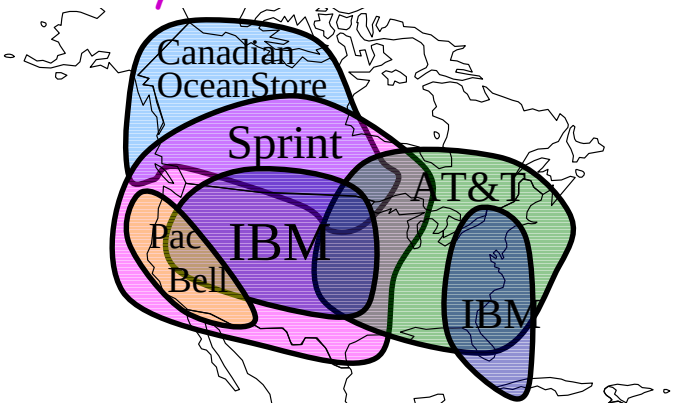
Opportunities for Continuous Tuning in a Global Scale File System

John Kubiatowicz
University of California at Berkeley

The OceanStore Premise: A Utility-based Infrastructure



- Data service provided by federation of companies:
  - Millions or Billions of clients, servers, routers
  - Companies buy and sell capacity from each other

 ©2002 John Kubiatowicz/UC Berkeley

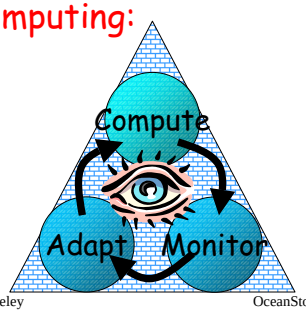
Key Observation: Need Automatic Maintenance

- Can't possibly manage billions of servers by hand!
  - Too many knobs to adjust
- System should automatically:
  - Adapt to failure and attack
  - Repair itself
  - Incorporate new elements
  - Remove faulty elements
- Self Tuning:
  - Placement of Data: Spatial Locality
  - Routing of requests: Network Locality
  - Scheduling of Data movement: Proactive Prefetching

 ©2002 John Kubiatowicz/UC Berkeley

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



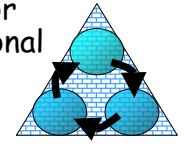
 ©2002 John Kubiatowicz/UC Berkeley

Outline

- Motivation
- Some Properties of OceanStore
- Specific Opportunities for Autonomic Computing:
  - Replica Management
  - Communication Infrastructure
  - Deep Archival Storage
- Conclusion

©2002 John Kubiatowicz/UC Berkeley

The New Architectural Creed



- Question: Can we use Moore's law gains for something other than just raw computational performance?
- Examples:
  - Online algorithmic validation
  - Model building for data rearrangement
    - Availability
    - Better prefetching
  - Extreme Durability (1000-year time scale?)
    - Use of erasure coding and continuous repair
  - Stability through Statistics
    - Use of redundancy to gain more predictable behavior
    - *Systems version of Thermodynamics!*
  - Continuous Dynamic Optimization of other sorts
- Berkeley Term for Autonomic Computing:
  - Introspective computing!

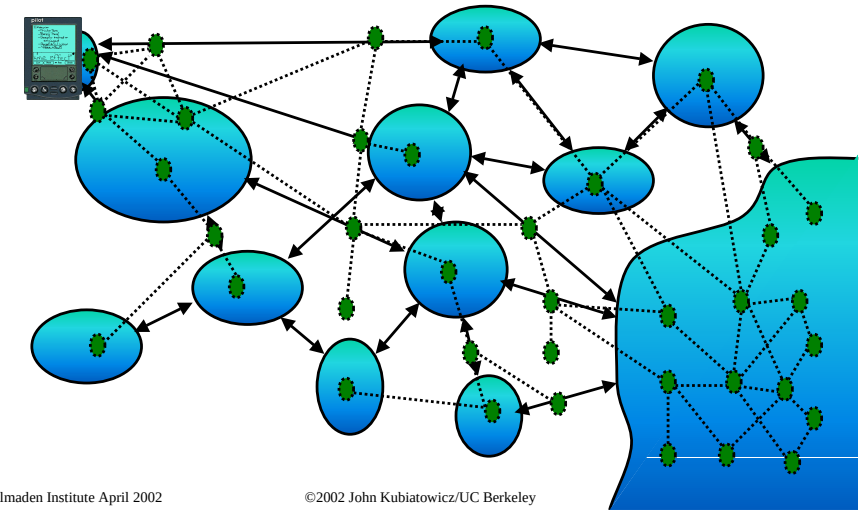
©2002 John Kubiatowicz/UC Berkeley

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

©2002 John Kubiatowicz/UC Berkeley

Basic Structure: Irregular Mesh of "Pools"

