Pond: the OceanStore Prototype

Sean Rhea, Patrick Eaton, Dennis Geels,
Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz

2nd USENIX Conference on File and Storage Technologies
2003

Presented By: **Paul Timmins**

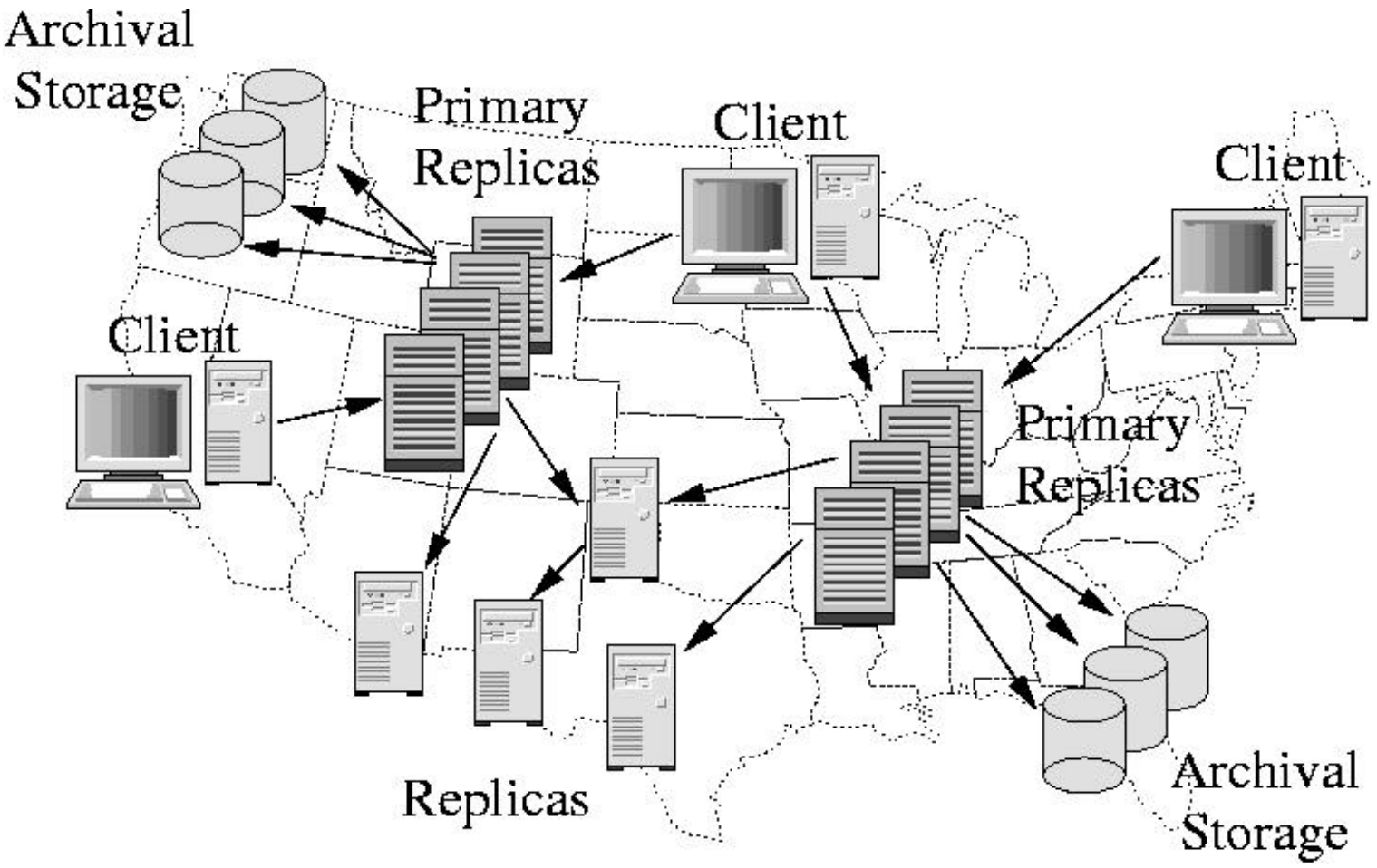
Objectives

- Universally available/accessible storage
  - Access is independent of user's location
  - Share data among hosts "globally" on the Internet
- High Durability
  - Protect against data loss
  - Resilient to node and network failures
- Consistent
  - And, with easily understandable and usable consistency mechanisms
- Integrity
  - What is read is what was written
- Privacy
  - Prevent others from reading your data
- Scalable
  - "Internet-scale"

