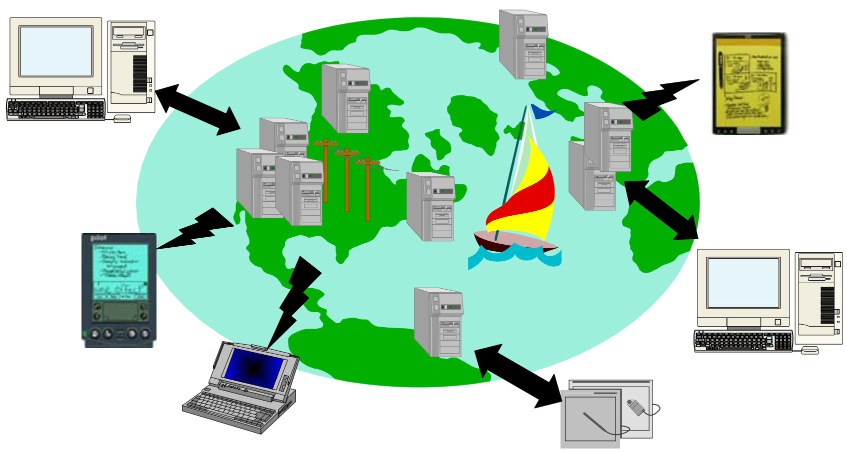
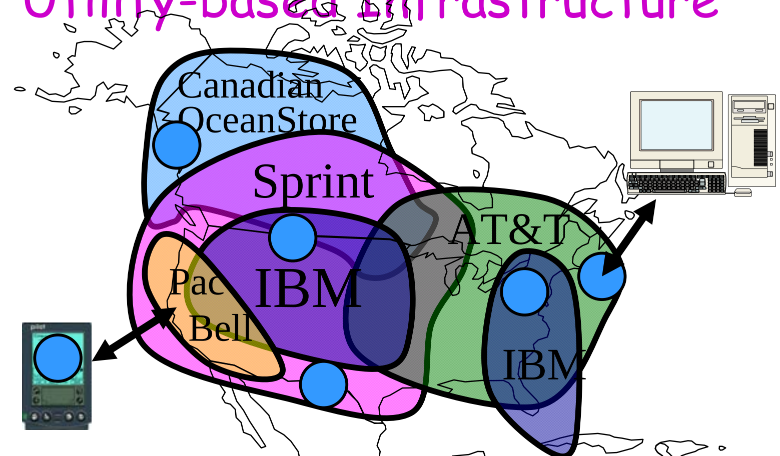
An OceanStore Retrospective



John Kubiatowicz
University of California at Berkeley

OceanStore Vision: Utility-based Infrastructure



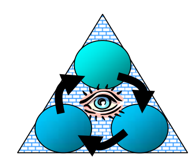
- Data service provided by storage federation
- Cross-administrative domain
- Contractual Quality of Service ("someone to sue")

What are the advantages of a utility?

- For Clients:
  - Outsourcing of Responsibility
    - Someone else worries about quality of service
  - Better Reliability
    - Utility can muster greater resources toward durability
    - System not disabled by local outages
    - Utility can focus resources (manpower) at security-vulnerable aspects of system
  - Better data mobility
    - Starting with secure network model⇒sharing
- For Utility Provider:
  - Economies of scale
    - Dynamically redistribute resources between clients
    - Focused manpower can serve many clients simultaneously

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 100s of years):
  - Geographic distribution of information
  - New servers added/Old servers removed
  - Continuous Repair ⇒ Data survives for long term

Why Peer-to-Peer?

Peer-to-Peer is:

- Old View:
  - A bunch of flakey high-school students stealing music
- New View:
  - A philosophy of systems design at extreme scale
  - Probabilistic design when it is appropriate
  - New techniques aimed at unreliable components
  - A rethinking (and recasting) of distributed algorithms
  - Use of Physical, Biological, and Game-Theoretic techniques to achieve guarantees

OceanStore Assumptions

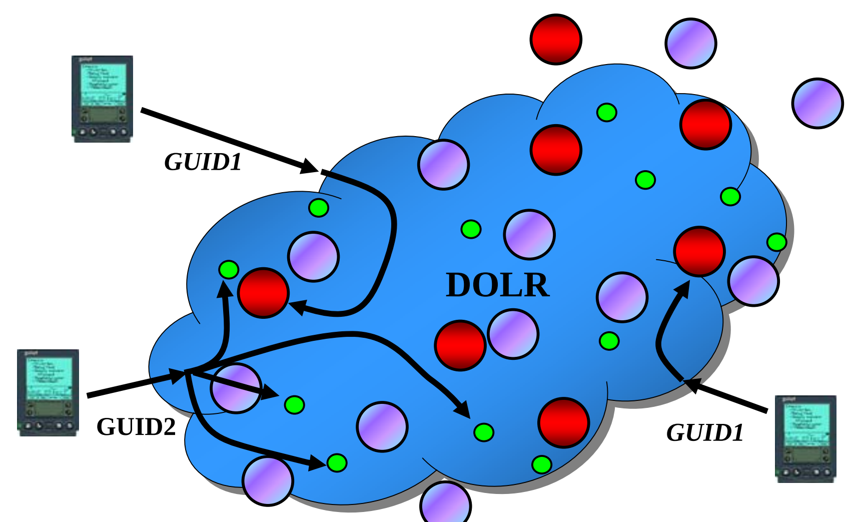
Peer-to-peer

- Untrusted Infrastructure:
  - The OceanStore is comprised of untrusted components
  - Individual hardware has finite lifetimes
  - All data encrypted within the infrastructure
- 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

