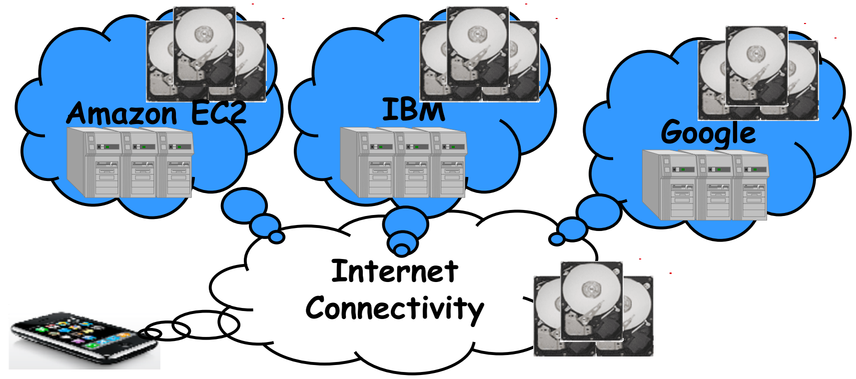
Cloud Storage is the Future

Lessons from the OceanStore Project

John Kubiatowicz
University of California at Berkeley

Where should data reside?

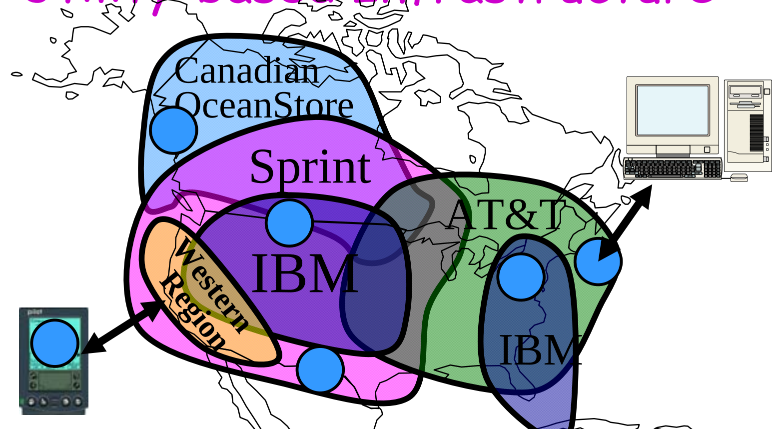


- Consider modern clients:
  - 32GB flash storage
  - 1.5 TB desktop drives
- How is data protected?
  - What if organization fails?
  - What if organization has major IT disaster?
- Today we hear a lot about cloud computing
  - Not much about cloud storage or cross-domain storage
- Storage Cloud ⇒
  - Data is "out there"
  - Migrates to where it is needed

It's all about the data

- Cloud *computing* requires *data*
  - Interesting computational tasks require data
  - Output of computation *is* data
  - Most cloud computing solutions provide *temporary* data storage; little guarantee of reliability
- Reliable services from soft SLAs:
  - Use of multiple clouds for reliable computation
  - Move source data and results between computing clouds and clients
  - Data must move seamlessly between clouds in order to make it viable to use multiple computing clouds for a single task
- Cloud computing is "easy" in comparison
  - Computation is fungable, data is not
  - Data privacy cannot be recovered once compromised
  - Unique data cannot be recovered once lost
  - Integrity of data cannot be restored if written improperly
  - Movement of data requires network resources and introduces latency if done on demand

Original OceanStore Vision: Utility-based Infrastructure



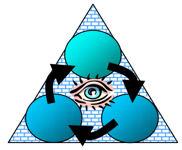
- Data service provided by storage federation
- Cross-administrative domain
- Contractual Quality of Service ("someone to sue")

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 (Cloud Storage?) Provider:
  - Economies of scale
    - Dynamically redistribute resources between clients
    - Focused manpower can serve many clients simultaneously

Key Observation: Want Automatic Maintenance

- Assume that we have a global utility that contains all of the world's data
  - I once looked at plausibility of "mole" of bytes (10^{24})
  - Total number of servers could be in millions?
- System should automatically:
  - Adapt to failure
  - Exclude malicious elements
  - Repair itself
  - Incorporate new elements



- System should be secure and private
  - Encryption, authentication, access control (w/ integrity)
- System should preserve data over the long term (*accessible* for 100s/1000s of years):
  - Geographic distribution of information
  - New servers added/Old servers removed
  - Continuous Repair ⇒ Data survives for long term