Assumptions

- **Infrastructure (hosts and network) is untrusted**
  - **Except in aggregate (large % of infrastructure)**
  - **Thus, requiring security and integrity**
- **Infrastructure is constantly changing**
  - **Requiring adaptability and redundancy**
  - **But, without management overhead (self-managing)**

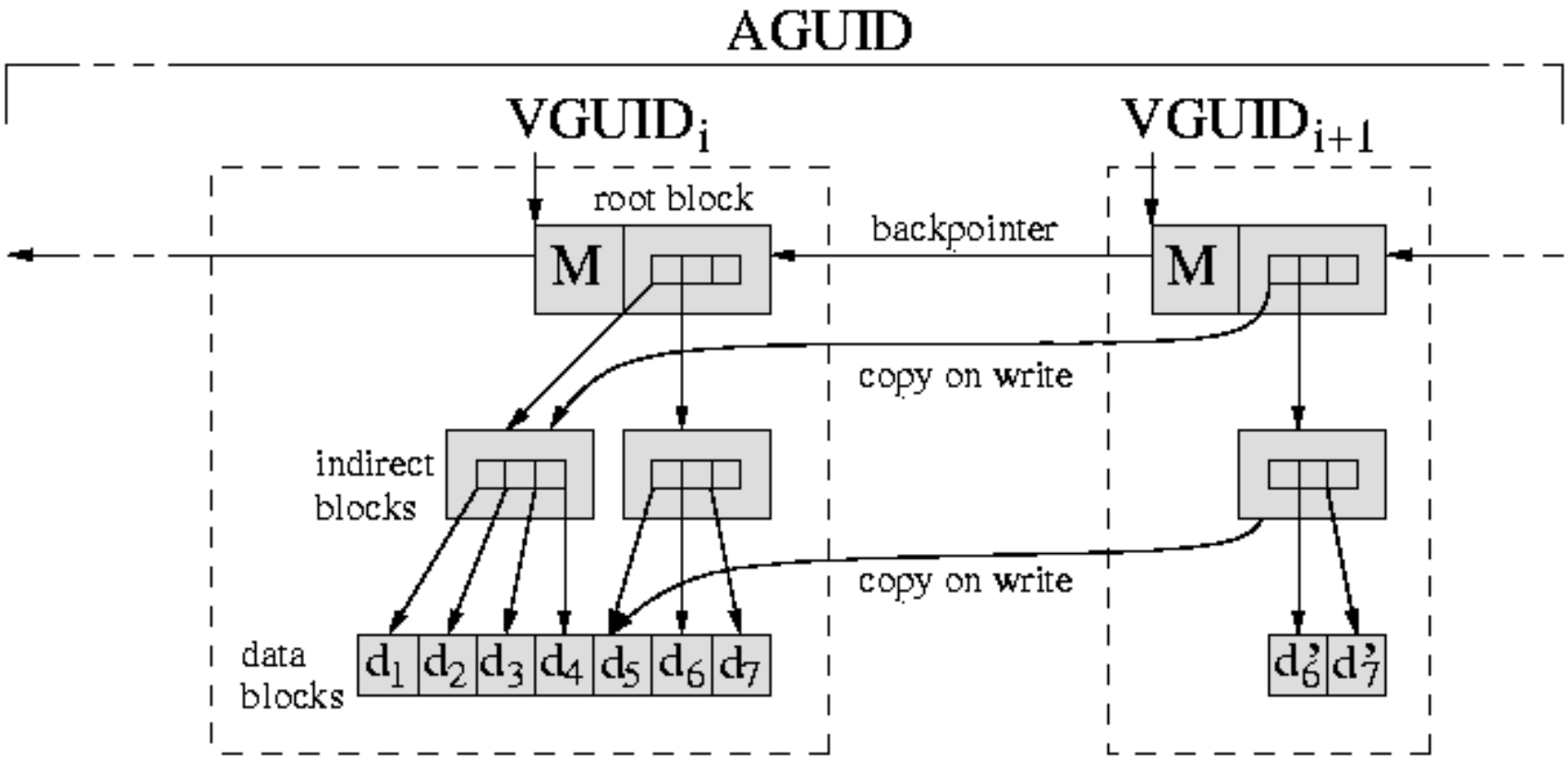
OceanStore System Layout



Storage Organization

- Everything is identified by a GUID (globally unique identifier)
- Data objects (typically a file) are the unit of storage
  - Versioned
  - Latest version is identified by an Active GUID: hash of owner's public key + app specified name
  - Each version is identified by a Version GUID: hash of contents of a version
- Objects are divided into blocks
  - Blocks are identified by a Block GUID, constructed through a hash on the block content.
  - Divided into immutable blocks
  - Blocks are immutable
  - Pond uses 8KB blocks

Data Object Structure



Why Hashes for Identifiers?

