Lightweight Authentication of Freshness in Outsourced Key-Value Stores

Yuzhe Tang（汤宇哲）
Syracuse University（雪城大学）

Jan. 11, 2017 at XJTU

Introduction (1)

- Assistant Prof. at Syracuse University (雪城大学), New York

Introduction (2)

- Research:
- Security, applied cryptography,
- database systems,
- distributed computing.
- Group
- PhD students
- Projects:



1. Secure outsourced systems (in clouds)
2. Secure multi-party database systems

Lightweight Authentication of Freshness in Outsourced Key-Value Stores

Yuzhe Tang（汤宇哲），
T. Wang, L. Liu, X. Hu, J. Jang
Syracuse University（雪城大学）

IBM Research

ACSAC

What is Cloud?

What is Cloud?

Cloud Security

Cloud Security

- Cloud computing evolves to support critical missions

Cloud Security

- Cloud computing evolves to support critical missions
- Public cloud providers are “evil”.

Cloud Security is a Big Issue

- Cloud computing evolves to support critical missions
- Public cloud providers are “evil”.
- Security becomes #1 issue in growing cloud adoption!

Focus of this Work

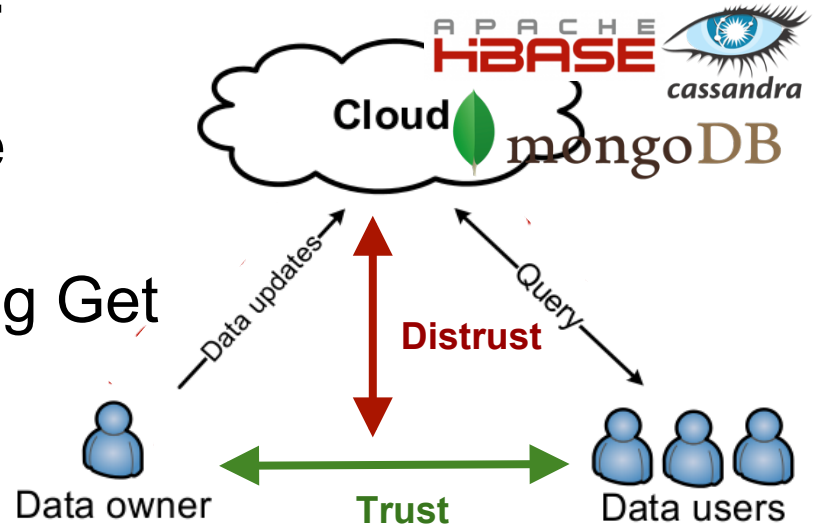
CryptDB: End-to-end encryption, SQL query over ciphertext

Our focus:

- Outsourced KV-store in clouds: Put/Get API
- Authenticating freshness:
- "Does the cloud 'modify' my data?" (Authenticity)
- Not about confidentiality/secrecy
- "Does the Get result return the latest version of data"

Research Formulation

- Three-party model (ADS):
- A trusted **source**: issuing Put
- An untrusted **server**: KV store on public cloud (Amazon S3)
- Several trusted **clients**: issuing Get



- Outsourcing computation/storage
- Assumptions: PKI, fresh digest (replay attack)

Research Formulation

Set-membership authentication scheme:

- Scheme of three algorithms.:

1. $\{S', d'\} \leftarrow$ **update**(upd, S)
2. $\Pi(x) \leftarrow$ **query**(x, S)
3. {accept, reject} $\leftarrow$ **verify**(x, Π(x), d)

- Security def.
- Adversary can choose updates in a set
- Unforge-ability (neg. probability to accept a forged result)

Baseline Construction: MerkleDigest-then-Sign

Merkle tree based auth.(KoMT)