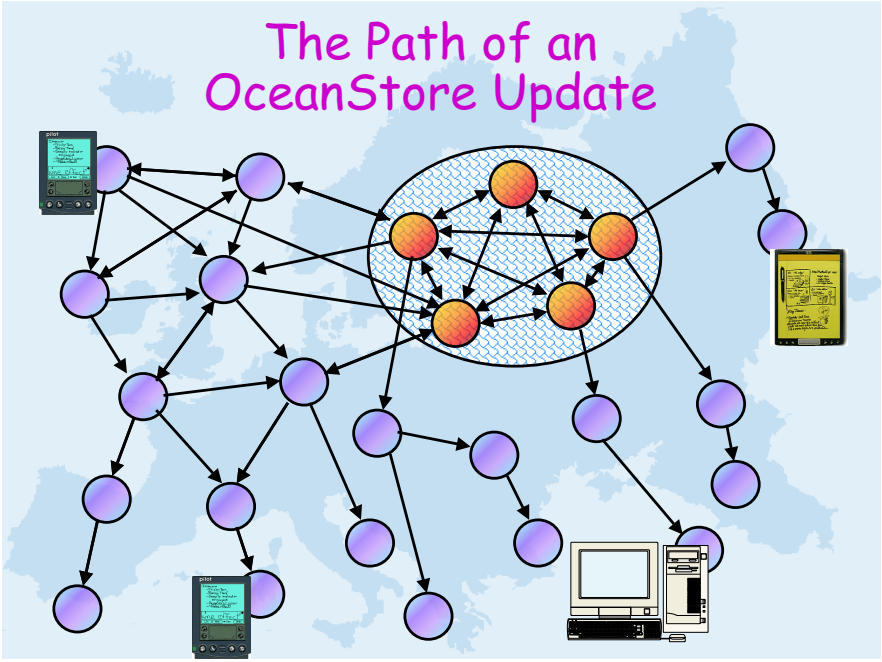


©2002 John Kubiatowicz/UC Berkeley

Bringing Order to this Chaos

- How do you *ensure consistency*?
  - Must scale and handle intermittent connectivity
  - Must prevent unauthorized update of information
- How do you *name* information?
  - Must provide global uniqueness
- How do you *find* information?
  - Must be scalable and provide maximum flexibility
- How do you *protect* information?
  - Must preserve privacy
  - Must provide *deep archival storage*
- How do you *tune* performance?
  - Locality very important

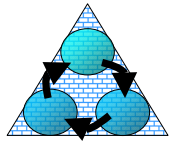
Throughout all of this: how do you maintain it???



OceanStore Consistency via Conflict Resolution

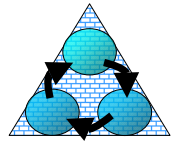
- Consistency is form of optimistic concurrency
  - An update packet contains a series of *predicate-action* pairs which operate on encrypted data
  - Each predicate tried in turn:
    - If none match, the update is *aborted*
    - Otherwise, action of first true predicate is *applied*
- Role of Responsible 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

Tunable Components



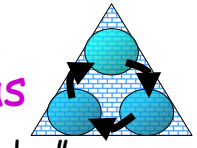
- Primary (Inner ring) replicas:
  - Where are they?
  - Which servers are stable enough?
- Second Tier Replicas
  - How many active replicas?
  - Where are they?
  - When are they present?
- Communication Infrastructure
  - What is the path of requests and updates?
  - How is the multicast update tree built?
  - What level of redundancy needed?
- Update Management
  - Are all updates pushed to all replicas?
  - Proactive push of updates v.s. selective invalidation

Specific Examples: Inner Ring



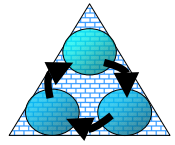
- Byzantine Commitment for inner ring:
  - Can tolerate up to 1/3 faulty servers in inner ring
    - Bad servers can be arbitrarily bad
    - Cost ~n^2 communication
  - Can we detect misbehaving servers?
    - Markov models of "good" servers.
  - Continuous refresh of set of inner-ring servers
    - Reinstall and rotate physical servers
- Automatic tuning of:
  - Update size and groupings
  - Recognition of access patterns to prefetch data?

