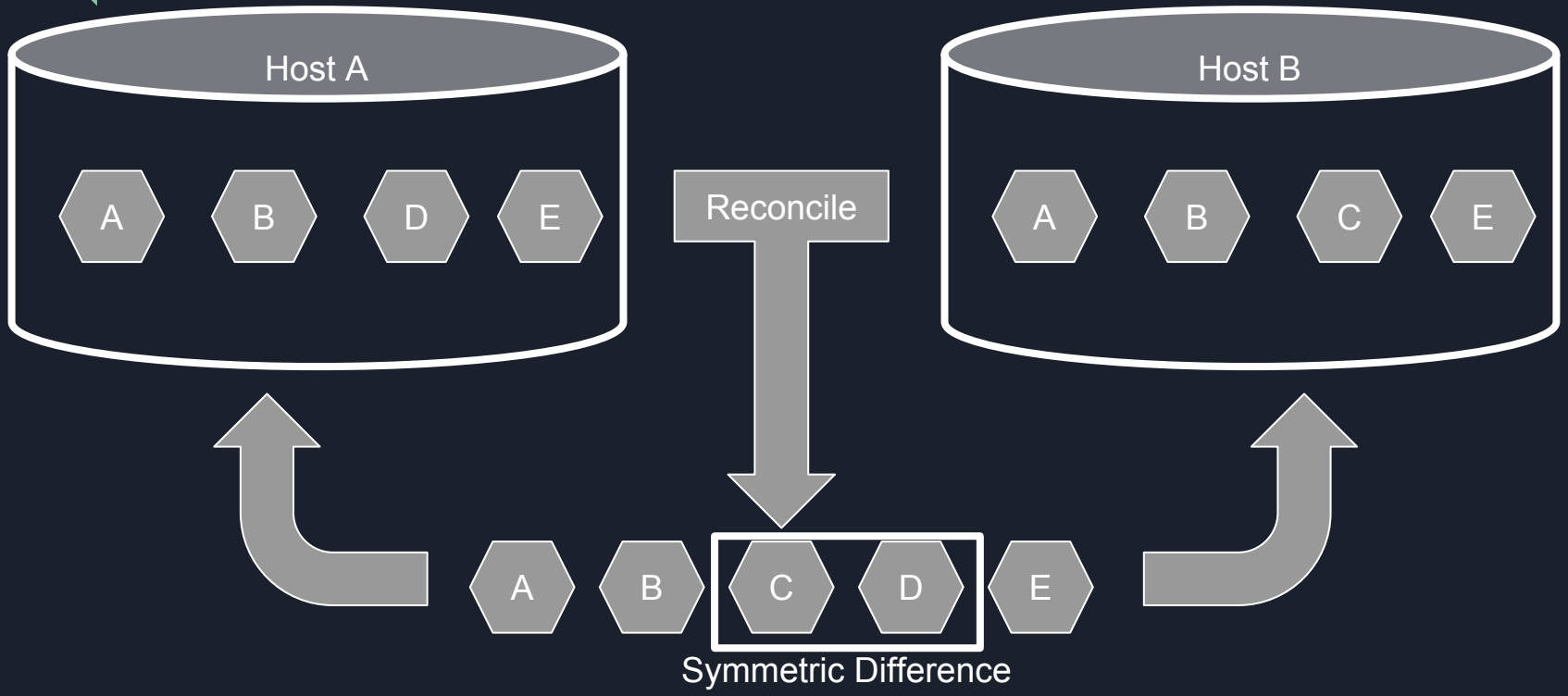
Robust Set Reconciliation[1]

Present by Bowen Song

[1] D. Chen, C. Konrad, K. Yi, W. Yu, and Q. Zhang, “Robust set reconciliation,” in Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. ACM, 2014, pp. 135–146.

What is set reconciliation



What is Set Reconciliation used for?

Distributed file system (GFS, Cloud Sync)

Database Synchronization

Example distributed system settings: [2]

- Peer-to-peer: worker nodes A and B wish to receive missing files from each other
- Partition Healing: Joining partitioned hosts
- Deduplication: Continuous backup or backup with prior context
- Synchronizing parallel activations: Multiple actors performs similar functions*
- Opportunistic ad hoc networks: low budget connectivity to other peers

