The OceanStore Write Path

Sean C. Rhea John Kubiatowicz
University of California, Berkeley

June 11, 2002

Introduction: the OceanStore Write Path

Introduction: the OceanStore Write Path

- The Inner Ring
  - Acts as *the* single point of consistency for a file

Introduction: the OceanStore Write Path

- The Inner Ring
  - Acts as *the* single point of consistency for a file
  - Performs write access control, serialization
  - Creates archival fragments of new data and disperses them

Introduction: the OceanStore Write Path

- The Inner Ring
  - Acts as *the* single point of consistency for a file
  - Performs write access control, serialization
  - Creates archival fragments of new data and disperses them
  - *Certifies the results of its actions with cryptography*

Introduction: the OceanStore Write Path

- The Inner Ring
  - Acts as *the* single point of consistency for a file
  - Performs write access control, serialization
  - Creates archival fragments of new data and disperses them
  - *Certifies the results of its actions with cryptography*
- The Second Tier
  - Caches certificates and data produced at the inner ring
  - Self-organizes into an *dissemination tree* to share results

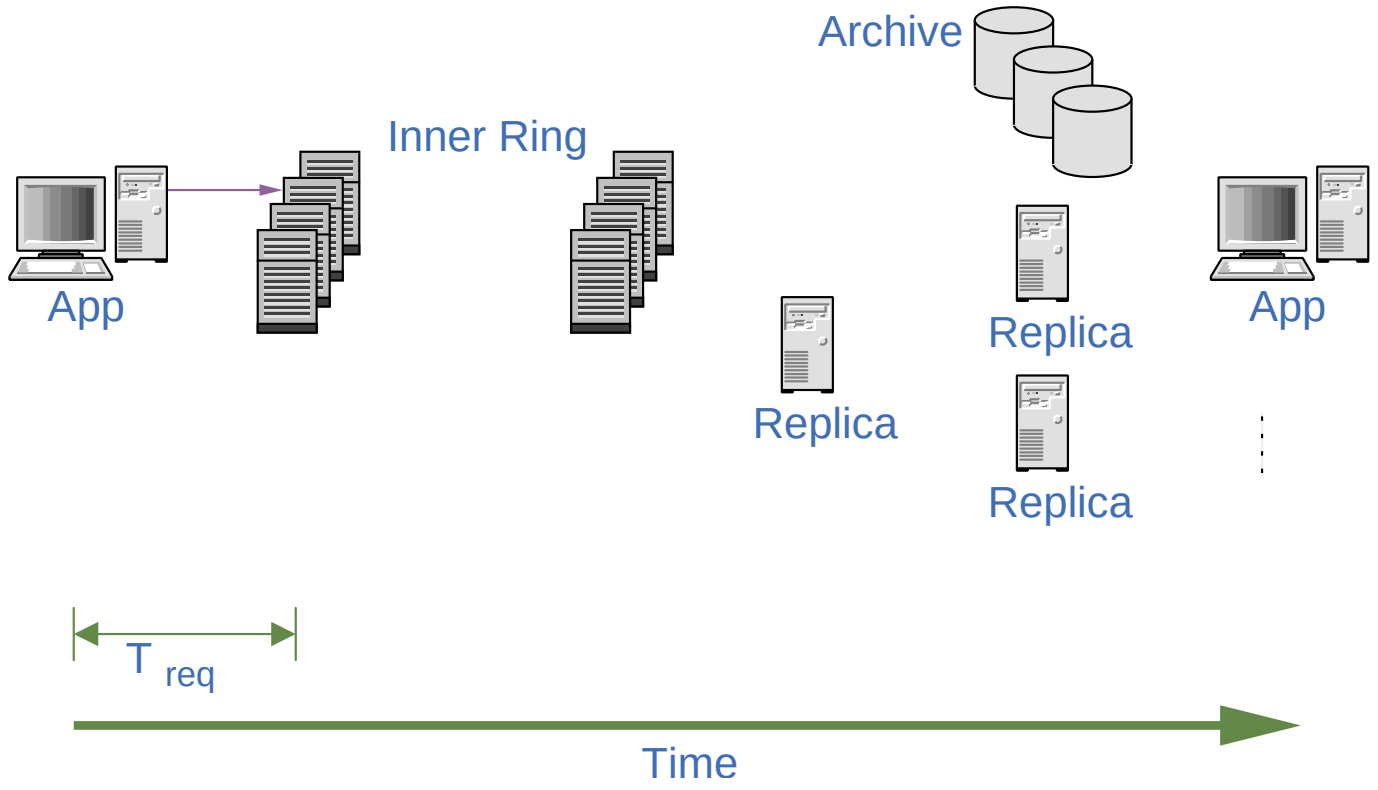
Introduction: the OceanStore Write Path

- The Inner Ring
  - Acts as *the* single point of consistency for a file
  - Performs write access control, serialization
  - Creates archival fragments of new data and disperses them
  - *Certifies the results of its actions with cryptography*
- The Second Tier
  - Caches certificates and data produced at the inner ring
  - Self-organizes into an *dissemination tree* to share results
- The Archival Storage Servers
  - Store archival fragments generated in the Inner Ring