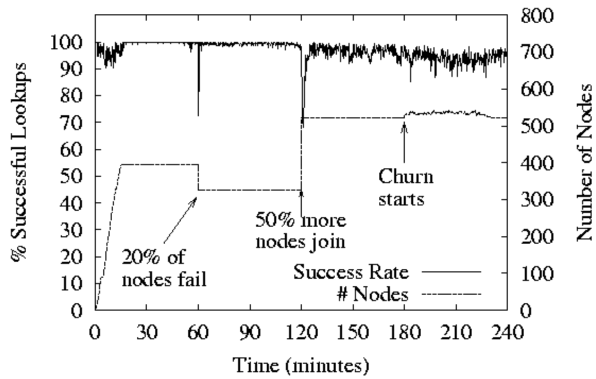
Quality-of-Service

- 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*

Important Peer-to-Peer Service:
Decentralized Object Location and Routing to Self-Verifying Handles (GUIDs)



The Tapestry DOLR: Peer-to-peer Stability



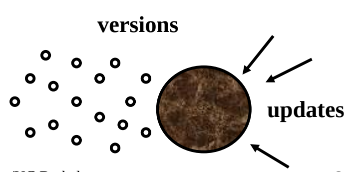
(May 2003: 1.5 TB over 4 hours)

DOLR Model generalizes to many simultaneous apps

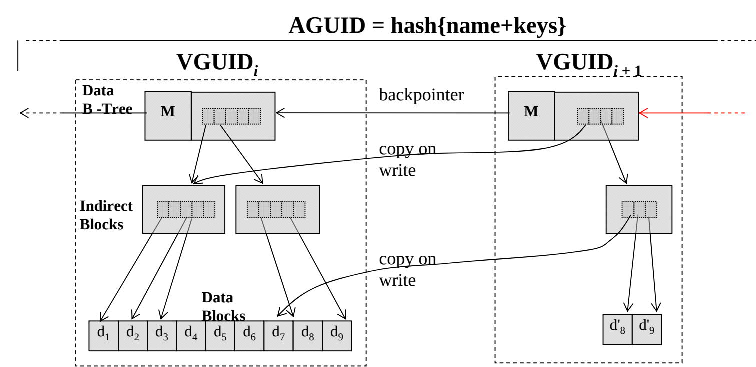
OceanStore Data Model

- 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

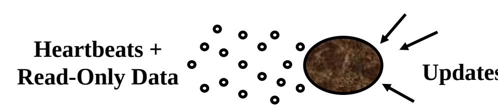
Comet Analogy



Self-Verifying Objects



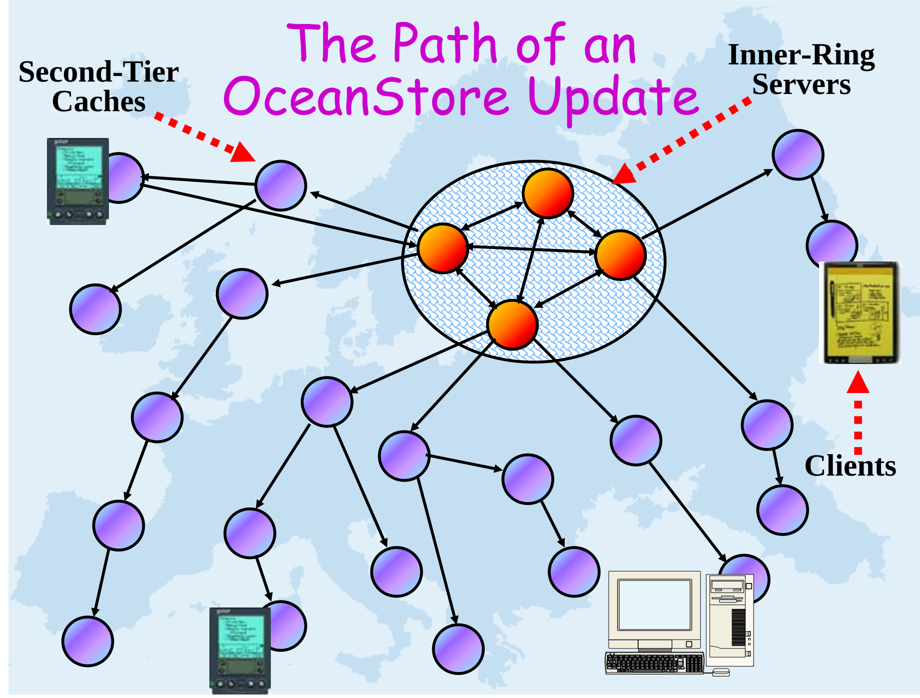
❤ **Heartbeat: {AGUID,VGUID, Timestamp}$_{signed}$**



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÷m (e.g 4)
  - Fragments are cryptographically self-verifying
- Most data in the OceanStore is archival!