Some advantages of 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

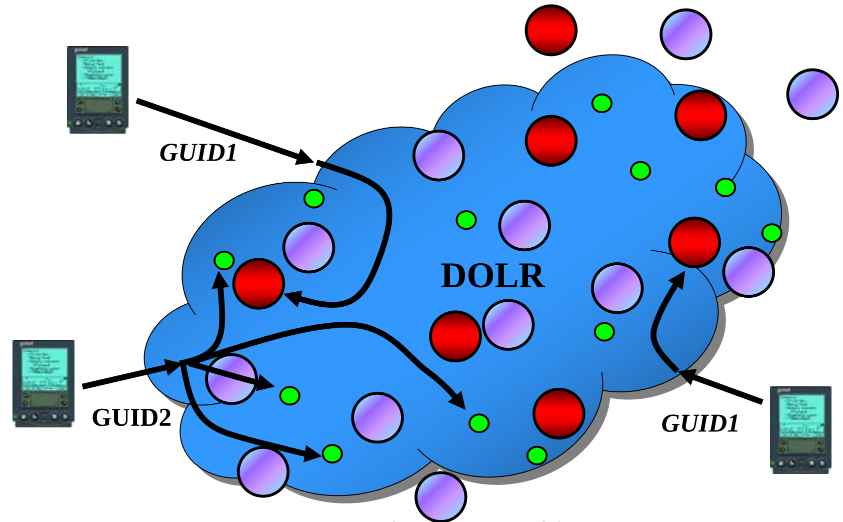
- Untrusted Infrastructure:
  - The OceanStore is comprised of untrusted components
  - Individual hardware has finite lifetimes
  - All data encrypted within the infrastructure
- Mostly Well-Connected:
  - Data producers and consumers are connected to a high-bandwidth network most of the time
  - Exploit 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*

Routing to Data, not endpoints!

Decentralized Object Location and Routing to Self-Verifying Handles (GUIDs)

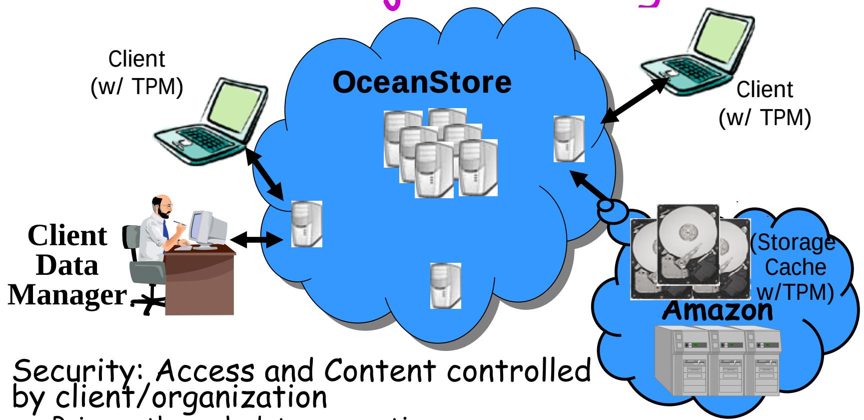


Possibilities for DOLR?

- Original Tapestry
  - Could be used to route to data or endpoints with locality (not routing to IP addresses)
  - Self adapting to changes in underlying system
- Pastry
  - Similarities to Tapestry, now in n^{th} generation release
  - Need to build locality layer for true DOLR
- Bamboo
  - Similar to Pastry – very stable under churn
- Other peer-to-peer options
  - Coral: nice stable system with course-grained locality
  - Chord: very simple system with locality optimizations

Pushing the Vision

Secure Object Storage



- Security: Access and Content controlled by client/organization
  - 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

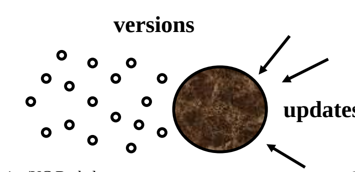
Issues with assuming that data is encrypted

- Key Management!
  - Root keys carried around with users? Biometric unlocking?
- Sharing ⇒ granting keys to clients or servers
  - Need unencrypted versions for computing
  - Does system keep decrypted data around (caching)?
    - Is this risky?
- Do you trust third parties with your data?
  - Perhaps only large organizations?
  - Could impact number/size of computational clouds
- How many keys?
  - One per data field?
  - One per file?
- How to handle/revoke keys?
  - Never give them out (TPM)
  - Reencrypt on revocation (expensive, impractical)