Introduction: the OceanStore Write Path

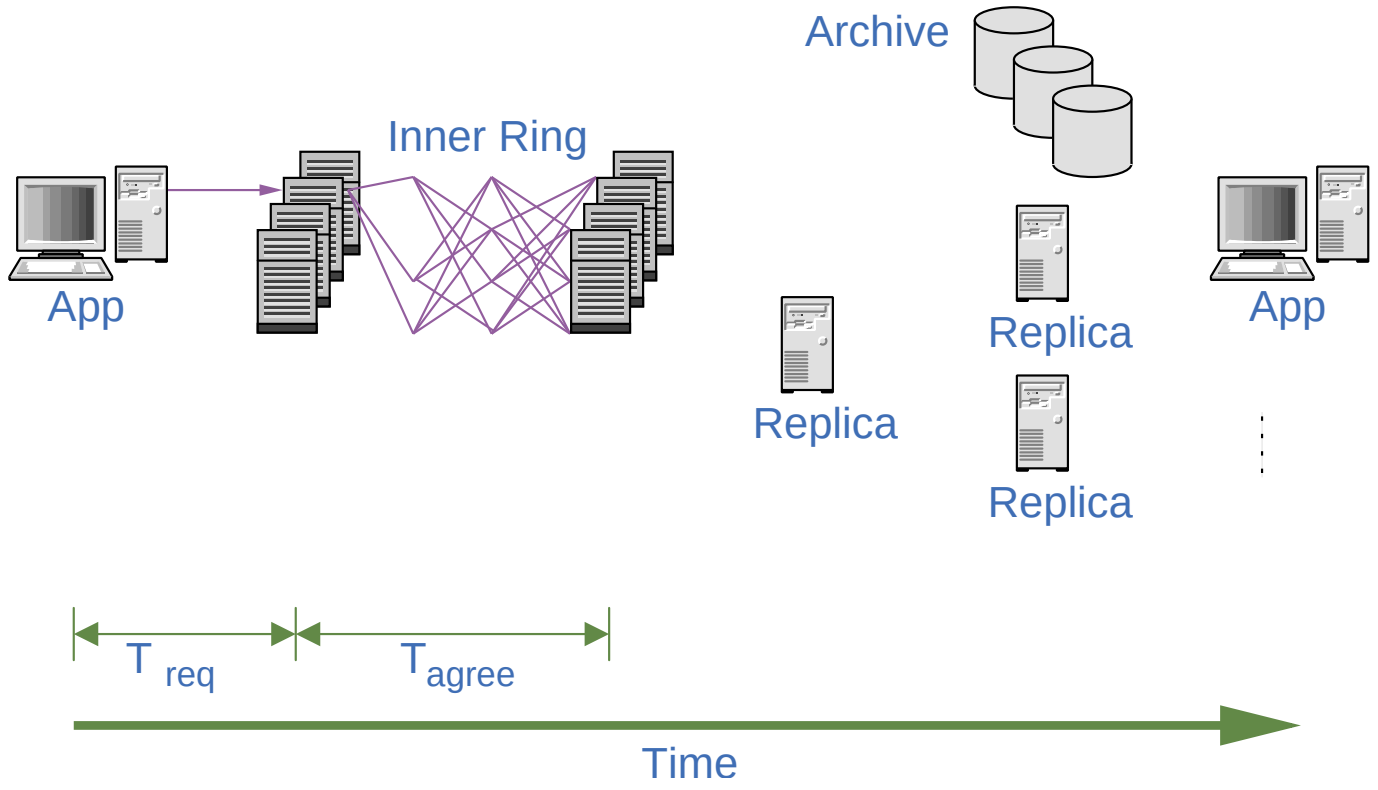
- The Inner Ring
  - Acts as *the* single point of consistency for a file
  - Performs write access control, serialization
  - Creates archival fragments of new data and disperses them
  - *Certifies the results of its actions with cryptography*
- The Second Tier
  - Caches certificates and data produced at the inner ring
  - Self-organizes into an *dissemination tree* to share results
- The Archival Storage Servers
  - Store archival fragments generated in the Inner Ring
- The Client Machines
  - Create updates and send them to the inner ring
  - Wait for responses to come down the dissemination tree

Introduction: the OceanStore Write Path (con't)



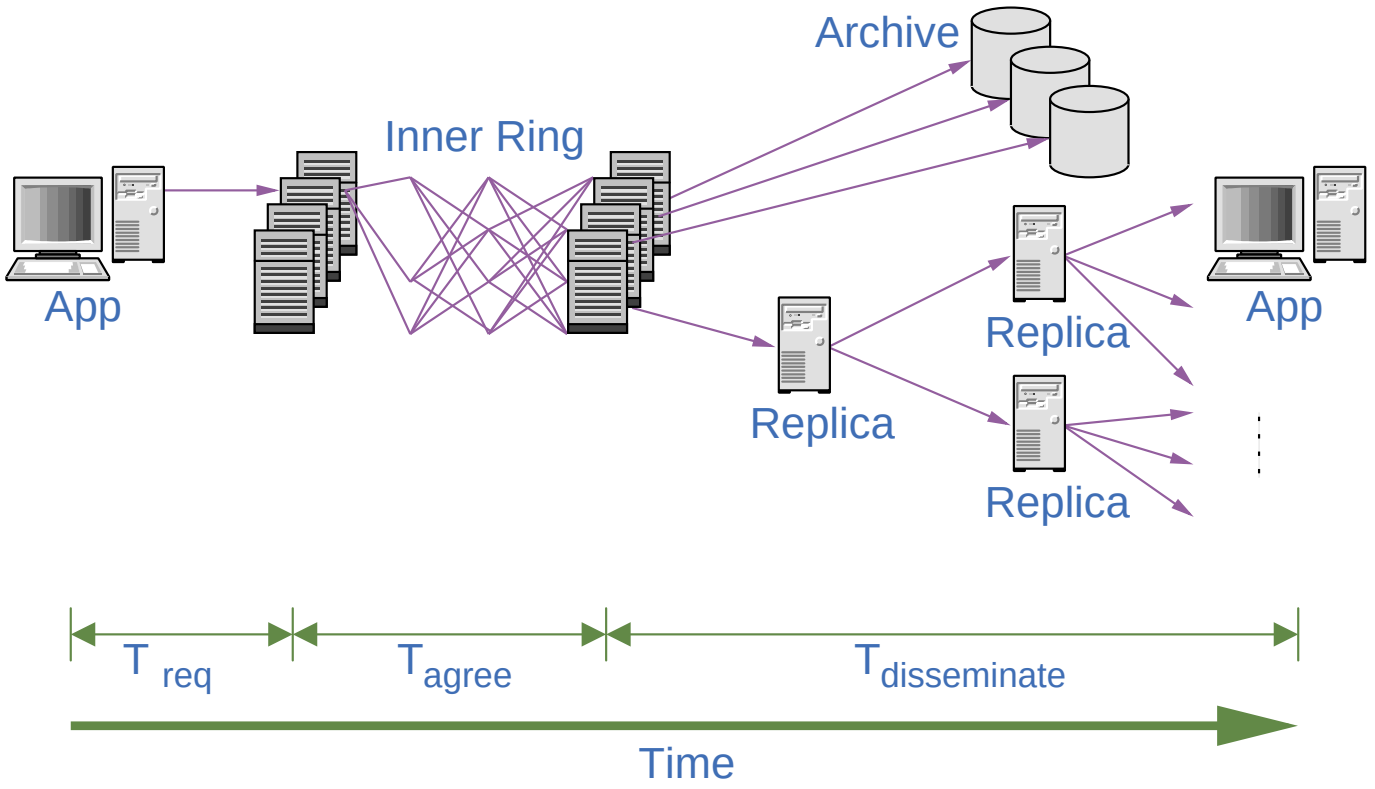
1. *A client sends an update to the inner ring*

Introduction: the OceanStore Write Path (con't)



1. A client sends an update to the inner ring
2. *The inner ring performs a Byzantine agreement, applying the update*

Introduction: the OceanStore Write Path (con't)



1. A client sends an update to the inner ring
2. The inner ring performs a Byzantine agreement, applying the update
3. *The results are sent down the dissemination tree and into the archive*

Write Path Details

- Inner Ring uses Byzantine agreement for fault tolerance
  - Up to f of $3f + 1$ servers can fail
  - We use a modified version of the Castro-Liskov protocol

