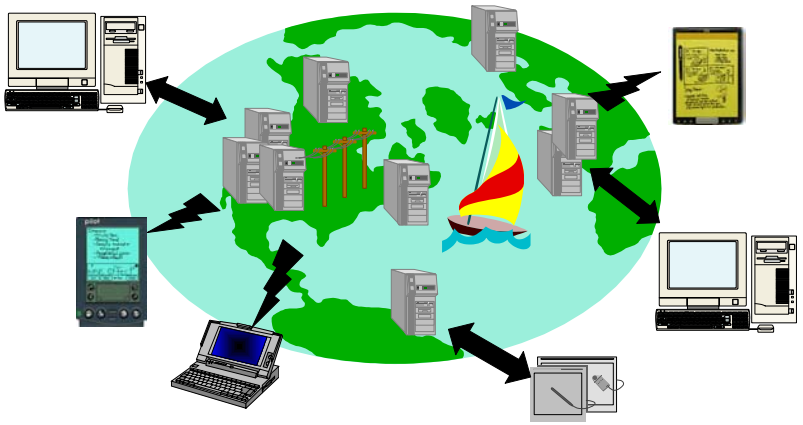
OceanStore

Theoretical Issues and Open Problems



John Kubiatowicz
University of California at Berkeley

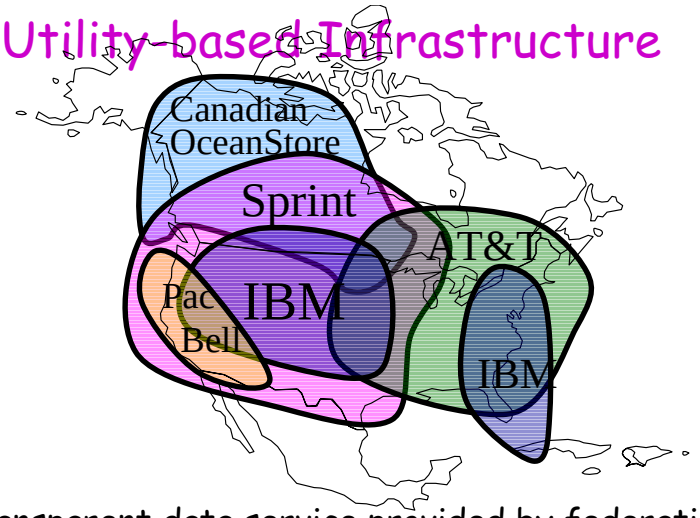
OceanStore Context: Ubiquitous Computing

- 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

Questions about information:

- Where is persistent information stored?
  - *Want: Geographic independence for availability, durability, and freedom to adapt to circumstances*
- How is it protected?
  - *Want: Encryption for privacy, signatures for authenticity, and Byzantine commitment for integrity*
- Can we make it indestructible?
  - *Want: Redundancy with continuous repair and redistribution for long-term durability*
- Is it hard to manage?
  - *Want: automatic optimization, diagnosis and repair*
- Who owns the aggregate resouces?
  - *Want: Utility Infrastructure!*

Utility-based Infrastructure



- Transparent data service provided by federation of companies:
  - Monthly fee paid to one service provider
  - Companies buy and sell capacity from each other

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 (1.5×10^{18})

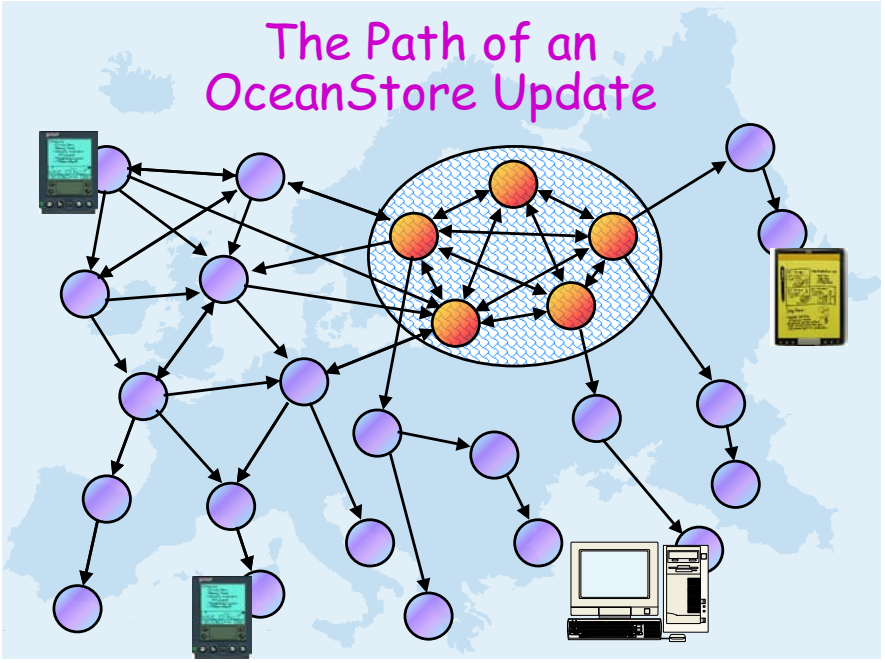
OceanStore Assumptions

- Untrusted Infrastructure:
  - The OceanStore is comprised of untrusted components
  - Only ciphertext 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
- Optimistic Concurrency via Conflict Resolution:
  - Avoid locking in the wide area
  - Applications use object-based interface for updates

Game Theoretic Issues?