Presentation Outline

Problem Definition

Motivation

Algorithm Contributions

Algorithm Workflow

Analysis and Discussion

Experimental Performance

Conclusion

Problem Definition

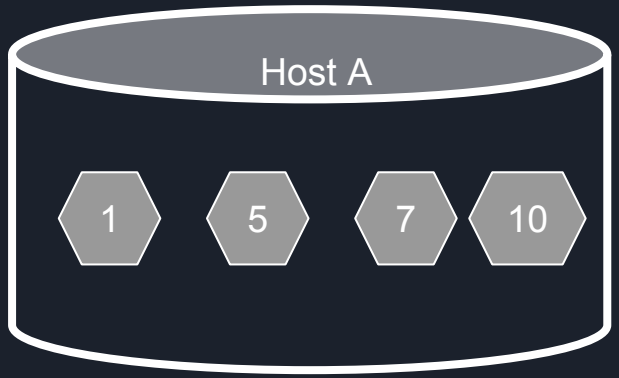
Two or more hosts, each holding a set of similar elements, wish to exchange their differences to establish data consistency

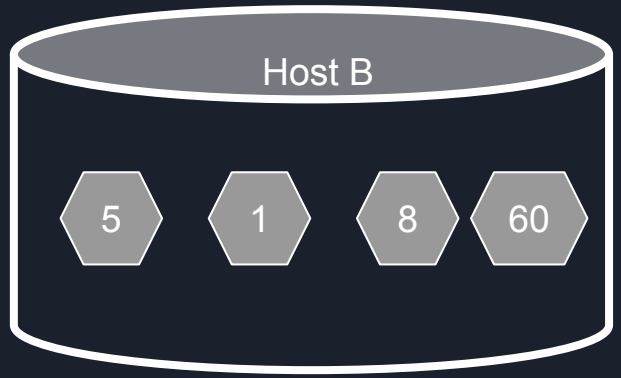
Limited communication and computation resources: Bandwidth, message size, persistency, process power, storage

Minimize communication and computation complexity, memory overhead, and number of communication rounds.

Allowing tolerable differences and reconciling truly different elements*

Tolerate Set Differences





Reconcile true difference

Motivation for allowing differences

Differences coming from:

- Noisy transmission channel and lossy data compression
- Different but mathematically equivalent data (rounding errors)
- Privacy-preserving data (intentionally adding controlled noise)

I.e. Google's Image Backup:

High quality: Great visual quality at reduced file size

Original quality: Full resolution

Challenges

Get the best reconciliation quality:

- knowing budget
- without knowing number of differences

Efficient in communication and computation: minimizing message size, Memory overhead, processing time

Single round communication to allow one-way broadcasting reconciliation

Measuring differences between sets of (multi-dimensional) data for performance evaluation

Algorithm Contribution

Control communication budget trade-off with reconciliation quality

Low communication and and computation costs

Algorithm fit for both One-way and two-way

A hierarchy structure increasing reconciliation resolution

Use of IBLT to compress messages

An Evaluation of minimum difference matching between Alice and Bob using Earth Mover’s Distance (EMD)

Invertible Bloom Lookup Table (IBLT)

Bloom filter data structure designed for key-value pairs

Supporting Operations: **Insert, Delete, List entries**, Subtract[2], Get

Size of Table determine number of key-value pairs to perform entry listing*

Purpose:

- Message Compression
- Extract symmetric differences between two sets

IBLT

- Table of fixed number of cells with key-value pair (key is a fixed-length integer rep. item)
- A set of hash functions r
- One fingerprint hash function f
- Key = keySum, value = fpSum (fingerprint sum)

Insertion/Deletion:

- Use all r's to designate |r| locations of each key inserted
- Each key is XOR added into keySum
- Use f to compute fingerprint of the key
- Each fingerprint is XOR added into fpSum

Notice the XOR, inserting the same item removes the item

IBLT

Listing entries: (a peeling process)

- Process can start with at least one cell containing one item: f(keysum) = fpsum
- Use the key to retain the item
- Insert the key again to remove it from the table
- Previous step creates more one item cells
- Successfully end with table empty (all zeros)
- Or fail with no more one item cells