- Cryptographically secure hashes have a number of useful properties:
  - Provides statistically insignificant likelihood of collision
    - To have a 50% chance of collision, you need to store about 2^(n/2) objects
    - Pond uses 512 and 1024 bit hashes
  - Reversing hash (learning something about what was stored) is difficult/impossible
  - When used over content, provides integrity, as data can be verified
- However, a number of concerns:
  - Undetectable (or at least difficult to detect) collisions
  - Hash Function Obsolescence

Ref: Henson. "An Analysis of Compare-By-Hash". 9th HotOS, 2003.

Consistency

- Changes are atomic updates
  - Adds blocks, identified by Block GUIDs
  - Then adds new version (Version GUID)
  - Then, updates Active GUID to latest Version GUID
- Primary replica governs updates to GUID, to minimize number of hosts involved in updates
  - Alternative would be to require all hosts to participate, which is inherently unstable
    - Gray et al, “The Dangerous of Replication and a Solution”, SigMod 1996
- Small set of hosts serve as the primary replica
  - Using a Byzantine-fault-tolerant protocol to agree on updates
    - Nodes sign messages using private-keys (between rings) or symmetric-key (node to node in inner-ring)
  - Requires agreement of ~2/3 of servers to make a decision, and is infeasible for large number of servers
  - Chosen by a “responsible party” that chooses stable nodes

Tapestry

- Decentralized object location and routing system
- Routes messages based on a GUID
- Hosts and resources named by GUIDs
- Hosts join tapestry by providing a GUID for itself, then publish the GUIDs of resources
- Hosts can also unpublish or leave tapestry

Erasure Codes

- To protect data, replication is needed…
  - But, resilience against a single failure requires 2x storage (2 copies), resilience against 2 failures requires 3 copies, etc.
- Erasure Codes divide data in *m* identical fragments, which are then encoded into n fragments (n>m).
  - Erasure codes allow the reconstruction of original object from any *m* fragments
  - n/m is the storage cost
  - For example:
    - N=2, m=1, storage cost=2x (mirroring)
    - N=5, m=4, storage cost=1.25x (RAID5)
    - N=32, m=16, storage cost=2x (used in Pond prototype)
  - Uses Cauchy Reed-Solomon coding: oversampling of a polynomial created from the data
  - Cool huh?