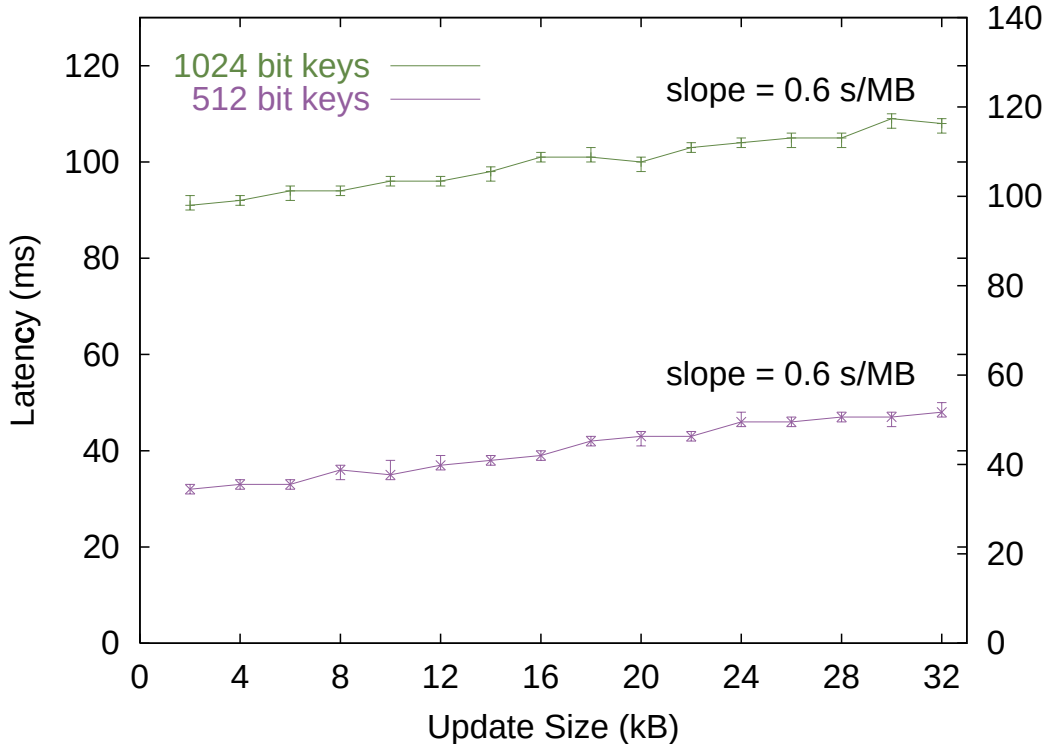
Write Path Details

- Inner Ring uses Byzantine agreement for fault tolerance
  - Up to f of $3f + 1$ servers can fail
  - We use a modified version of the Castro-Liskov protocol
- Inner Ring certifies decisions with proactive threshold signatures
  - Single public (verification) key
  - Each member has a *key share* which lets it generate *signature shares*
  - Need $f + 1$ signature shares to generate full signature
  - Independent sets of key shares can be used to control membership

Write Path Details

- Inner Ring uses Byzantine agreement for fault tolerance
  - Up to f of $3f + 1$ servers can fail
  - We use a modified version of the Castro-Liskov protocol
- Inner Ring certifies decisions with proactive threshold signatures
  - Single public (verification) key
  - Each member has a *key share* which lets it generate *signature shares*
  - Need $f + 1$ signature shares to generate full signature
  - Independent sets of key shares can be used to control membership
- Second Tier and Archive are ignorant of composition of Inner Ring
  - Know only the single public key
  - Allows simple replacement of faulty Inner Ring servers

Micro Benchmarks: Update Latency vs. Update Size



- Use two key sizes to show effects of Moore's Law on latency
  - 512 bit keys are not secure, but are $4\times$ faster
  - Gives an upper bound on latency three years from now

Micro Benchmarks: Update Latency Remarks

- Threshold signatures are expensive
  - Takes 6.3 ms to generate regular 1024 bit signature
  - But takes 73.9 ms to generate 1024 bit threshold signature share
  - (Combining shares takes less than 1 ms)

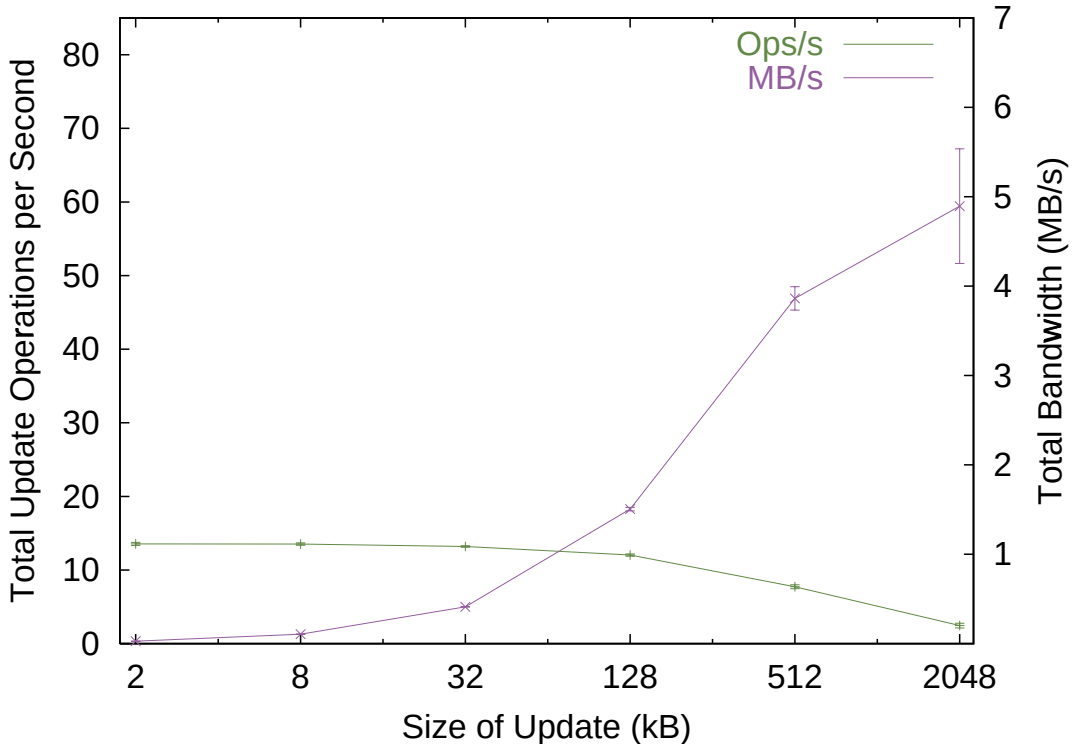
Micro Benchmarks: Update Latency Remarks