The Path of an OceanStore Update



OceanStore API: Universal Conflict Resolution

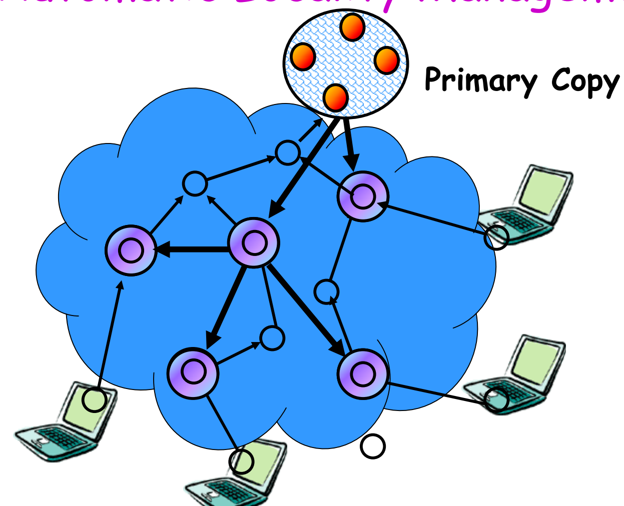
Native Clients | NFS/AFS | HTTP | IMAP/SMTP | NTFS (soon?)

OceanStore API

1. Conflict Resolution
2. Versioning/Branching
3. Access control
4. Archival Storage

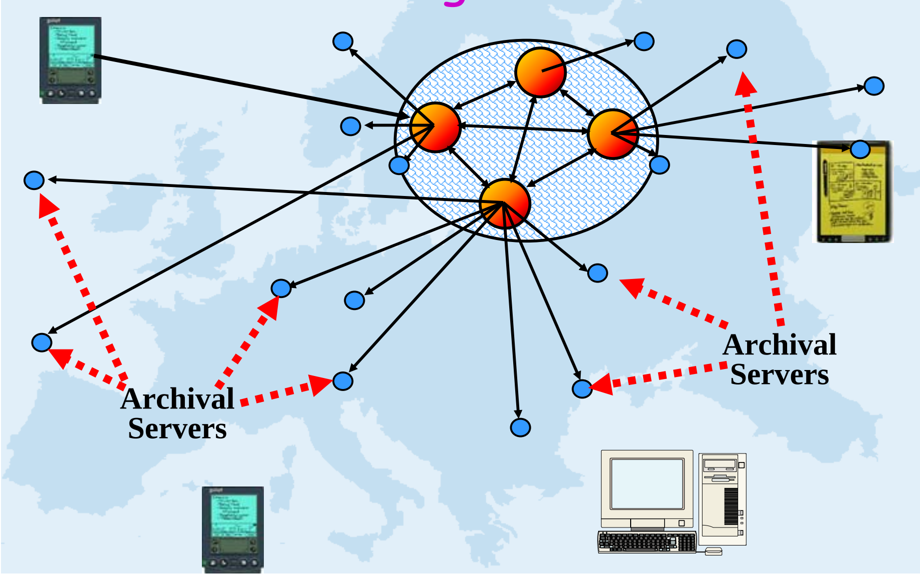
- Consistency is form of optimistic concurrency
  - Updates contain *predicate-action* pairs
  - Each predicate tried in turn:
    - If none match, the update is *aborted*
    - Otherwise, action of first true predicate is *applied*
- Role of Responsible Party (RP):
  - Updates submitted to RP which chooses total order

Peer-to-Peer Caching: Automatic Locality Management



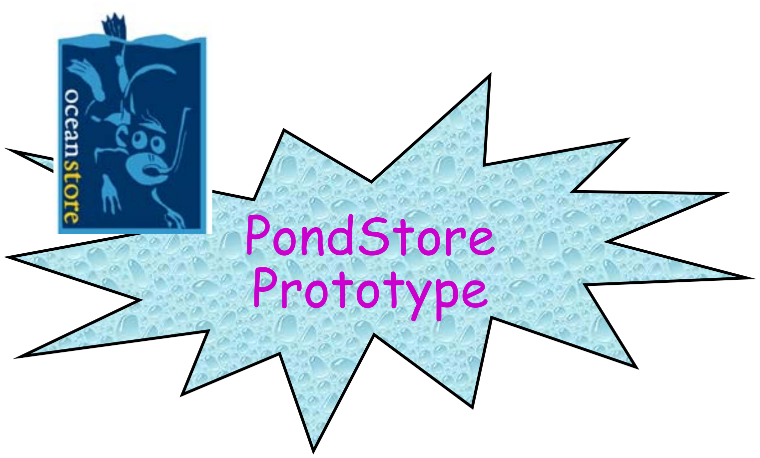
- Self-Organizing mechanisms to place replicas
- Automatic Construction of Update Multicast

Archival Dissemination of Fragments



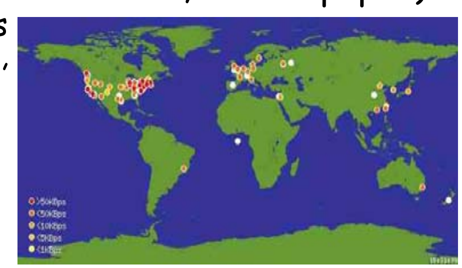
Extreme Durability

- 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 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