Specific Examples: Second Tier Replicas



- On demand fetching of replica "header"
  - Move some copy close to where it is being used
  - Use of data location infrastructure to suggest location for a replica
- Proactive Prefetching of Information
  - Hidden Markov model of user behavior: clusters
    - Clustering of data items
    - Proactive prefetching of data close to user
    - Get the data there *before* the user needs it!
  - Other clustering techniques
  - Time-series analysis (Kalman filters)
    - Data used every Tuesday at 3:00
    - Or: Professor Joe starts mornings in café, afternoons in office

Specific Examples: Multicast Overlay



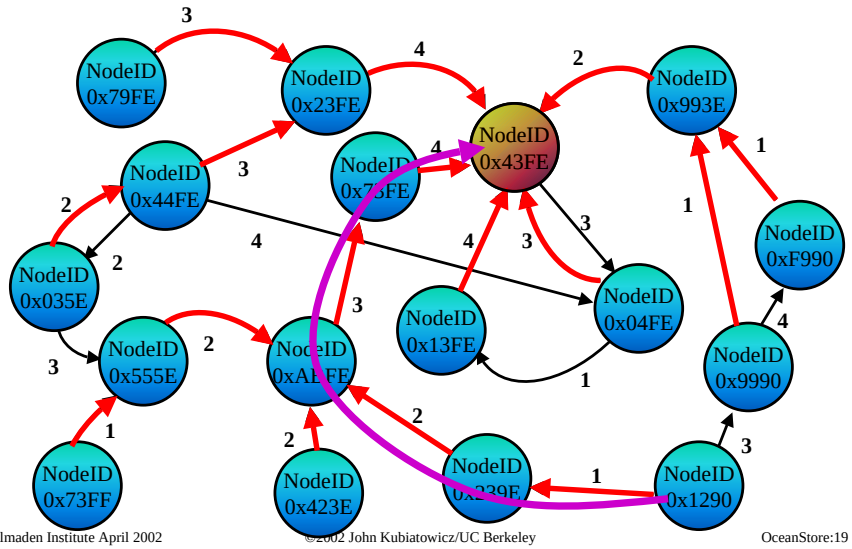
- Built between second-tier replicas
  - Restricted fanout, shortest path connections
  - Simultaneous placement and tree building
  - Self-adapting: must rebuild if parent fails
- Update v.s. Invalidate?
  - Build tree with inclusion property
  - Parents make decision what to forward to children
  - Low-bandwidth children get minimal traffic
- Streaming v.s. Block Access
  - Can second-tier adapt by keeping just ahead?

Routing and Data Location

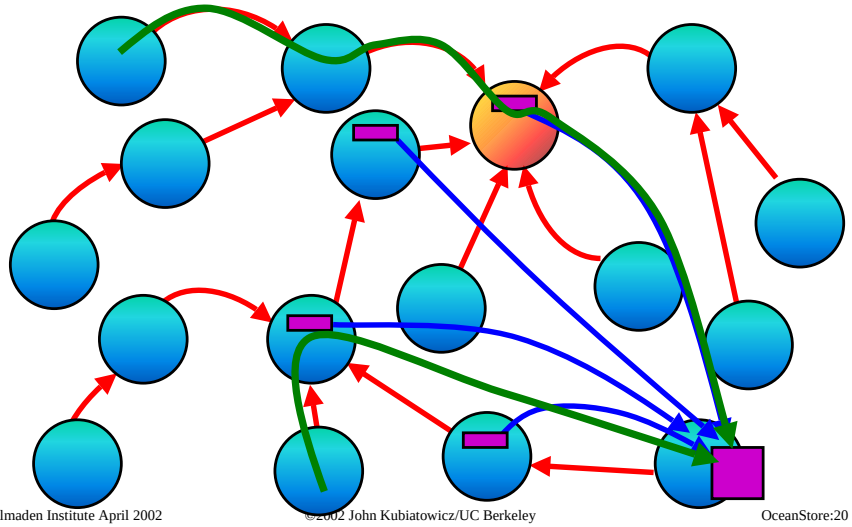
- Requirements:
  - Find data quickly, wherever it might reside
  - Insensitive to faults and denial of service attacks
  - Repairable infrastructure
    - Easy to reconstruct routing and location information
- Technique: Combined Routing and Data Location
  - Packets are addressed to data (GUIDs), not locations
  - Infrastructure routes packets to destinations and verifies that servers are behaving