- Threshold signatures are expensive
  - Takes 6.3 ms to generate regular 1024 bit signature
  - But takes 73.9 ms to generate 1024 bit threshold signature share
  - (Combining shares takes less than 1 ms)
- Unfortunately, this is a mathematical fact of life
  - Cannot use Chinese Remainder Theorem in computing shares ($4\times$)
  - Making individual shares verifiable is expensive

Micro Benchmarks: Update Latency Remarks

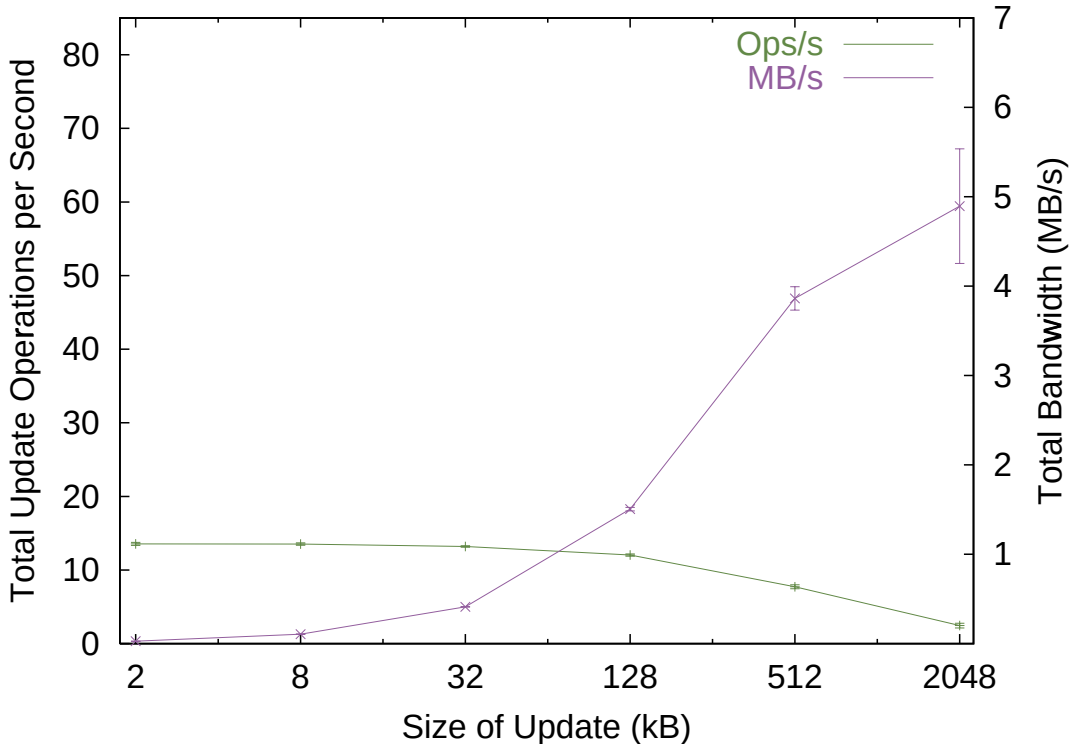
- Threshold signatures are expensive
  - Takes 6.3 ms to generate regular 1024 bit signature
  - But takes 73.9 ms to generate 1024 bit threshold signature share
  - (Combining shares takes less than 1 ms)
- Unfortunately, this is a mathematical fact of life
  - Cannot use Chinese Remainder Theorem in computing shares ($4\times$)
  - Making individual shares verifiable is expensive
- Almost no research into performance of threshold cryptography

Micro Benchmarks: Throughput vs. Update Size



- Using 1024 bit keys, 60 synchronous clients
- Max throughput is a respectable 5 MB/s
  - Berkeley DB through Java can only do about 7.5 MB/s

Micro Benchmarks: Throughput vs. Update Size



- Using 1024 bit keys, 60 synchronous clients
- Max throughput is a respectable 5 MB/s
  - Berkeley DB through Java can only do about 7.5 MB/s
- But we have a problem with small updates
  - 13 ops/s is atrocious!

Batching: A Solution to the Small Update Problem

- What if we could combine many small updates into a single *batch*?

Batching: A Solution to the Small Update Problem

- What if we could combine many small updates into a single *batch*?
- Each Inner Ring member
  - Decides result of each update individually
  - Generates a signature share over the results of *all* of the updates

Batching: A Solution to the Small Update Problem

- What if we could combine many small updates into a single *batch*?
- Each Inner Ring member
  - Decides result of each update individually
  - Generates a signature share over the results of *all* of the updates
- Saves CPU time
  - Generating signature shares is expensive

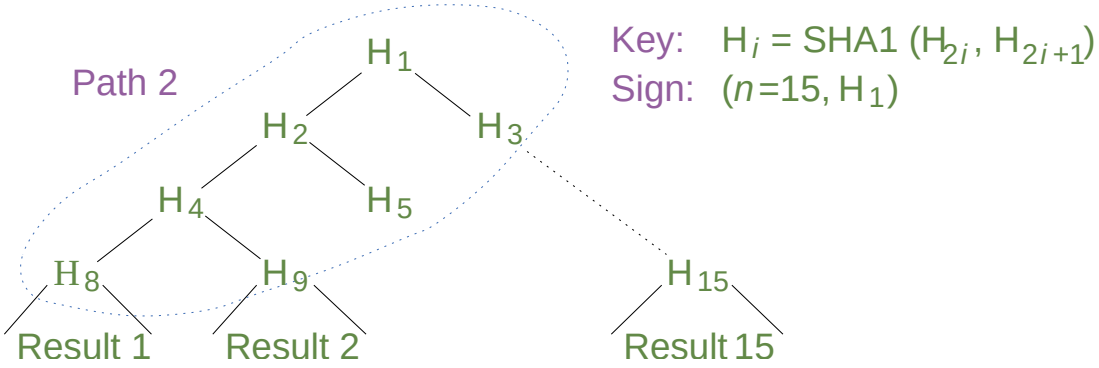
Batching: A Solution to the Small Update Problem

- What if we could combine many small updates into a single *batch*?
- Each Inner Ring member
  - Decides result of each update individually
  - Generates a signature share over the results of *all* of the updates
- Saves CPU time
  - Generating signature shares is expensive
- Saves network bandwidth
  - Each Byzantine agreement requires $O(\text{ringsize}^2)$ messages