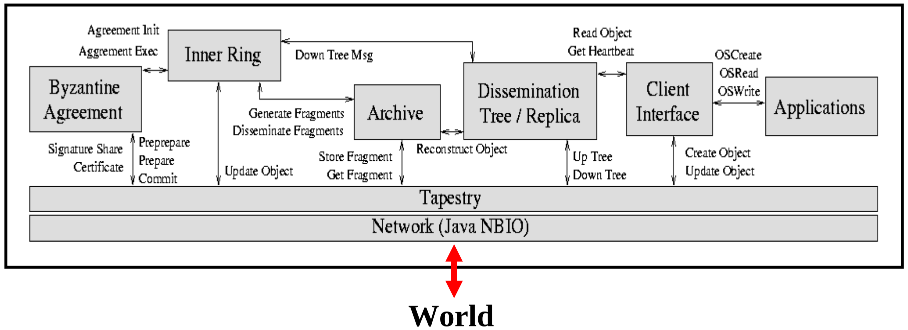


OceanStore Prototype

- All major subsystems operational
  - Self-organizing Tapestry base
  - Primary replicas use Byzantine agreement
  - Secondary replicas self-organize into multicast tree
  - Erasure-coding archive
  - Application interfaces: NFS, IMAP/SMTP, HTTP
- 280K lines of Java (J2SE v1.3)
  - JNI libraries for cryptography, erasure coding
- PlanetLab Deployment (FAST 2003, "Pond" paper)
  - 220 machines at 100 sites in North America, Europe, Australia, Asia, etc.
  - 1.26Ghz PIII (1GB RAM), 1.8Ghz PIV (2GB RAM)
  - OceanStore code running with 1000 virtual-node emulations



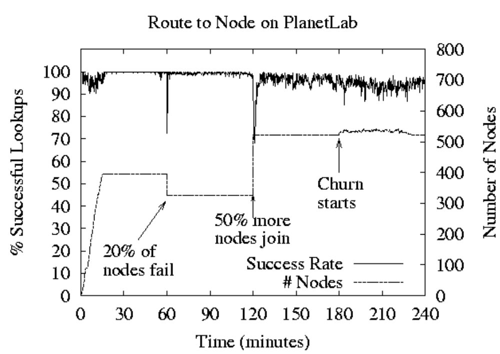
Event-Driven Architecture of an OceanStore Node



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

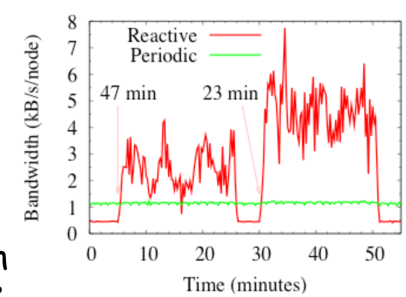
Problem #1: DOLR is Great Enabler— but only if it is stable

- Had Reasonable Stability:
  - In simulation
  - Or with small error rate
- But trouble in wide area:
  - Nodes might be lost and never reintegrate
  - Routing state might become stale or be lost
- Why?
  - Complexity of algorithms
  - Wrong design paradigm: strict rather than loose state
  - Immediate repair of faults
- Ultimately, Tapestry Routing Framework succumbed to:
  - Creeping Featurism (designed by several people)
  - Fragilility under churn
  - Code Bloat



Answer: Bamboo!

- Simple, Stable, Targeting Failure
- Rethinking of design of Tapestry:
  - Separation of correctness from performance
  - Periodic recovery instead of reactive recovery
  - Network understanding (e.g. timeout calculation)
  - Simpler Node Integration (smaller amount of state)
- Extensive testing under Churn and partition
- Bamboo is so stable that it is part of the OpenHash public DHT infrastructure.
- In wide use by many researchers



Problem #2: Pond Write Latency

- Byzantine algorithm adapted from Castro & Liskov
  - Gives fault tolerance, security against compromise
  - Fast version uses symmetric cryptography
- Pond uses threshold signatures instead
  - Signature proves that *f* +1 primary replicas agreed
  - Can be shared among secondary replicas
  - Can also change primaries w/o changing public key
- Big plus for maintenance costs
  - Results good for all time once signed
  - Replace faulty/compromised servers transparently

Closer Look: Write Cost

- Small writes
  - Signature dominates
  - Threshold sigs. slow!
  - Takes 70+ ms to sign
  - Compare to 5 ms for regular sigs.
- Large writes
  - Encoding dominates
  - Archive cost per byte
  - Signature cost per write

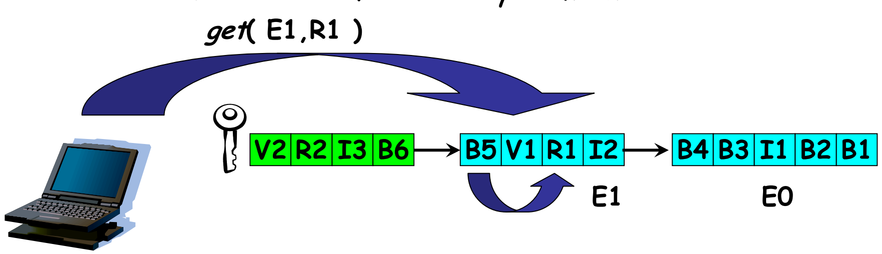
Phase	4 kB write	2 MB write
Validate	0.3	0.4
Serialize	6.1	26.6
Apply	1.5	113.0
Archive	4.5	566.9
Sign Result	77.8	75.8

(times in milliseconds)