IBLT Example

2 hash functions: 2 insert locations

One fingerprint hash function

One difference between sets

IBLT size 3

Alice Key	11100	00011	00100
Key fp	10101	10100	11000
Bob Key	00011	11100	00010
Key fp	10100	10101	01100

keySum	00000	00000	00000
fpSum	00000	00000	00000

IBLT Example

Insert Alice's first key

2 hash functions: 2 locations

Alice Key	11100	00011	00100
Key fp	10101	10100	11000
Bob Key	00011	11100	00010
Key fp	10100	10101	01100

keySum	11100	00000	10101
fpSum	10101	00000	10101

IBLT Example

Insert Alice's second key into IBLT

2nd and 3rd cell

3rd cell is XOR

Alice Key	11100	00011	00100
Key fp	10101	10100	11000
Bob Key	00011	11100	00010
Key fp	10100	10101	01100

keySum	11100	00011	11111
fpSum	10101	10100	00001

IBLT Example

Insert the Last item

Send to Bob

Alice Key	11100	00011	00100
Key fp	10101	10100	11000
Bob Key	00011	11100	00010
Key fp	10100	10101	01100

keySum	11000	00111	11111
fpSum	01101	01100	00001

IBLT Example

Bob Insert (delete) first item into IBLT

Hash function locate item to same cells

Alice Key	11100	00011	00100
Key fp	10101	10100	11000
Bob Key	00011	11100	00010
Key fp	10100	10101	01100

keySum	11000	00100	11100
fpSum	01101	11000	10101

IBLT Example

Insert Bob's second key into IBLT

Order of insertion does not matter

Alice Key	11100	00011	00100
Key fp	10101	10100	11000
Bob Key	00011	11100	00010
Key fp	10100	10101	01100

keySum	00100	00100	00000
fpSum	11000	11000	00000

IBLT Example

Insert the last item from Bob

Alice Key	11100	00011	00100
Key fp	10101	10100	11000
Bob Key	00011	11100	00010
Key fp	10100	10101	01100

keySum	00100	00110	00010
fpSum	11000	10100	01100

IBLT Example

Peeling process to find out Symmetric difference

Evaluate f(keysum) ==fpSum

Alice Key	11100	00011	00100
Key fp	10101	10100	11000
Bob Key	00011	11100	00010
Key fp	10100	10101	01100

keySum	00100	00110	00010
fpSum	11000	10100	01100

IBLT Example

Extract first key from IBLT out

Reinsert again to peel it off

Left with another key

Alice Key	11100	00011	00100
Key fp	10101	10100	11000
Bob Key	00011	11100	00010
Key fp	10100	10101	01100

keySum	00000	00010	00010
fpSum	00000	01100	01100

IBLT Example

We end with an empty IBLT

Retrieved symmetrical difference

Bob knows what Bob has

Alice Key	11100	00011	00100
Key fp	10101	10100	11000
Bob Key	00011	11100	00010
Key fp	10100	10101	01100

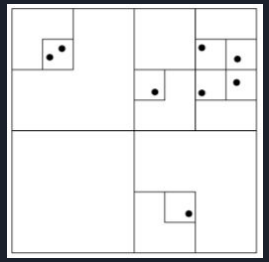
keySum	00000	00000	00000
fpSum	00000	00000	00000

Quadtree

Data structure in which each node has exactly 4 children

Each child is equal space partition of the parent

Every node recursively subdivided into four quadrants



Quadtree

Root = one cell containing entire Δ^d (d-dim grid), where S_A and $S_B \in [\Delta]^d$

Leaf = single cell of the gird

Every layer correspond to a resolution of the point set

Bottom Layer (all leaf): each element in the set has a corresponding cell with 1 number of item

Construction: (Better from bottom up)

- Number of points in a set that falls inside a cell
- Remove empty nodes
- Storing in a hashtable: key = cell coordinates, value = number of items in the cell

Quadtree Reconciliation

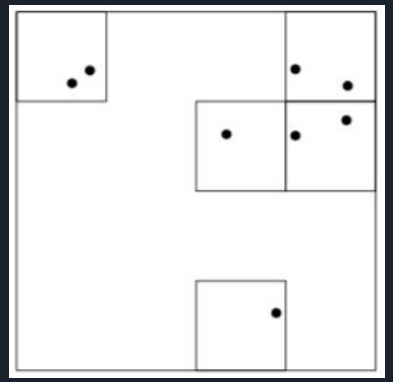
A successful extractable level of quadtree (all item and their position shown for visual)

Bob only knows count of items for each cell and cell coordinates, cells of symmetric difference

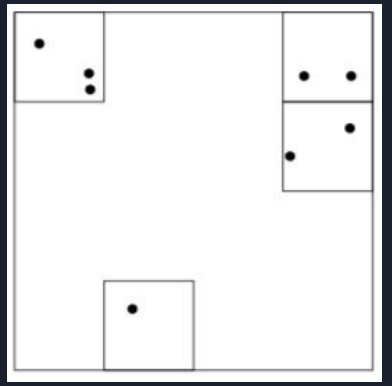
Bob (randomly delete a item if one-way) adds an item at the center of a new cell