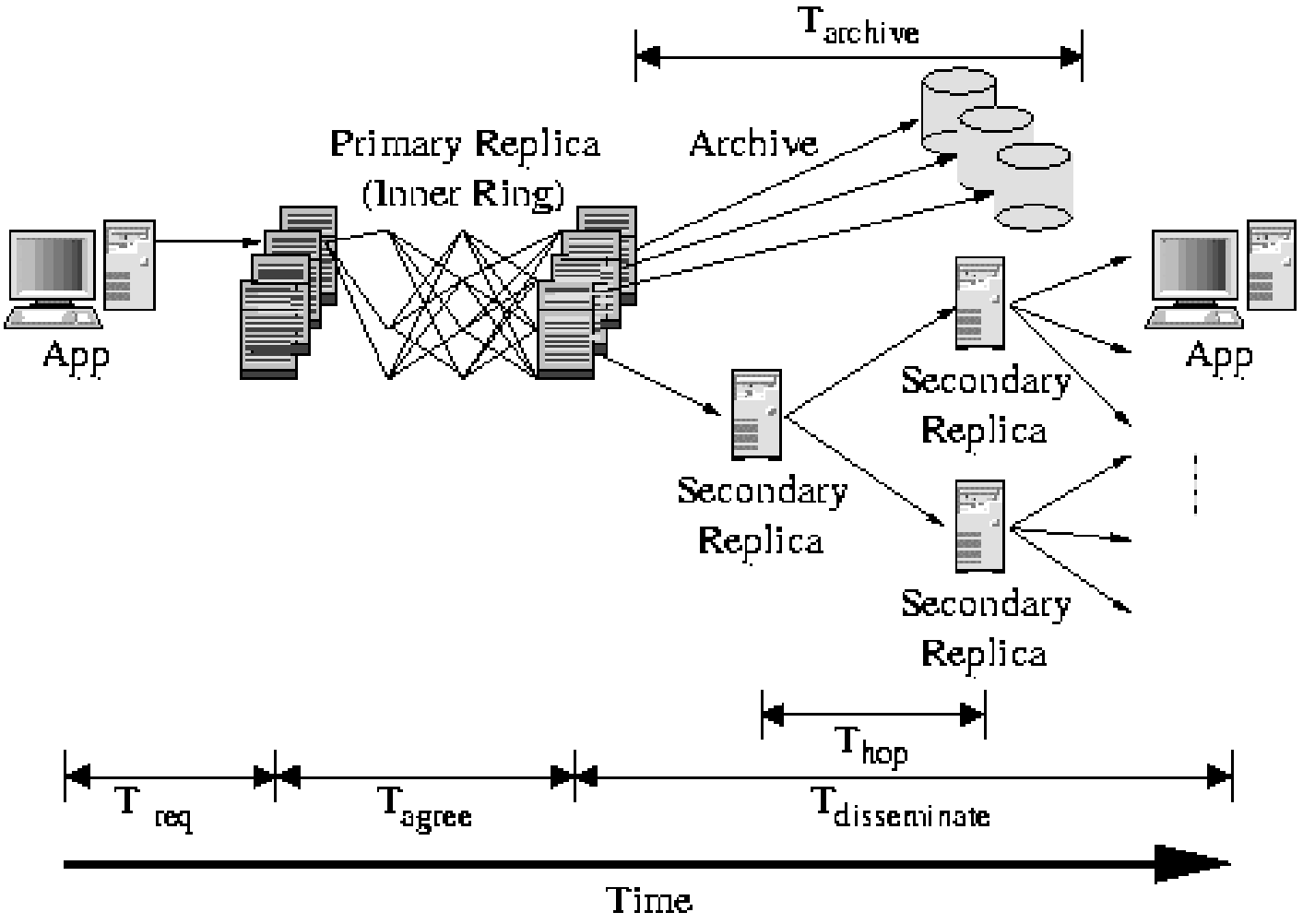
Erasure Codes (2)

- **Used in Pond:**
  - **First, update the primary replica with new blocks**
  - **Erasure code the new blocks**
  - **Distribute the erase-coded blocks**
  - **To reconstruct a block, a host uses tapestry to get fragments (identified by BGUID and fragment number)**