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

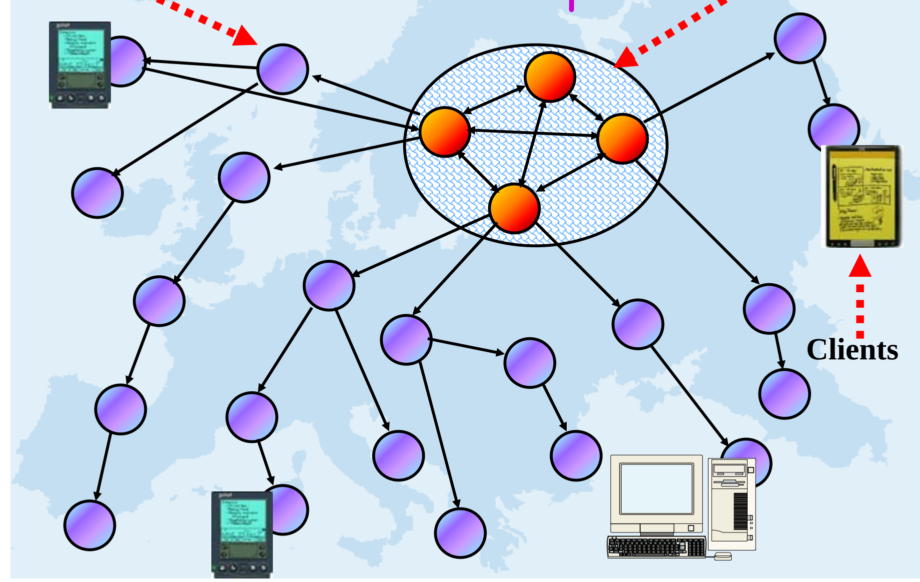
Comet Analogy



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
  - Use of newer codes: n and m flexible
    - Much cheaper to repair data
- 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

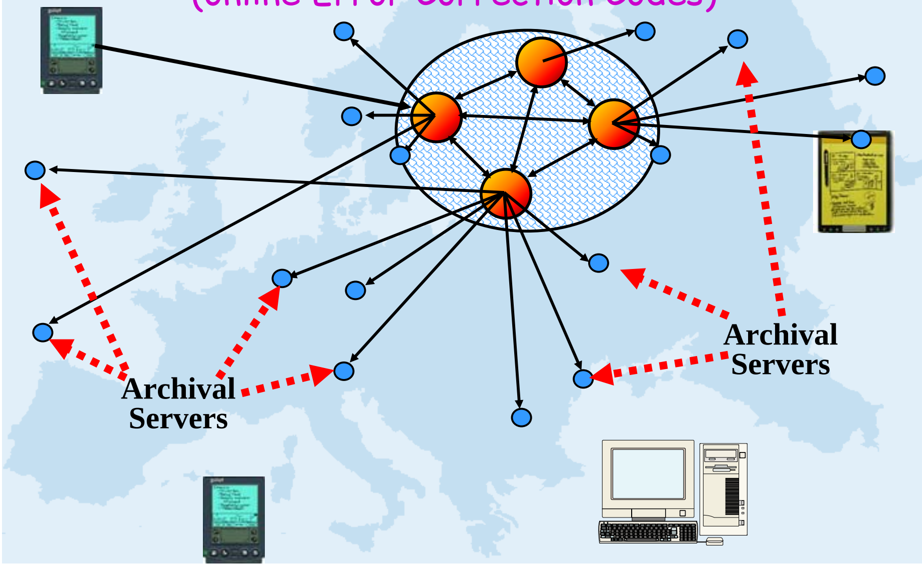
1. Conflict Resolution
2. Versioning/Branching
3. Access control
4. Archival Storage

- 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

Long-Term Archival Storage

Archival Dissemination of Fragments

(online Error Correction Codes)



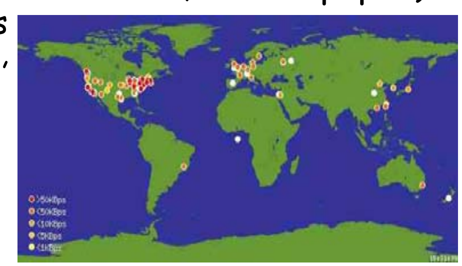
Archival Storage Discussion

- Continuous Repair of Redundancy: Data transferred from physical medium to physical medium
  - No "tapes decaying in basement"
  - Information becomes fully Virtualized
  - Keep the raw bits safe
- Thermodynamic Analogy: Use of Energy (supplied by servers) to Suppress Entropy
  - 1000 year time frame?
- Format Obsolescence
  - Continuous format evolution
  - Saving of virtual interpretation environment
- Proof that storage servers are "doing their job"
  - Can use zero-knowledge proof techniques
  - Reputations
- Data deletion/scrubbing?
  - Harder with this model, but possible in principle