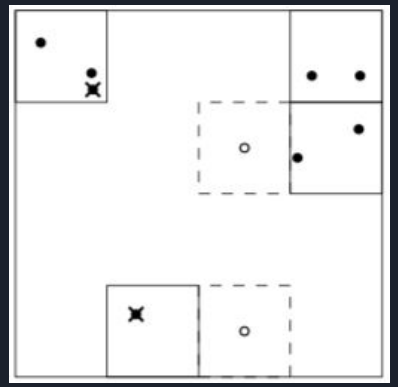
Notice: size of the cell is the tolerable differences

Alice



Bob





One-Way

Random Shift

Solving: False positive comparing different quadtrees

Improve the chance of similar item falls into the same grid

Pick a vector $\varepsilon^d = (\varepsilon_1,...,\varepsilon_d)$ uniformly at random from Δ^d

Forward: $S_A' = (S_A + \varepsilon^d)$ mode Δ^d

Reverse: $S_B = (S_B' - \varepsilon^d)$ mode Δ^d

Algorithm Workflow

Alice:

1. Random Shift: Shift all points by ε^d
2. Construct QuadTree: from bottom up to save space and time
3. IBLT: Insert all elements in a layer into a IBLT
4. Send Message: Send ε and the IBLT's (table size & number of levels depend on budget)

Bob:

1. Random Shift: Shift all points with same ε
2. Construct QuadTree
3. Decode IBLT: insert all elements from a layer to the IBLT from the same Alice's layer*
4. Move/Add points:
5. Reverse Random Shift: Shift all points back with same ε

Analysis

Communication Cost: $O(\alpha k \log(n\Delta^d)\log(\Delta))$

Computation Cost: $O(dn \log(\Delta))$

α = Redundancy factor
d = Dimension of an item
Δ = Size of grid including both sets (range)
n = Number of items in a set
k = Fixed number of symmetric differences to be reconciled

Requires less communication cost than existing exact if $\alpha k \lg(\Delta)\Delta^d/n < m$: two sets are large and dense with a significant amount of elements under tolerable differences.

Discussion

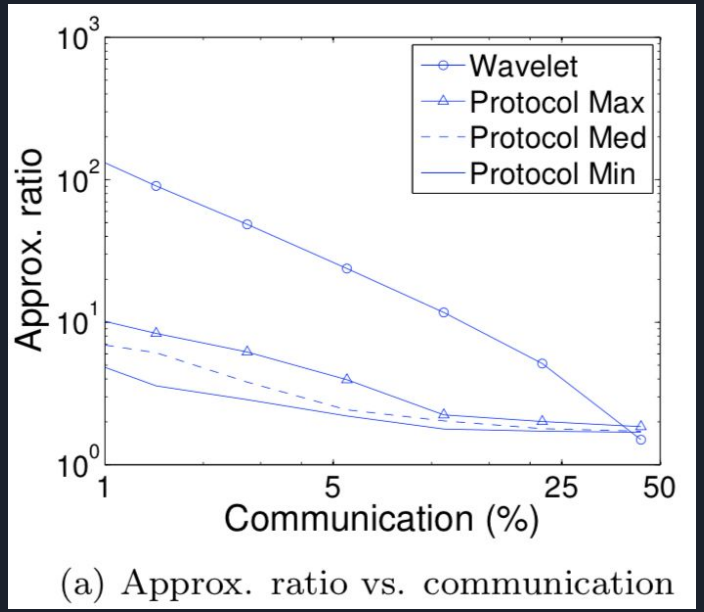
Pros

- Single round message (reduced latency)
- Message is compressed by IBLT
- Trade-off tolerable difference with communication budget
- One-way broadcast or Two-way reconcile