- Answer: Reduction in overheads
  - More Powerful Hardware at Core
  - Cryptographic Hardware
    - Would greatly reduce write cost
    - Possible use of ECC or other signature method
  - Offloading of Archival Encoding

Problem #3: Efficiency

- No resource aggregation
  - Small blocks spread widely
  - Every block of every file on different set of servers
  - Not uniquely OceanStore issue!
- Answer: Two-Level Naming
  - Place data in larger chunks ('extents')
  - Individual access of blocks by name within extents



  - Bonus: Secure Log good interface for secure archive
  - Antiquity: New Prototype for archival storage

Problem #4: Complexity

- Several of the mechanisms were complex
  - Ideas were simple, but implementation was complex
  - Data format combination of live and archival features
  - Byzantine Agreement hard to get right
- Ideal layering not obvious at beginning of project:
  - Many Applications Features placed into Tapestry
  - Components not autonomous, i.e. able to be tied in at any moment and restored at any moment
- Top-down design lost during thinking and experimentation
- Everywhere: reactive recovery of state
  - Original Philosophy: Get it *right* once, then repair
  - Much Better: keep working toward ideal (but assume never make it)

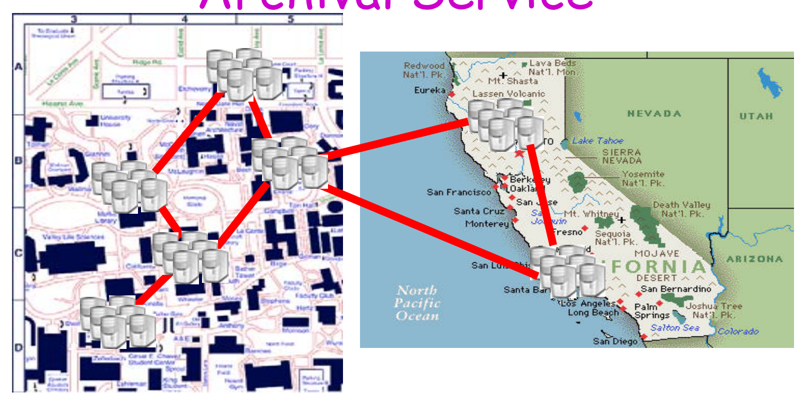
Other Issues/Ongoing Work:

- Archival Repair Expensive if done incorrectly:
  - Small blocks consume excessive storage and network bandwidth
  - Transient failures consume unnecessary repair bandwidth
  - Solutions: collect blocks into *extents* and use *threshold* repair
- Resource Management Issues
  - Denial of Service/Over Utilization of Storage serious threat
  - Solution: Exciting new work on fair allocation
- Inner Ring provides incomplete solution:
  - Complexity with Byzantine agreement algorithm is a problem
  - Working on better Distributed key generation
  - Better Access control + secure hardware + simpler Byzantine Algorithm?
- Handling of low-bandwidth links and Partial Disconnection
  - Improved efficiency of data storage
  - Scheduling of links
  - Resources are *never* unbounded
- Better Replica placement through game theory

Bamboo ⇒ OpenDHT

- PL deployment running for several months
- Put/get via RPC over TCP
- Looking for new users/New applications

The Berkeley PetaByte Archival Service



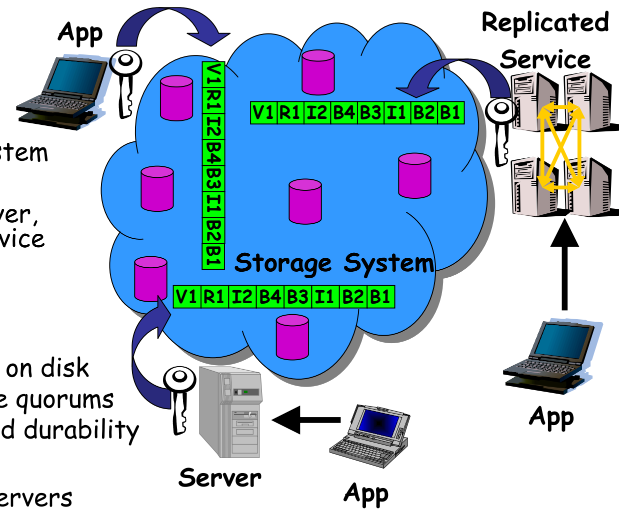
- OceanStore Concepts Applied to Tape-less backup
  - Self-Replicating, Self-Repairing, Self-Managing
  - No need for actual Tape in system
    - (Although could be there to keep with tradition)

OceanStore Archive ⇒ Antiquity

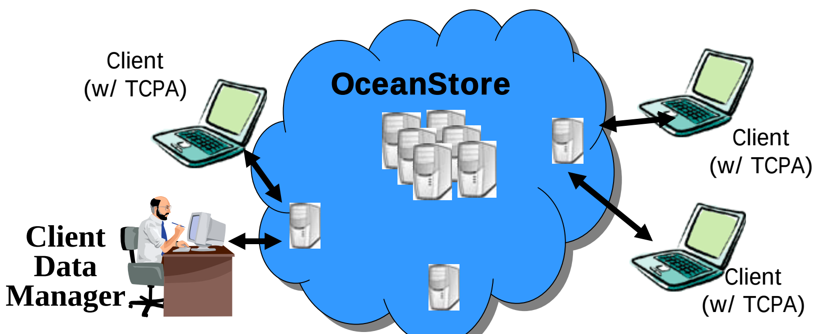
- Secure Log:
  - Can only modify at one point – log head.
    - Makes consistency easier
  - Self-verifying
    - Every entry securely points to previous forming *Merkle* chain
    - Prevents substitution attacks
  - Random read access – can still read efficiently
- Simple and secure primitive for storage
  - Log identified by cryptographic key pair
  - Only owner of private key can modify log
  - Thin interface, only *append*()
- Amenable to secure, durable implementation
  - Byzantine quorum of storage servers
    - Can survive failures at O(n) cost instead of $O(n^2)$ cost
  - Efficiency through aggregation
    - Use of Extents and Two-Level naming