- Mechanism design to achieve optimal behavior from service providers
- Resource allocation algorithms
  - Quality of service contracts?
  - Proof of data (challenge/response protocol?)
  - Proof of resources?
  - Control over redundancy?
  - Introspection?

The Path of an OceanStore Update



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

Oblivious Updates on Encrypted Data?

- Tentative Scheme:
  - Divide data into small blocks
  - Updates on a per-block basis
  - Predicates derived from techniques for searching on encrypted data

TimeStamp
Client ID
{Pred1, Update1}
{Pred2, Update2}
{Pred3, Update3}
Client Signature

- Challenges:
  - Better Predicates on Encrypted Data
  - The Merge problem:
    - Insert a small block of data into a larger block on a server which doesn't have the encryption key!

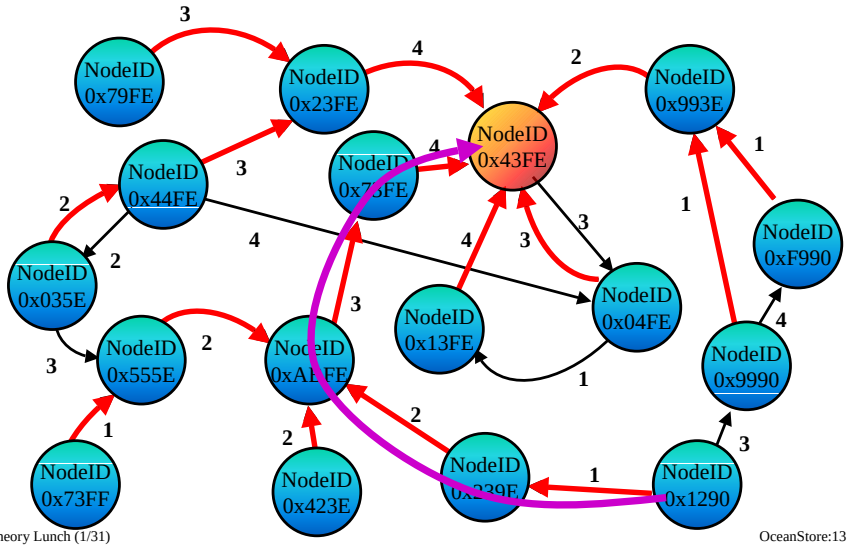
Other Challenges

- Reencryption on insecure server
  - Existance of a *one-way* function, *f* [*Key1, Key2*](Data):
    - Transforms data encrypted with Key 1 into data encrypted with Key 2
    - Can't derive Key 1 or Key 2 from *f* [*Key1, Key2*]
    - Can't derive *f* [*Key2, Key1*] from *f* [*Key1, Key2*]
- Better Byzantine Algorithm:
  - Inner ring takes all updates, serializes them, applies their predicate/update pairs, then signs the result
  - Current scheme derived from Liskov and Castro
    - Has single server in inner ring that acts like general – must be reelected when bad
    - Result must be *signed* by aggregate. Using Threshold signatures – at the moment, this adds another phase!
  - Is there a decentralized algorithm?
    - Any update can be sent to any ring member
    - Assume presence of synchronized clocks (We have been looking at periodic heart beats
    - Must keep update traffic and latency low

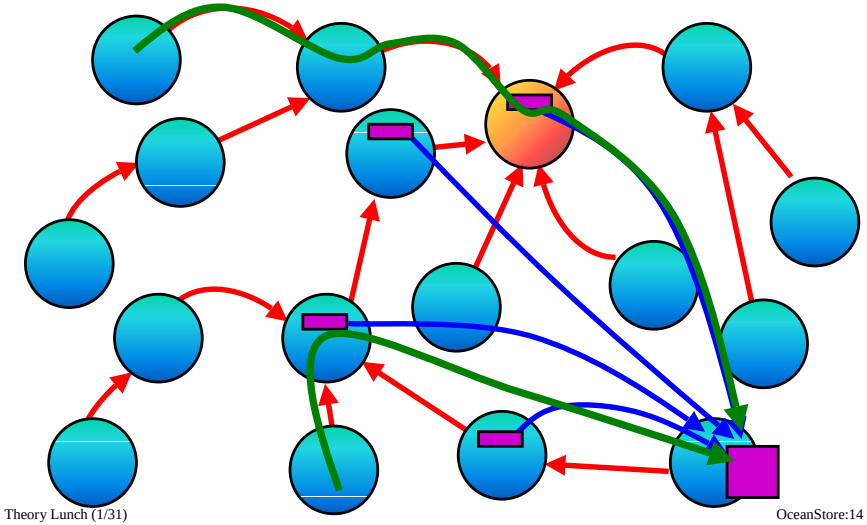
Two-levels of Routing

- Fast, probabilistic search for "routing cache":
  - Built from *attenuated* bloom filters
  - Approximation to gradient search potential function
- Redundant *Plaxton Mesh* used for underlying routing infrastructure:
  - Randomized data structure with locality properties
  - Redundant, insensitive to faults, and repairable
  - Amenable to continuous adaptation to adjust for:
    - Changing network behavior
    - Faulty servers
    - Denial of service attacks