Lessons from Pond Prototype

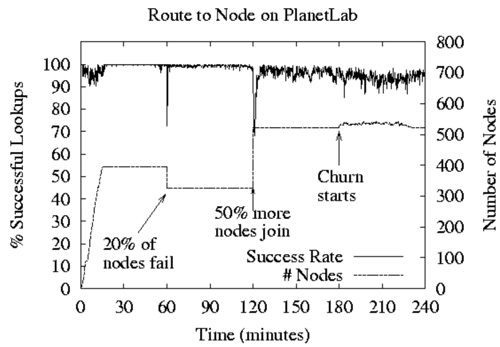
OceanStore Prototype (Pond)

- All major subsystems operational
  - Self-organizing DOLR base (Tapestry)
  - 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



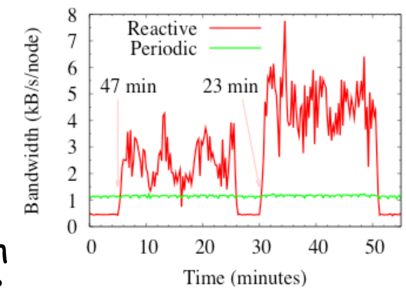
Lesson #1: DOLR is Great Enabler—but only if it is stable

- Tapestry (original DOLR) performed well:
  - 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



Lesson #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

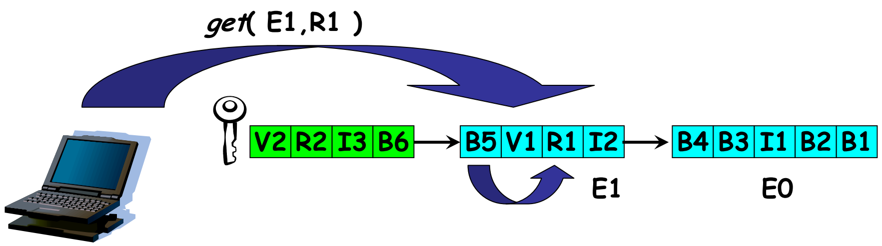
Phase	4 kB write	2 MB write
Validate	0.3	0.4
Serialize	6.1	26.6
Apply	1.5	113.0
Archive	4.5	566.9
Sign Result	77.8	75.8

(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

Lesson #3: Efficiency

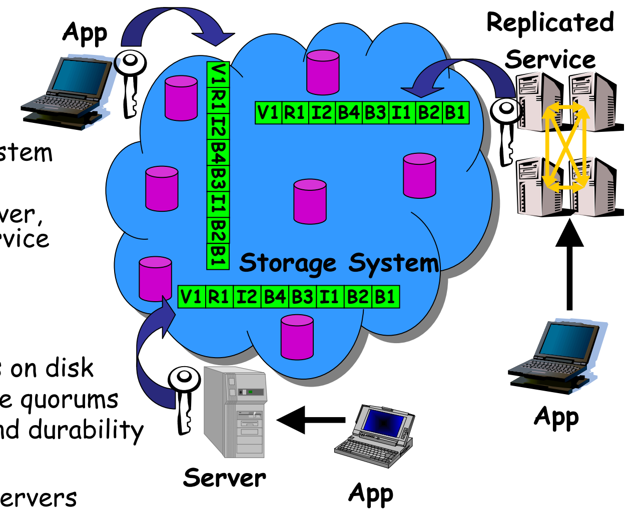
- No resource aggregation
  - Small blocks spread widely
  - Every block of every file on different set of servers
  - Not uniquely OceanStore issue!
- Answer: Two-Level Naming
  - Place data in larger chunks ('extents')
  - Individual access of blocks by name within extents