Antiquity Architecture: Universal Secure Middleware

- Data Source
  - Creator of data
- Client
  - Direct user of system
    - "Middleware"
    - End-user, Server, Replicated service
  - *append*()'s to log
  - Signs requests
- Storage Servers
  - Store log replicas on disk
  - Dynamic Byzantine quorums
    - Consistency and durability
- Administrator
  - Selects storage servers
- Prototype currently operational on PlanetLab



Secure Object Storage



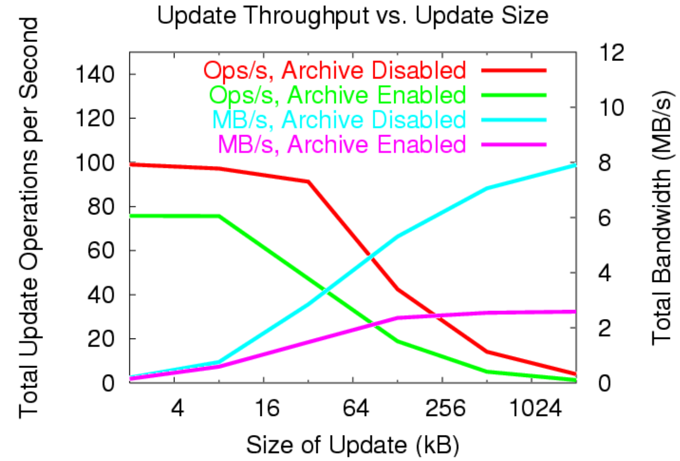
- Security: Access and Content controlled by client
  - Privacy through data encryption
  - Optional use of cryptographic hardware for revocation
  - Authenticity through hashing and active integrity checking
- Flexible self-management and optimization:
  - Performance and durability
  - Efficient sharing

For more info:
http://oceanstore.org

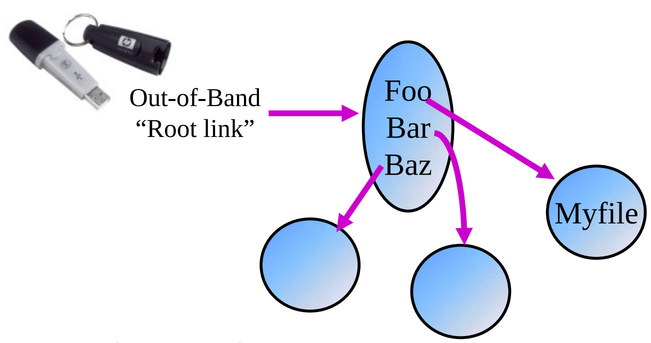
- OceanStore vision paper for ASPLOS 2000
  "OceanStore: An Architecture for Global-Scale Persistent Storage"
- Pond Implementation paper for FAST 2003
  "Pond: the OceanStore Prototype"
- Tapestry deployment paper (JSAC, to appear)
  "Tapestry: A Resilient Global-scale Overlay for Service Deployment"
- Bamboo Paper for Usenix 2004
  "Handling Churn in a DHT"
- OpenDHT Paper for SigCOMM 2005
  "OpenDHT: A Public DHT Service"

Backup Slides

Closer Look: Write Cost



Secure Naming



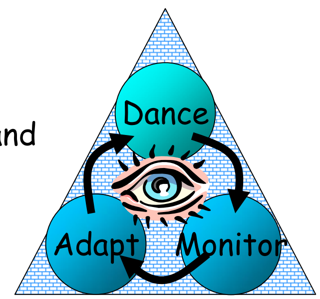
- Naming hierarchy:
  - Users map from names to GUIDs via hierarchy of OceanStore objects (*ala SDSI*)
  - Requires set of "root keys" to be acquired by user