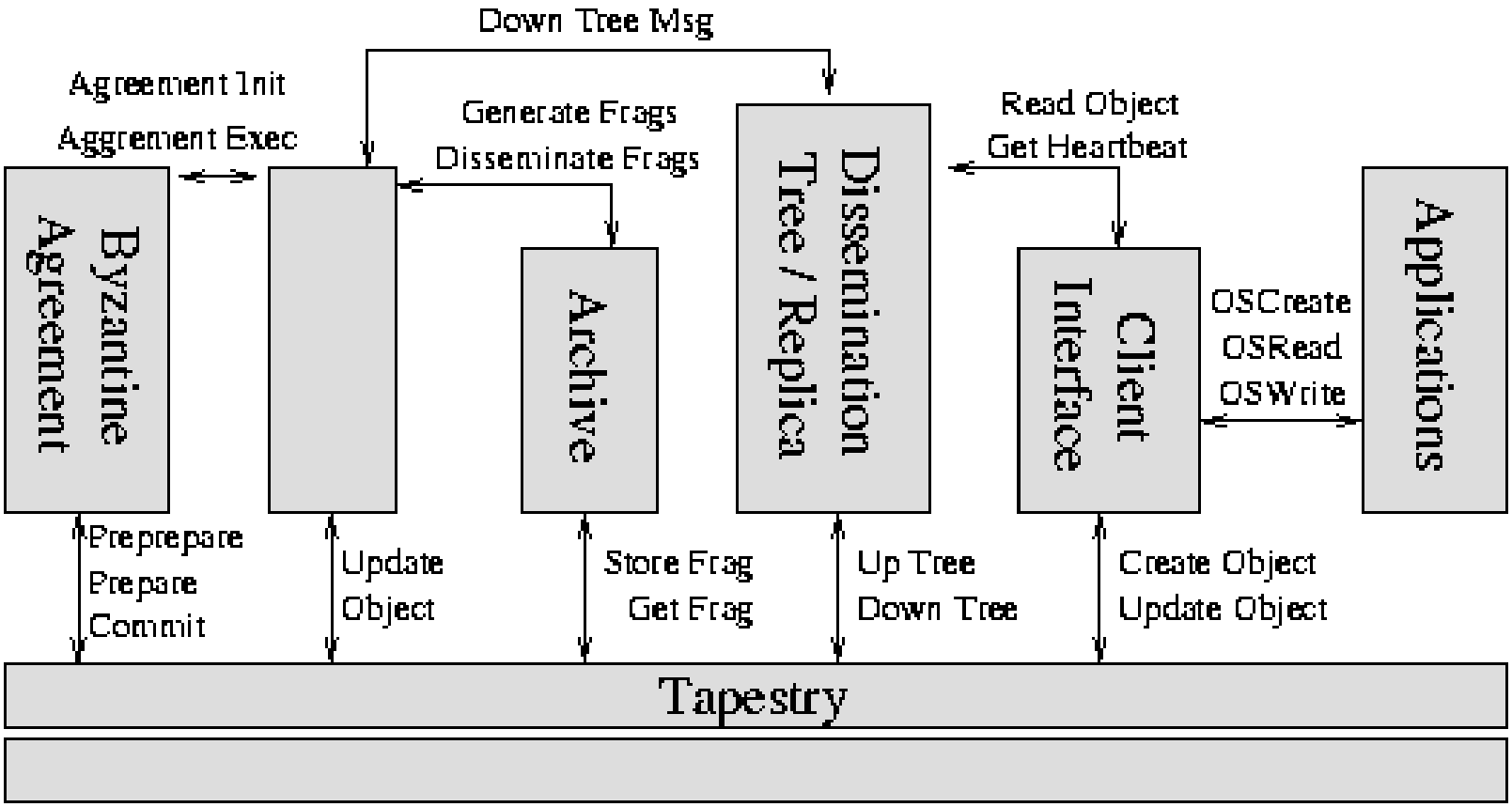
Block Caching

- **Nodes cache blocks, to avoid reconstructing from fragments:**
  - **Nodes request whole block from tapestry**
  - **If not available, then fragments (and caches the block)**
- **LRU cache maintenance**

Update Path


T archive
Primary Replica
(Inner Ring)
Archive
App
Secondary
Replica
App
Secondary
Replica
Secondary
Replica
T hop
T req
T agree
T disseminate
Time

Pond Architecture



Overhead

- 8kb blocks used
  - Meaning, some waste from small blocks
- Metadata:
  - so a 32/8 policy requires 4.8 times storage, not 4 times

Latency Tests

Wide Area

Inner Ring	Client	Avg. Ping	Update Size	Update Latency (ms) 5%	Median	95%
Cluster	Cluster	0.2	4 kB	98	99	100
			2 MB	1098	1150	1448
Cluster	UCSD	27.0	4 kB	125	126	128
			2 MB	2748	2800	9036
Bay Area	UCSD	23.2	4 kB	144	153	166
			2 MB	8763	9626	10231

Local Area

Key Size	Update Size	Archive	Update Latency (ms) 5%	Median	95%
512	4 kB	off	36	37	38
		on	39	40	41
	2 MB	off	494	513	778
		on	1037	1086	1348
1024	4 kB	off	94	95	96
		on	98	99	100
	2 MB	off	557	572	875
		on	1098	1150	1448

Latency Breakdown

Phase	Time (ms) 4 kB Update	2 MB Update
Check Validity	0.3	0.4
Serialize	6.1	26.6
Update	1.5	113.0
Archive	4.5	566.9
Sign Result	77.8	75.8