Basic Plaxton Mesh
Incremental suffix-based routing



Use of Plaxton Mesh
Randomization *and* Locality



Use of the Plaxton Mesh
(the Tapestry infrastructure)

- OceanStore enhancements for reliability:
  - Each node has multiple neighbors with given properties
  - Documents have multiple roots through salted hashes
    - Multiple paths to a given document
    - Multiple roots responsible for welfare of document
  - Searches along multiple paths simultaneously
    - Tradeoff between reliability and bandwidth utilization?
  - Routing-level validation of query results
- Dynamic node insertion and deletion algorithms
  - Uses redundant neighbors during integration
  - Continuous repair and incremental optimization of links

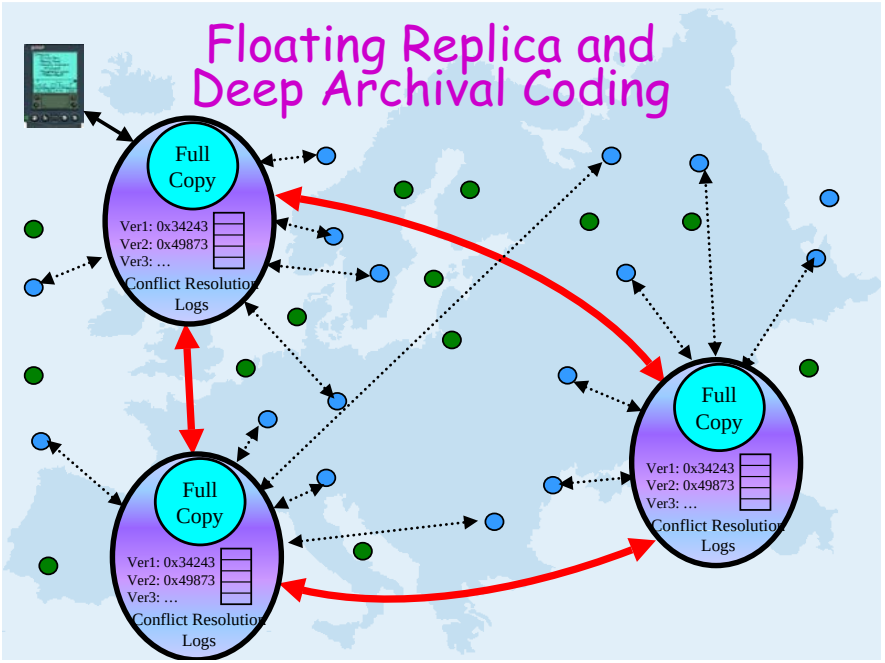
Questions for routing?

- Analysis of Tapestry properties
  - Routing Stretch?
  - Optimality of insertion algorithms?
  - Fault Tolerance
    - What is the tradeoff between redundancy of searches and tolerance of faults?
    - What is the best (optimal number of messages, etc) use of redundant info in Tapestry under failures?
  - Repair bandwidth/mechanisms
    - What are the lowest bandwidth repair techniques?
    - Least subject to attack?
- Better probabilistic potential functions
  - Goal: Compact Summary of neighbors which can be used to drive an incremental search

Data Coding Model

- Two distinct forms of data: active and archival
- *Active Data* in Floating Replicas
  - Per object virtual server
  - Logging for updates/conflict resolution
  - Interaction with other replicas to keep data consistent
  - May appear and disappear like bubbles
- *Archival Data* in Erasure-Coded Fragments
  - OceanStore equivalent of stable store
  - During commit, previous version coded with erasure-code and spread over 100s or 1000s of nodes
  - Fragments are self-verifying
  - Advantage: *any* 1/2 or 1/4 of fragments regenerates data
    - Law of large numbers advantage to fragmentation
- Most data in the OceanStore is archival!

Floating Replica and Deep Archival Coding



Archival Storage Issues

- Archival Properties
  - Behavior under correlated failures
  - Derivation of correlated failure model of servers which can be used to
  - Refresh behavior which consumes least "energy"
- General Question: Thermodynamic description of OceanStore
  - Entropy management
  - General tradeoff between redundancy and rate of information retrieval

For more info:

- OceanStore vision paper for ASPLOS 2000
  "OceanStore: An Architecture for Global-Scale Persistent Storage"
- OceanStore web site:
  http://oceanstore.cs.berkeley.edu/