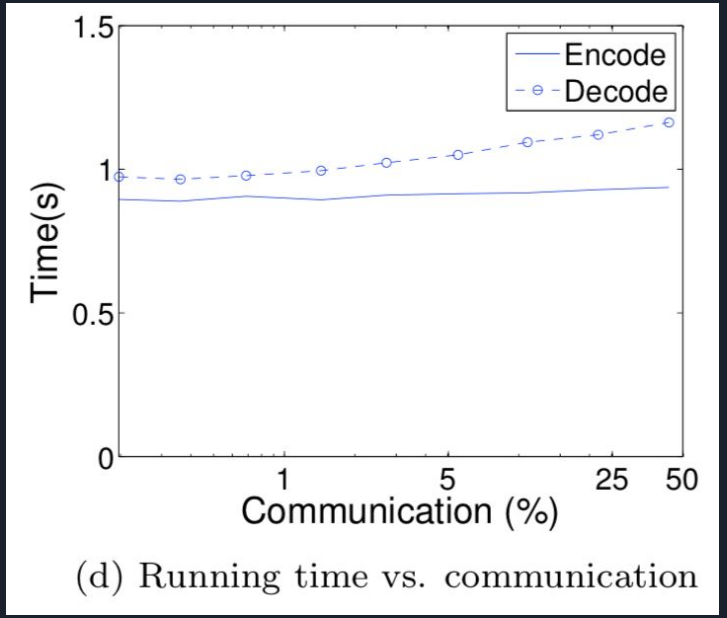
Cons

- Not Resumable: Cannot use the same IBLTs if new items are added, each layer is related
- Probabilistic model with high success rate but without quality guarantee
- If not at the lowest layer: (single element per cell)
  - Similar element can fell into different cells in a quadtree (False positive)
  - Reconcile from IBLT gives quadtree cells instead of the exact element, the algorithm regard the element at the center point of the cell (introduce error)

Experimental Performance

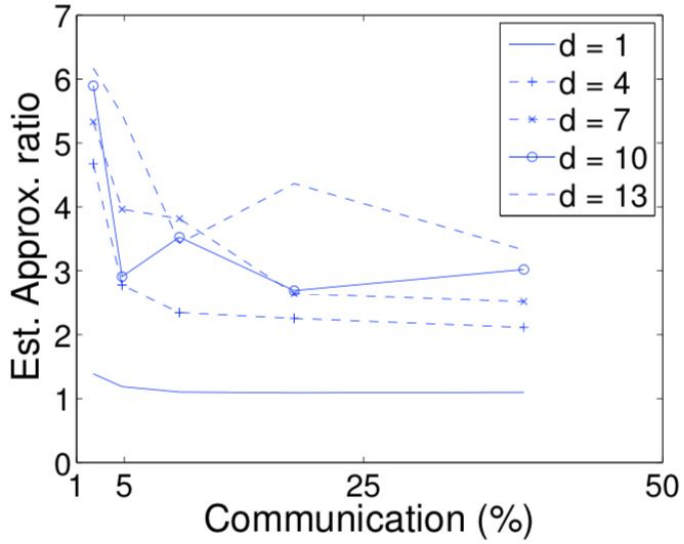


(a) Approx. ratio vs. communication

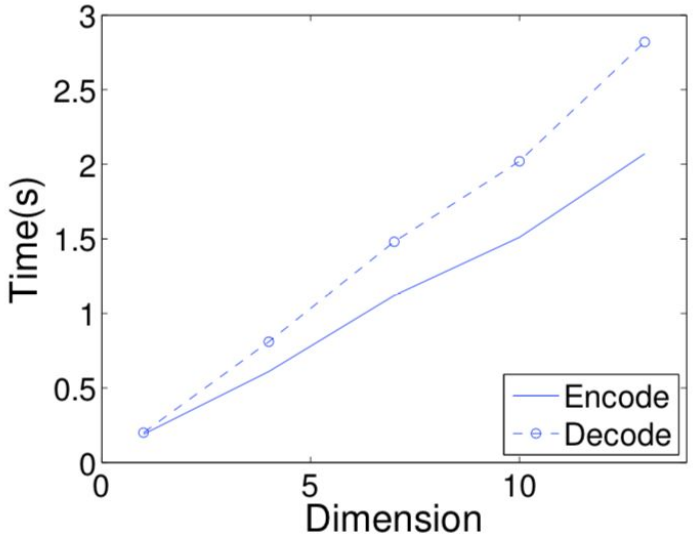


(d) Running time vs. communication

Experimental Performance



(a) Approx. ratio vs. dimension



(b) Running time vs. dimension

Conclusion

Protocol can control communication cost and quality (Trade-off)

Low communication and computation cost

Single-Round communication

Tight lower bound on communication budget

Future Work

Possible improvements through multi-round communication

Resumable reconciliation

Use of k-d tree to increase accuracy (define size of each grid)

Estimating set differences before setting budget

References

[1] D. Chen, C. Konrad, K. Yi, W. Yu, and Q. Zhang, “Robust set reconciliation,” in Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. ACM, 2014, pp. 135–146.

[2] Eppstein, David, et al. "What's the difference?: efficient set reconciliation without prior context." ACM SIGCOMM Computer Communication Review. Vol. 41. No. 4. ACM, 2011.

Thank you Q&A?