The Thermodynamic Analogy

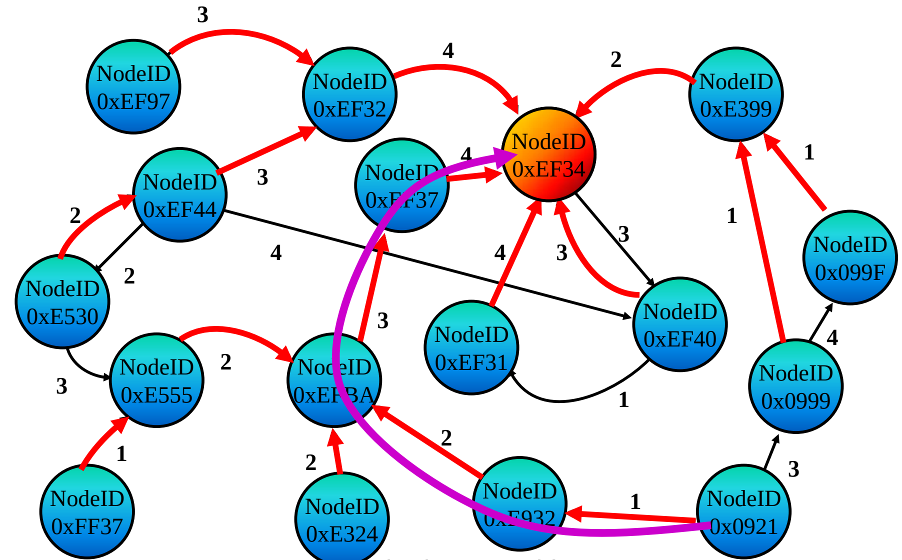
- Large Systems have a variety of *latent order*
  - Connections between elements
  - Mathematical structure (erasure coding, etc)
  - Distributions peaked about some desired behavior
- Permits "Stability through Statistics"
  - Exploit the behavior of aggregates (redundancy)
- Subject to Entropy
  -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

The Biological Inspiration

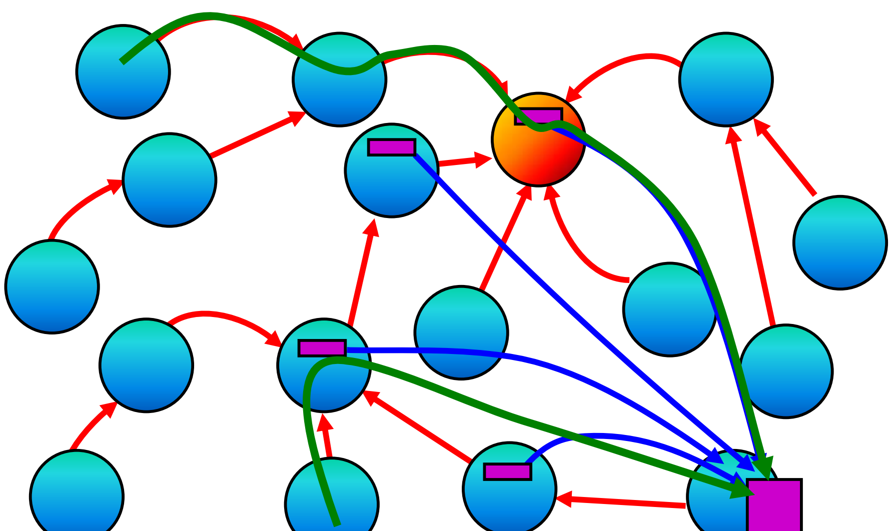
- Biological Systems are built from (extremely) faulty components, yet:
  - They operate with a variety of component failures ⇒ Redundancy of function and representation
  - They have stable behavior ⇒ Negative feedback
  - They are self-tuning ⇒ Optimization of common case
- Introspective (Autonomic) Computing:
  - Components for performing
  - Components for monitoring and model building
  - Components for continuous adaptation



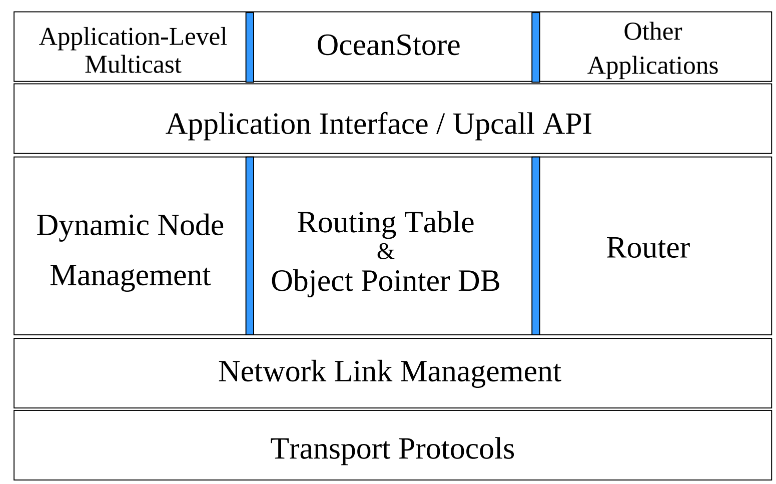
Basic Tapestry Mesh
Incremental Prefix-based Routing



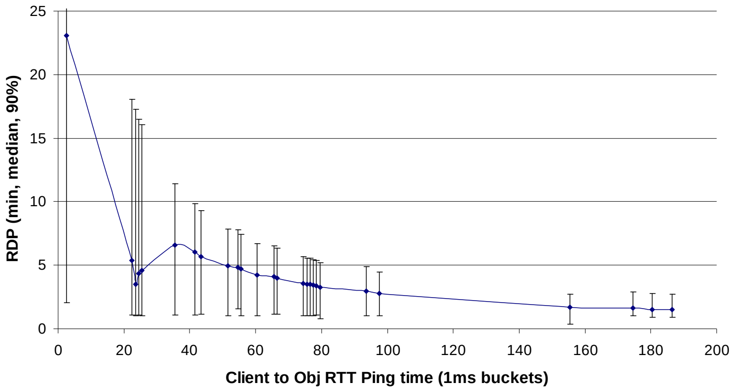
Use of Tapestry Mesh
Randomization *and* Locality