Batching: A Solution to the Small Update Problem

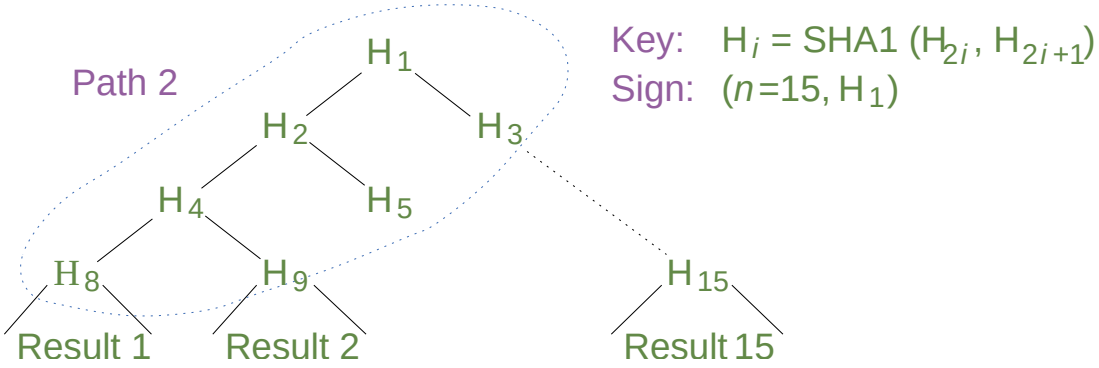
- What if we could combine many small updates into a single *batch*?
- Each Inner Ring member
  - Decides result of each update individually
  - Generates a signature share over the results of *all* of the updates
- Saves CPU time
  - Generating signature shares is expensive
- Saves network bandwidth
  - Each Byzantine agreement requires $O(\text{ringsize}^2)$ messages
- But makes signatures unwieldy
  - Each signature is now $O(\text{batchsize})$ long
  - For high throughput, we want batch sizes in the hundreds or thousands

Merkle Trees: Making Batching Efficient



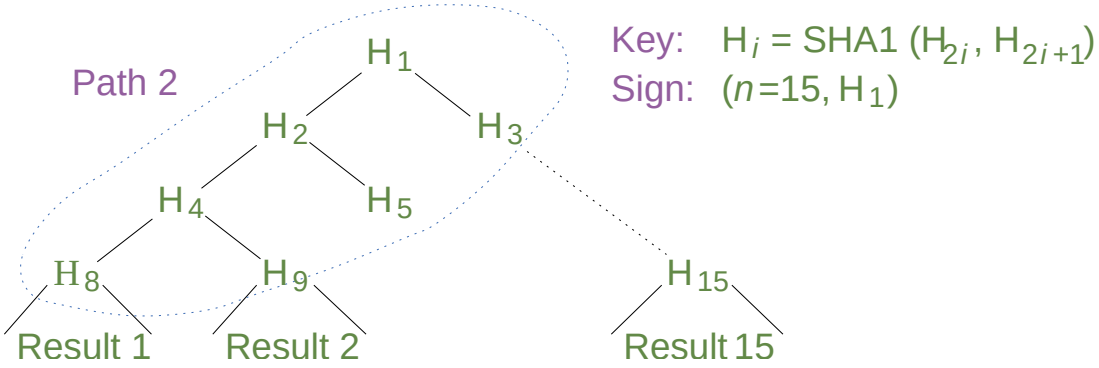
- Build a Merkle Tree over results
  - Each node is a hash of it's two children

Merkle Trees: Making Batching Efficient



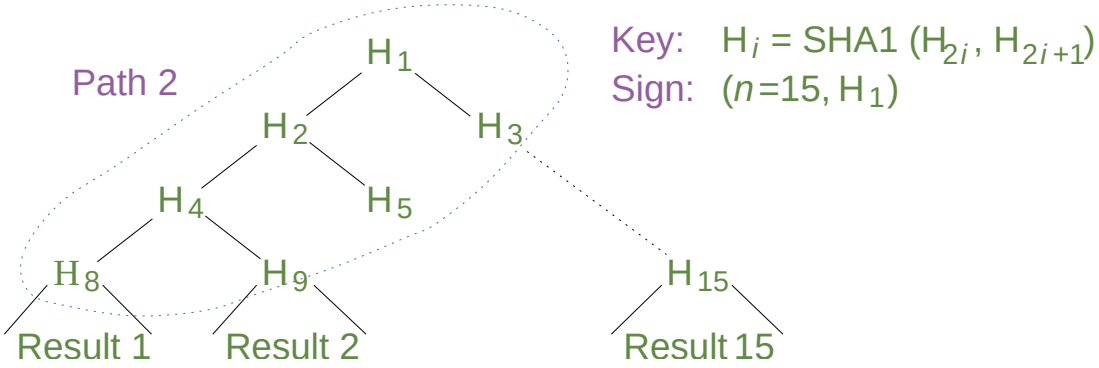
- Build a Merkle Tree over results
  - Each node is a hash of it's two children
- Sign only the tree size and the top hash
  - To verify *Result 2*, need only signature plus

Merkle Trees: Making Batching Efficient



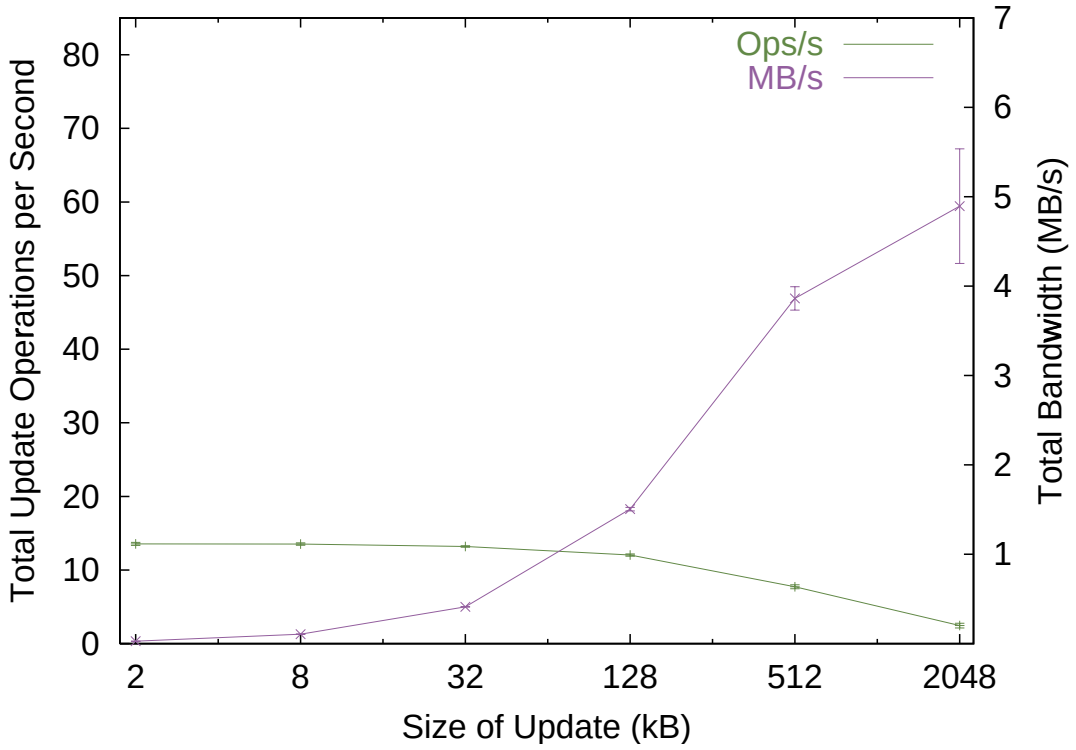
- Build a Merkle Tree over results
  - Each node is a hash of it's two children
- Sign only the tree size and the top hash
  - To verify *Result 2*, need only signature plus
  - Signature over any one result is only $O(\log \text{batchsize})$

