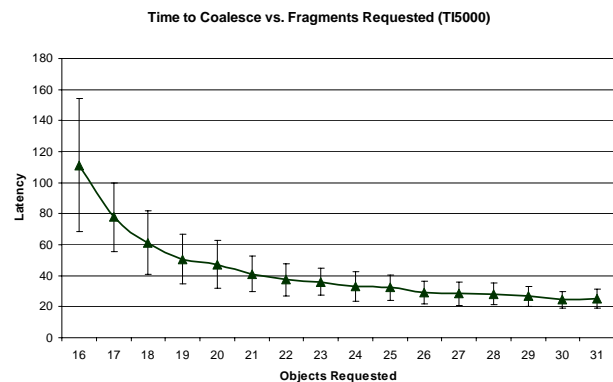
- Actual Web Cache running on OceanStore
  - Replica 1 far away
  - Replica 2 close to most requestors (created  $t \sim 20$ )
  - Replica 3 close to rest of requestors (created  $t \sim 40$ )

## Introspective Optimization



- Secondary tier self-organized into overlay multicast tree:
  - Presence of DOLR with locality to suggest placement of replicas in the infrastructure
  - Automatic choice between *update* vs *invalidate*
- Continuous monitoring of access patterns:
  - Clustering algorithms to discover object relationships
    - Clustered prefetching: demand-fetching related objects
    - Proactive-prefetching: get data there *before* needed
  - Time series-analysis of user and data motion
- Placement of Replicas to Increase Availability

## Statistical Advantage of Fragments



- Latency *and* standard deviation reduced:
  - Memory-less latency model
  - Rate  $\frac{1}{2}$  code with 32 total fragments

## Parallel Insertion Algorithms (SPAA '02)

- Massive parallel insert is important
  - We now have algorithms that handle "arbitrary simultaneous inserts"
  - Construction of nearest-neighbor mesh links
    - $\text{Log}^2 n$  message complexity  $\Rightarrow$  fully operational routing mesh
  - Objects kept available during this process
    - Incremental movement of pointers
- Interesting Issue: Introduction service
  - How does a new node find a gateway into the Tapestry?

## Can You Delete (Eradicate) Data?

- *Eradication* is antithetical to *durability!*
  - If you can eradicate something, then so can someone else! (denial of service)
  - Must have "eradication certificate" or similar
- Some answers:
  - Bays: limit the scope of data flows
  - Ninja Monkeys: hunt and destroy with certificate
- Related: Revocation of keys
  - Need hunt and re-encrypt operation
- Related: Version pruning
  - Temporary files: don't keep versions for long
  - Streaming, real-time broadcasts: Keep? Maybe
  - Locks: Keep? No, Yes, Maybe (auditing!)
  - Every key stroke made: Keep? For a short while?