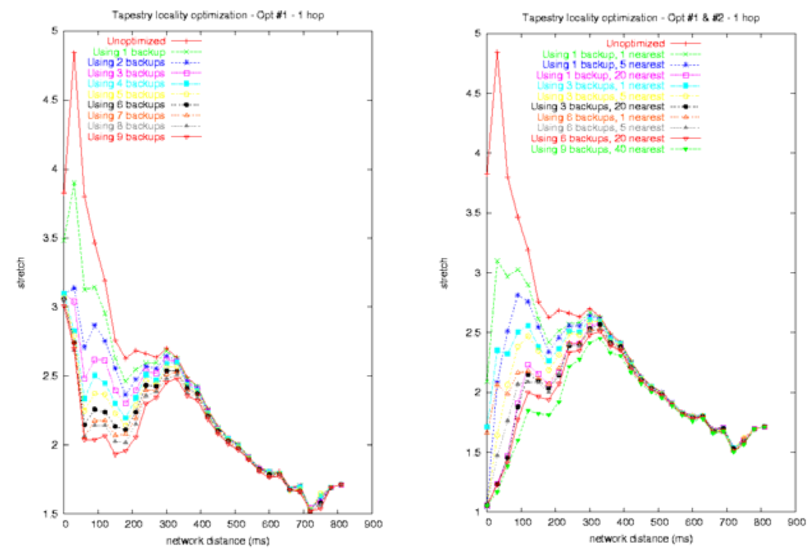
Single Node Tapestry



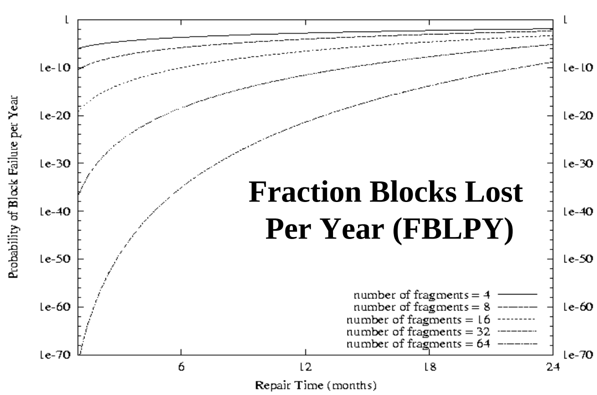
Object Location



Tradeoff: Storage vs Locality

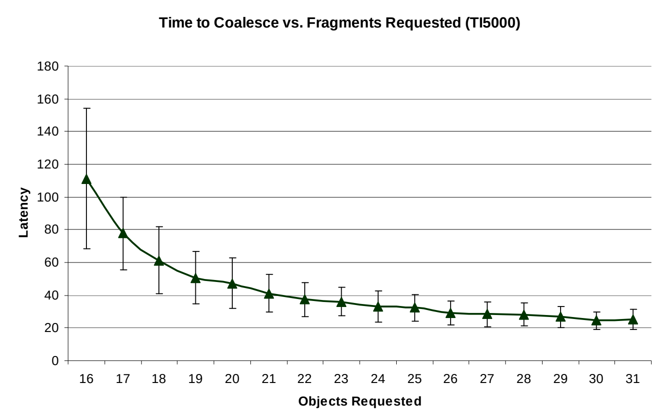


Aside: Why erasure coding? High Durability/overhead ratio!



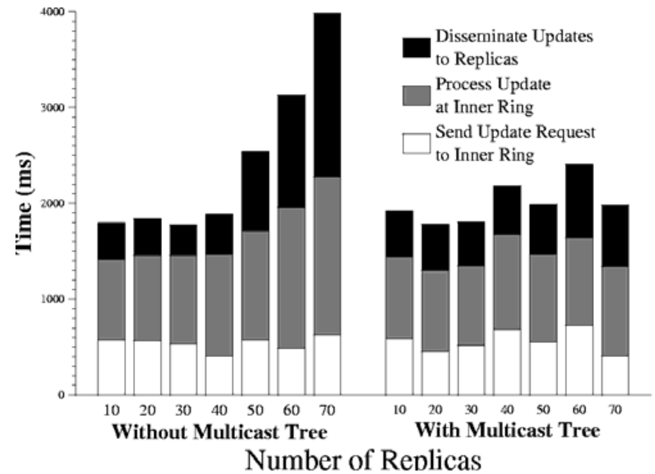
- Exploit law of large numbers for durability!
- 6 month repair, FBLPY:
  - Replication: 0.03
  - Fragmentation: 10^{-35}

Statistical Advantage of Fragments

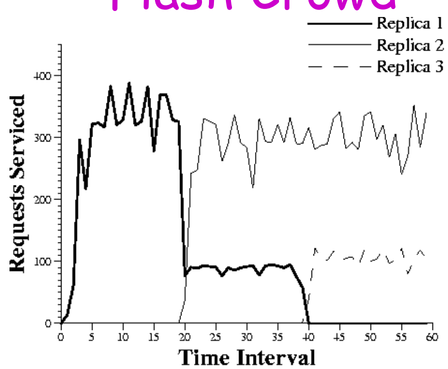


- Latency *and* standard deviation reduced:
  - Memory-less latency model
  - Rate ½ code with 32 total fragments

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 ~ 20)
  - Replica 3 close to rest of requestors (created t ~ 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

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
    - $\log^2 n$ 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?