Merkle Trees: Making Batching Efficient



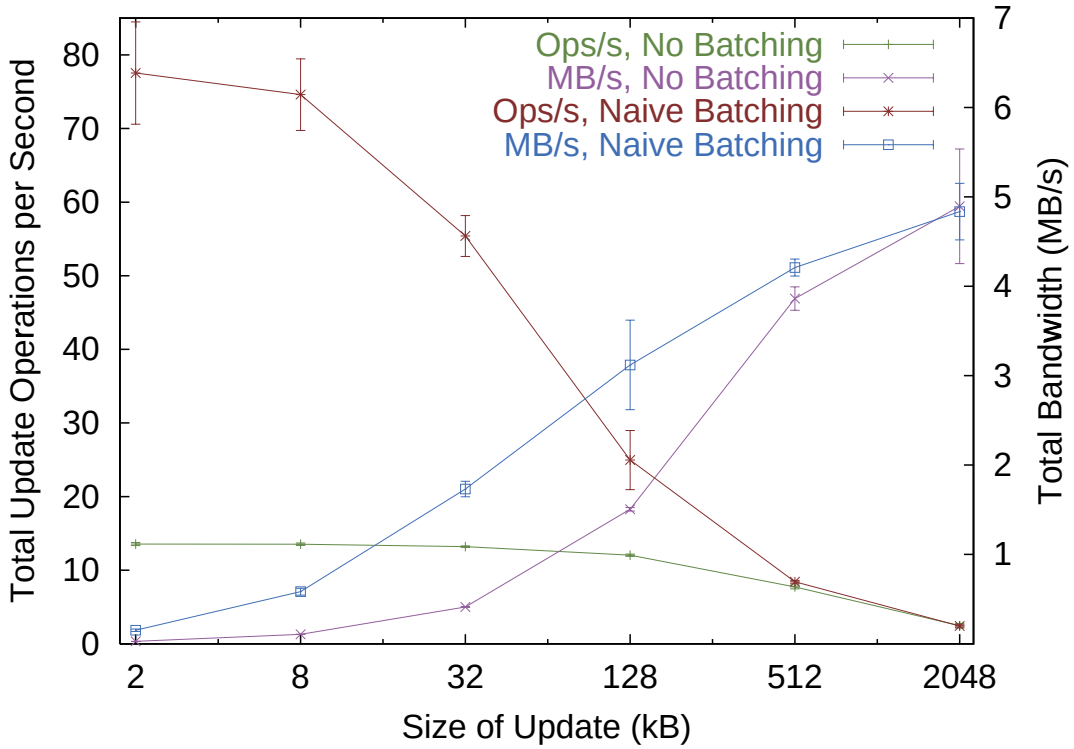
- Build a Merkle Tree over results
  - Each node is a hash of it's two children
- Sign only the tree size and the top hash
  - To verify *Result 2*, need only signature plus
  - Signature over any one result is only $O(\log \text{batchsize})$
  - Provably secure

Micro Benchmarks: Throughput vs. Update Size



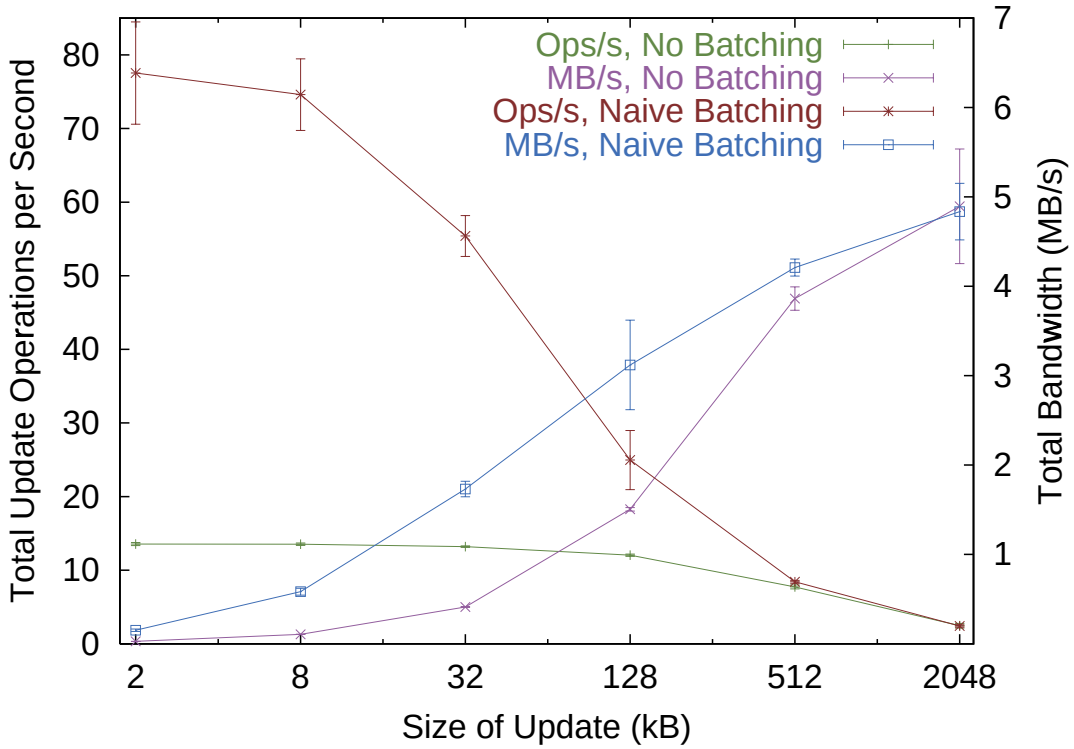
- Using 1024 bit keys
- Max throughput is a respectable 5 MB/s
  - Berkeley DB through Java can only do about 7.5 MB/s
- But we have a problem with small updates
  - 13 ops/s is atrocious!