Basic Plaxton Mesh

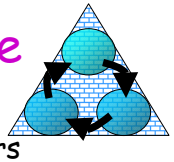
Incremental suffix-based routing



Use of Plaxton Mesh Randomization *and* Locality



Automatic Maintenance

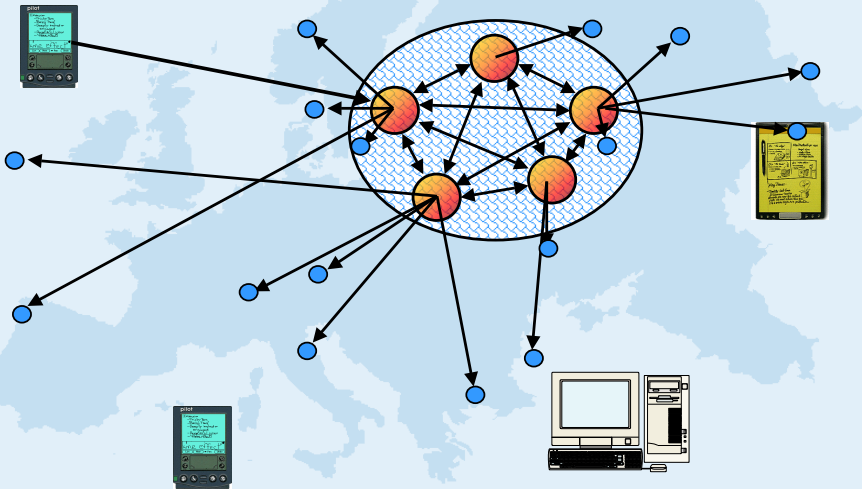


- All Tapestry state is Soft State
  - Continuous probing, selection of neighbors
  - Periodic restoration of state
- Dynamic insertion:
  - New nodes contact small number of existing nodes
  - Integrate themselves automatically
  - Later, data just flows to new servers
- Dynamic deletion:
  - Node detected as unresponsive
  - Pointer state routed around faulty node (signed deletion requests authorized by servers holding data)
- Markov Models again:
  - What is a misbehaving router? Communication link?
  - What level of redundancy necessary?
  - Are we under attack?

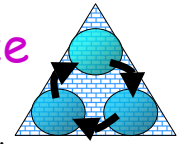
Deep Archival Storage

Archival Dissemination of Erasure Coded Fragments

(Better coding than RAID)

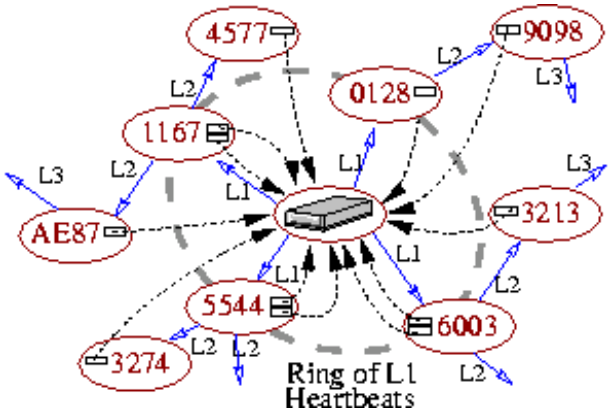


Automatic Maintenance



- Introspective Analysis of Servers
  - Fragments must be sent to servers which fail independently
  - OceanStore server model building: Select independent server sets based on history!
  - Mutual Information analysis: correlated downtime
- Level of Redundancy adapted to match infrastructure
- Continuous sweep through data?
  - Expensive, but possible if infrequent
- Distributed state management
  - Use Tapestry routing infrastructure to track replicas
  - Efficient heartbeat from server of data to
  - Infrastructure notices need to repair information

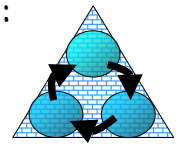
Example of Global Heartbeats



- Tapestry pointers can direct traffic
- Exponential backoff on TTL (ring level)

Final Word: Reputation Management

- Techniques needed to evaluate:
  - Servers
  - Organizations
  - Routers
  - ETC…
- Examples:
  - Does company X store data when they say they will?
  - Is router Y advertising a real path?
  - Is server Z a reliable place to put data?
- Information vital:
  - Affects payment
  - Placement
  - Level of redundancy

OceanStore Conclusions

- OceanStore: everyone's data, one big utility
  - Global Utility model for persistent data storage
- Billions of Servers, Moles of Bytes
  - Autonomic Computing is Inevitable!
- Many opportunities for tuning and autonomic repair of global-scale infrastructures:
  - Replica placement
  - Analysis of User behavior
  - Communication infrastructure tuning
  - Adaptation to failure, prediction of failure
  - Reputation Management