Andrew Benchmark

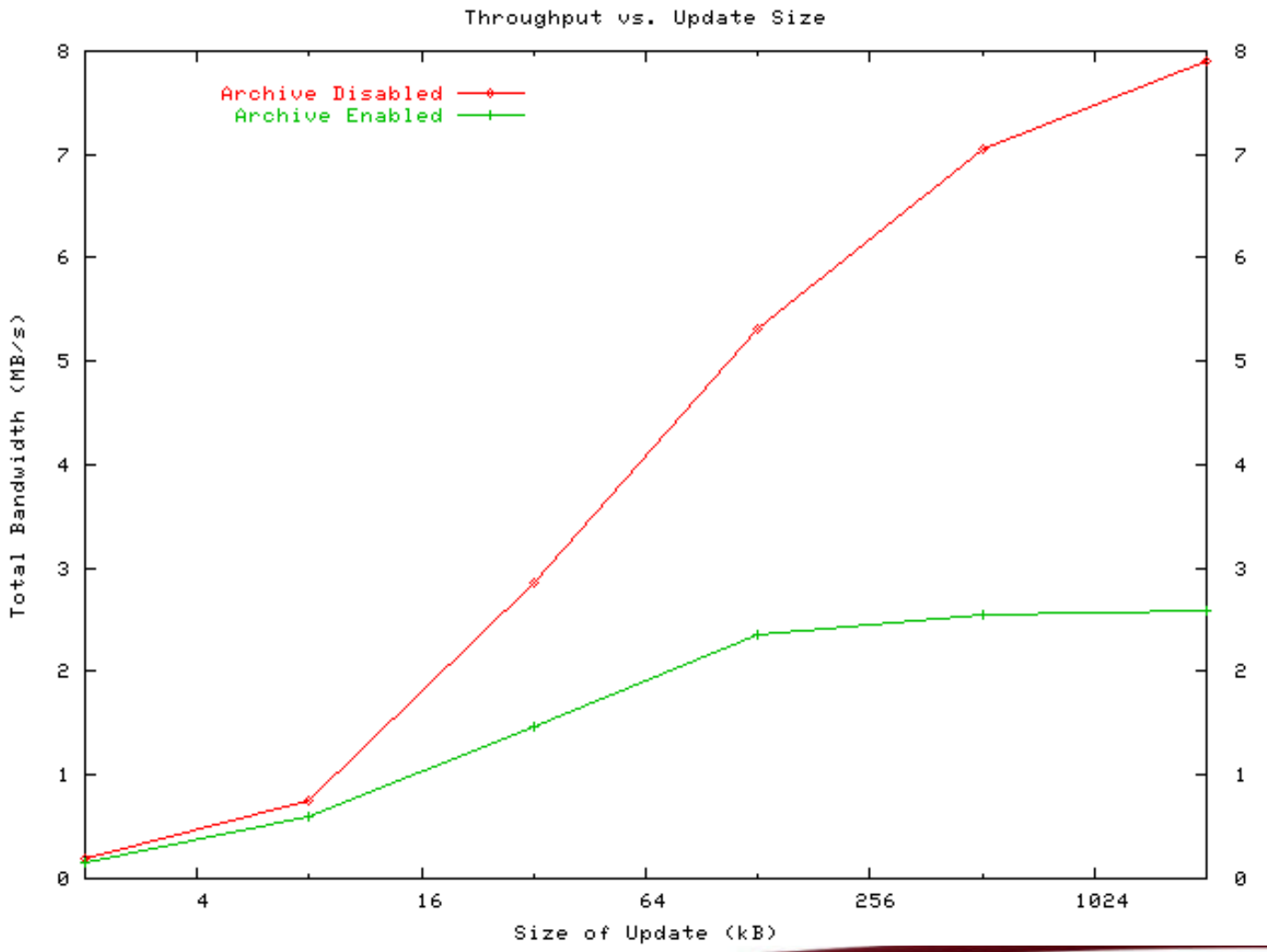
- **Native NFS performance compared to NFS over Pond, with AGUID as NFS file handle**

Results: Andrew Benchmark

Phase	Linux	Pond-512	Pond-1024
I	0.9	2.8	6.6
II	9.4	16.8	40.4
III	8.3	1.8	1.9
IV	6.9	1.5	1.5
V	21.5	32.0	70.0
Total	47.0	54.9	120.3

- **4.6x than NFS in read-intensive phases**
- **7.3x slower in write-intensive phases**

Throughput vs Update Size



Summary of Perf

- Throughput limited by wide area bandwidth
- Latency to read objects depends on latency to retrieve enough fragments
- Erasure coding is expensive

Comments

- Segmentation of the network where no group of inner tier servers can reach 2/3's majority
- Varying network quality/performance between nodes
- Byte shifting (since fixed length blocks)
- Offline/disconnected operation

Conclusions

- **Providing ubiquitous access to information requires addressing:**
  - **Unreliable systems**
  - **Consistency**
  - **Integrity**
  - **Privacy**
- **Pond achieves this through:**
  - **Tapestry: An overlay network that manages resources, a subset of servers managing updates, cryptographically secure hashes for identifiers**
- **Many optimizations exist.**

Questions?

WPI

Ref

- **Some material from: http://oceanstore.cs.berkeley.edu/publications/talks/tahoe-2003-01/geels.ppt**