Micro Benchmarks: Throughput vs. Update Size (w/ Batching)



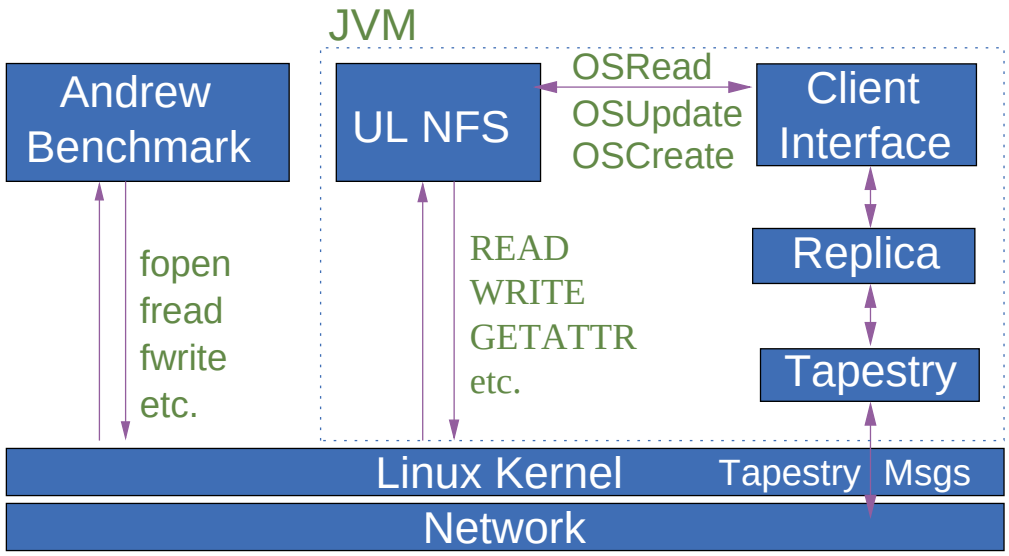
- Batching works great
  - Amortizes expensive agreements over many updates
  - For small updates, go from 13.5 ops/s to 76 ops/s

Micro Benchmarks: Throughput vs. Update Size (w/ Batching)



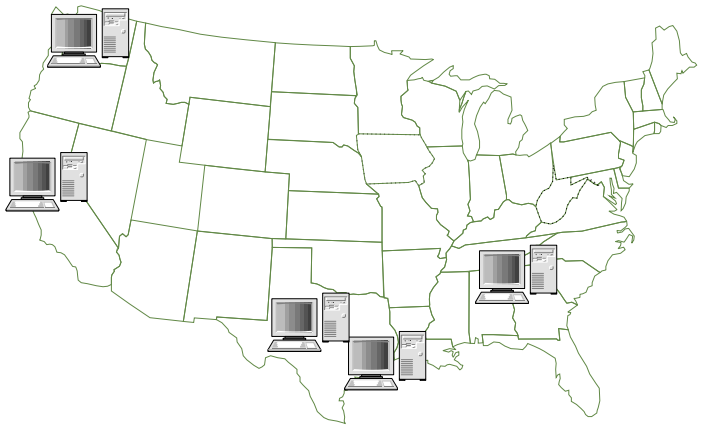
- Batching works great
  - Amortizes expensive agreements over many updates
  - For small updates, go from 13.5 ops/s to 76 ops/s
  - Introspecting on batch size should further improve small update tput

Macro Benchmarks: The Andrew Benchmark



- Built a UNIX file system on top of OceanStore
  - Runs as a user-level NFS daemon on Linux
  - Application's use familiar `fopen`, `fwrite`, etc. *No recompilation.*
  - Kernel translates to NFS requests and sends to local daemon
  - Daemon translates to OceanStore requests and sends out on network

Macro Benchmarks: The Andrew Benchmark

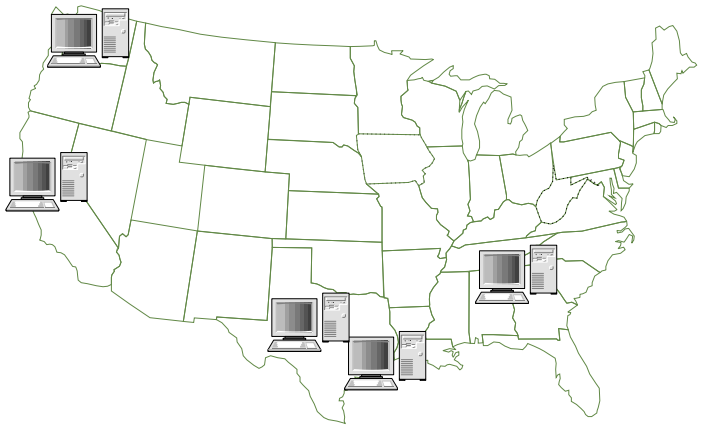


	Destination			
Source	U. TX	GA Tech	Rice	UW
UCB	45.3 (0.75)	56.5 (0.14)	49.6 (3.1)	20.0 (0.11)
UTA	–	24.1 (0.49)	8.45 (1.5)	61.7 (0.22)
GA Tech	–	–	27.7 (2.2)	59.0 (0.20)
Rice	–	–	–	61.5 (0.69)

Inter-host ping times in milliseconds

- For more realism, we used a nationwide network
  - Find out whether Byzantine agreement is practical in wide area
- Ran the Andrew Benchmark
  - Simulates software development workload

Macro Benchmarks: The Andrew Benchmark

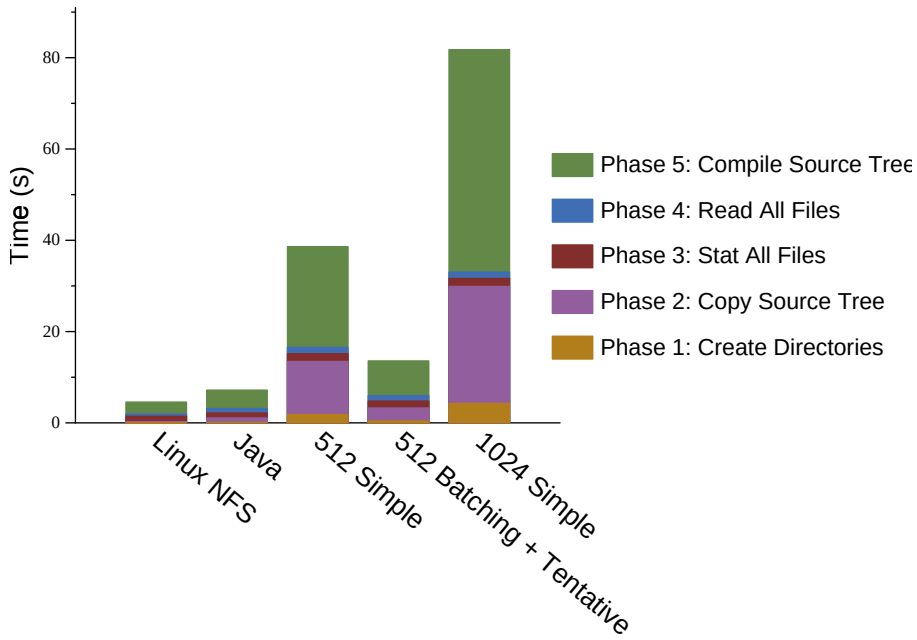


	Destination			
Source	U. TX	GA Tech	Rice	UW
UCB	45.3 (0.75)	56.5 (0.14)	49.6 (3.1)	20.0 (0.11)
UTA	–	24.1 (0.49)	8.45 (1.5)	61.7 (0.22)
GA Tech	–	–	27.7 (2.2)	59.0 (0.20)
Rice	–	–	–	61.5 (0.69)