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

Better Archival Storage: Antiquity Architecture

- 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



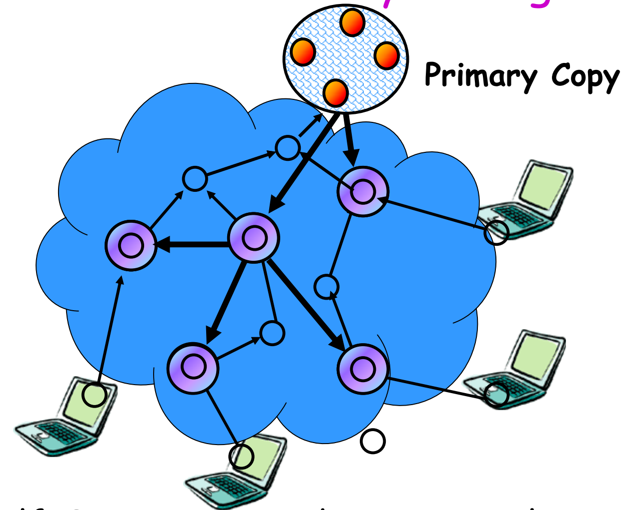
Lesson #4: Complexity

- 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)

QoS Guarantees/SLAs

- What should be guaranteed for Storage?
  - Read or Write Bandwidth to storage?
  - Transactions/unit time?
  - Long-term Reliability?
- *Performance* defined by request/response traffic
  - Ultimately, tied to low-level resources such as network bandwidth
  - High performance data storage requires:
    - On demand migration of data/Caching
    - Staging of updates (if consistency allows)
    - Introspective adaptation: observation-driven migration of data
- *Reliability* metrics harder to guarantee
  - Zero-knowledge techniques to "prove" data being stored
  - Reputation
  - Background reconstruction of data
- Responsible party (trusted third party??):
  - Guarantees of correct commitment of updates
  - Monitoring of level of redundancy

Peer-to-Peer Caching in Pond: Automatic Locality Management

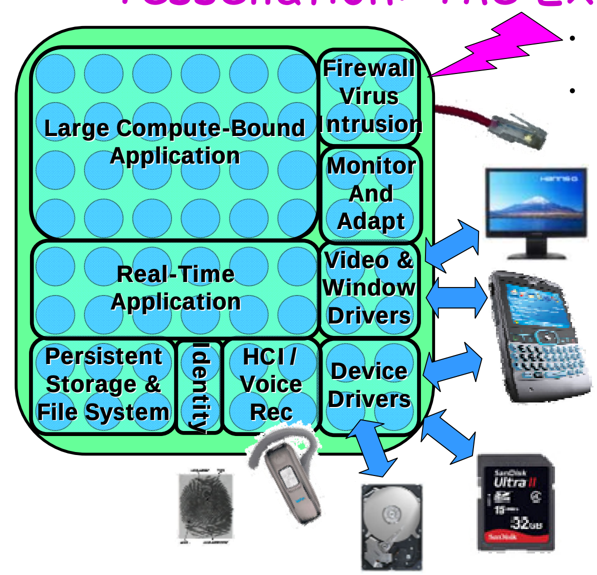


- Self-Organizing mechanisms to place replicas
- Automatic Construction of Update Multicast

What is next?

- Re-architecting of storage on a large scale
  - Functional cloud storage is a "simple matter of programming:"
    - Data location algorithms are a mature field
    - Secure, self-repairing archival storage is buildable
    - Basic Caching mechanisms easy to construct
  - Need to standardize storage interfaces
    - Including security, conflict resolution, managment
  - Missing pieces:
    - Good introspective algorithms to manage locality
    - QoS/SLA enforcement mechanisms
    - Data market?
- Client integration
  - Smooth integration of local storage as cache of cloud
  - Seamless use of Flash storage
    - Quick local commit
    - Caching of important portions of the cloud

Closing Note
Tessellation: The Exploded OS



- Component of Berkeley Parallel Lab (Par Lab)
- Normal Components split into pieces
  - Device drivers
  - Network Services (Performance)
  - Persistent Storage (Performance, Security, Reliability)
  - Monitoring services
  - Identity/Environment services (Security)
- Use of Flash as front-end to cloud storage
  - Transparent, high-speed read/write cache
  - Introspection to keep interested data onboard
  - Commit of data when connectivity/energy optimal

Conclusion

- Cloud Computing ⇒ Cloud Storage
  - More than just interoperability
  - Data preserved in the cloud, by the cloud
- OceanStore project
  - Took a very distributed view of storage
  - Security, performance, long-term archival storage
- Some original papers:
  - "OceanStore: An Architecture for Global-Scale Persistent Storage", ASPLOS 2000
  - "Pond: the OceanStore Prototype," FAST 2003
  - "Tapestry: A Resilient Global-scale Overlay for Service Deployment", JSAC 2004
  - "Handling Churn in a DHT", Usenix 2004
  - "OpenDHT: A Public DHT Service", SIGCOMM 2005
  - "Attested Append-Only Memory: Making Adversaries Stick to their Word", SOSP 2007