Inter-host ping times in milliseconds

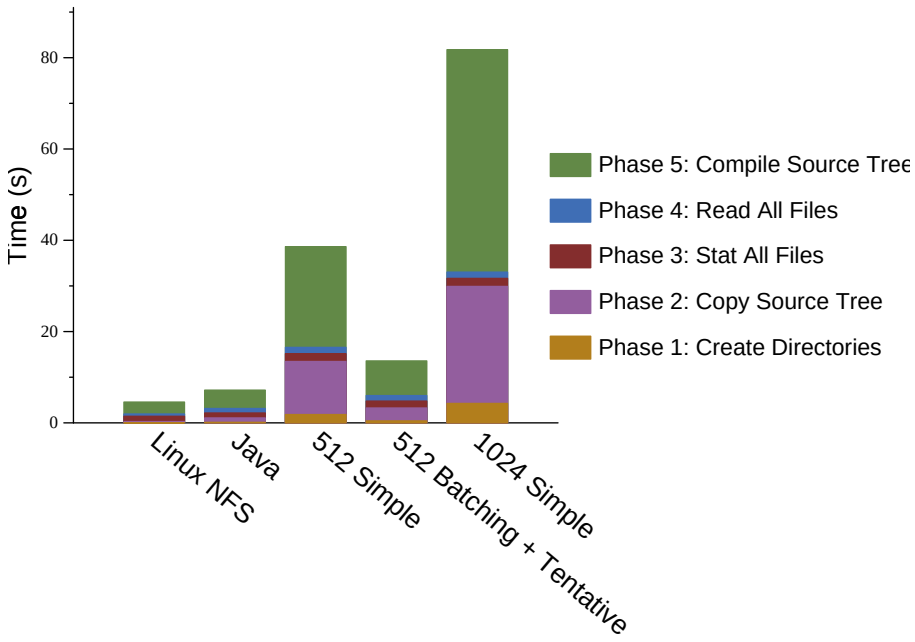
- For more realism, we used a nationwide network
  - Find out whether Byzantine agreement is practical in wide area
- Ran the Andrew Benchmark
  - Simulates software development workload
- For control, used several competitors
  - Linux user-level NFS daemon: real NFS, ships with Debian GNU/Linux
  - Java-based user-level NFS daemon: uses disk (not OceanStore)

Macro Benchmarks: Local Andrew



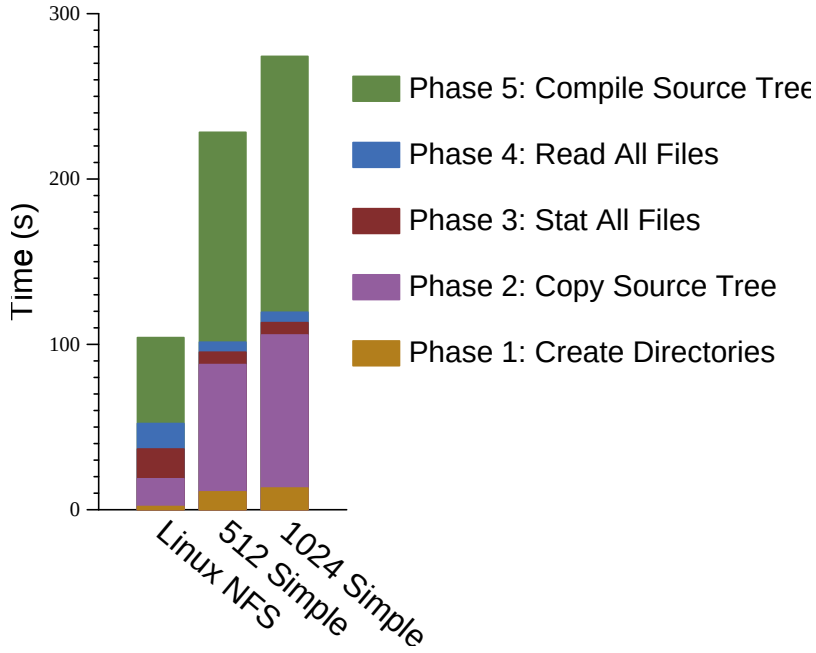
- Simple OceanStore performance not so hot
  - In the local area, *NFS is in its element*; OceanStore isn't

Macro Benchmarks: Local Andrew



- Simple OceanStore performance not so hot
  - In the local area, *NFS is in its element*; OceanStore isn't
- But with tentative update support and batching, OceanStore pretty good
  - Tentative updates let client go on while waiting for agreements
  - Batching allows inner ring to keep up
  - *Within a factor of two of Java-based NFS*

Macro Benchmarks: Nationwide Andrew



- In the wide area, *OceanStore is its element*; NFS isn't
  - Even simple OceanStore is nearly within a factor of two
  - Numbers with batching and tentative updates forthcoming
  - Should outperform NFS

Conclusion

- All the basics of the OceanStore write path implemented and working
  - Not doing full recovery yet

Conclusion

- All the basics of the OceanStore write path implemented and working
  - Not doing full recovery yet
- Performance is good
  - Single update time under 100 ms, improves directly with Moore’s Law

Conclusion

- All the basics of the OceanStore write path implemented and working
  - Not doing full recovery yet
- Performance is good
  - Single update time under 100 ms, improves directly with Moore’s Law
  - Throughput great for large updates

Conclusion

- All the basics of the OceanStore write path implemented and working
  - Not doing full recovery yet
- Performance is good
  - Single update time under 100 ms, improves directly with Moore's Law
  - Throughput great for large updates
  - Batching allows inner ring to amortize signatures over many updates
    - *Get large update throughput with small updates*
    - Secure and space-efficient

Conclusion

- All the basics of the OceanStore write path implemented and working
  - Not doing full recovery yet
- Performance is good
  - Single update time under 100 ms, improves directly with Moore's Law
  - Throughput great for large updates
  - Batching allows inner ring to amortize signatures over many updates
    - *Get large update throughput with small updates*
    - Secure and space-efficient
- Provides a lot more functionality than competition
  - Higher durability and availability than NFS
  - Cryptographic data integrity
  - Versioning allows logical "undo"