Update: Digest and sign

MerkleDigest([1,2,3,5:X,9])= ★

Query: Proof construction

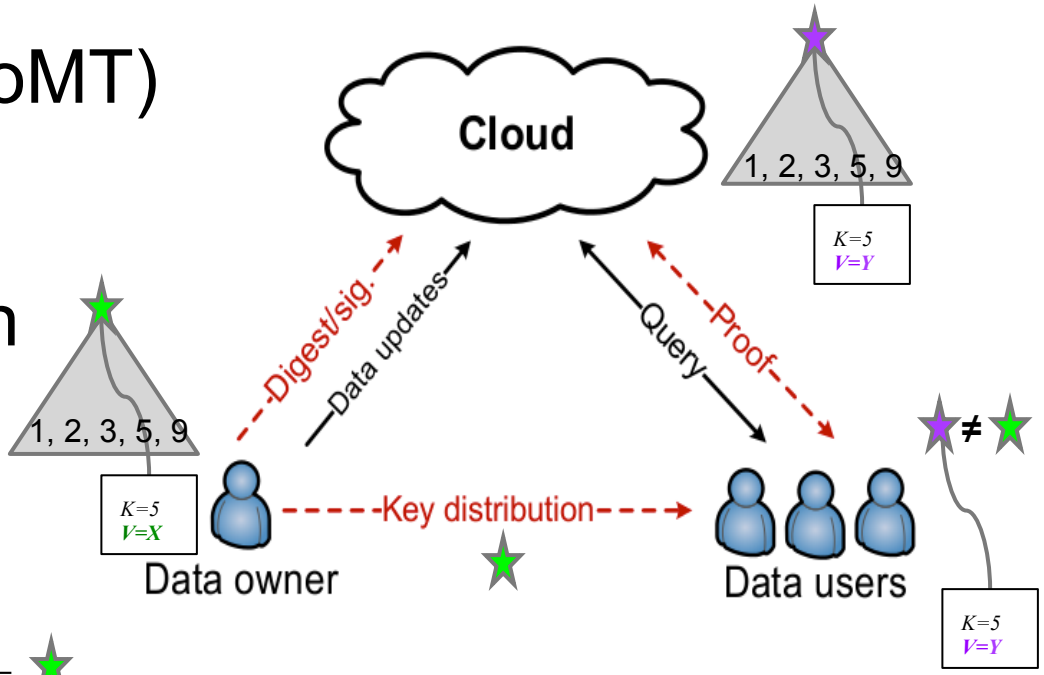
*Honest: AuthPath(h(<5,**X**>))*

*Dishonest: AuthPath(h(<5,**Y**>))*

Verify

*Honest: AuthPath(h(<5,**X**>)) =* ★

*Dishonest: AuthPath(h(<5,**Y**>)) =* ★ ≠ ★



Problem of MerkleDigest-then-Sign

- Works fine with static data (i.e. no Put)
- In the case of dynamic data, it
- Either maintains full copy of data locally
- Or maintains digest updated by a 2-round protocol (UIP)
- Neither is efficient

- Our research aims at *update-efficient Merkle tree for freshness authentication*.

Idea: Combining MHT with BF

Goal:

When MHT grows out of local “memory”, it builds a **Bloom-filter digest** before “flushing” local-state.

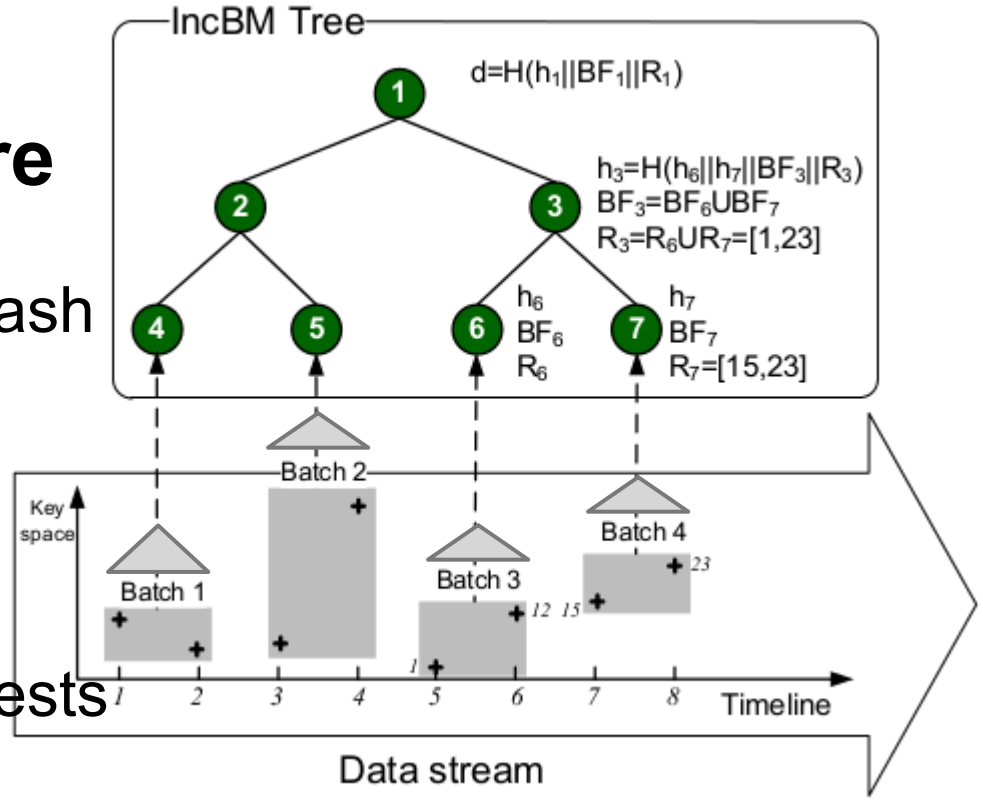
Based on BFs, it builds another Merkle tree (**IncBM tree**) to improve query performance.

IncBM tree

1. Structure, 2. maintenance, and 3. query processing

IncBM Tree: Structure

Per-node data digest and hash
- Bloom-filter BF
- Range digest R



IncBM: Merkle tree with digests

$$
\begin{aligned}
R(\text{node}) &= R(\text{left\_child}) \cup R(\text{right\_child}) \\
BF(\text{node}) &= BF(\text{left\_child}) \cup BF(\text{right\_child}) \\
h(\text{node}) &= H(h(\text{left\_child}) \| h(\text{right\_child}) \| BF(\text{node}) \| R(\text{node}))
\end{aligned}
$$

IncBM Tree: Update

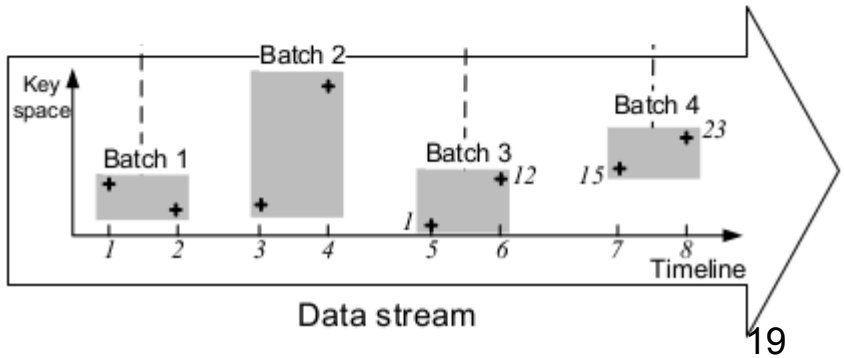
Signing a stream of data-updates

Workflow:

1. Batching data updates
2. Build MHT (upon data size reaches memory capacity)
3. Updating local partial IncBM tree
4. Sign the root of local IncBM tree, and upload it to the cloud

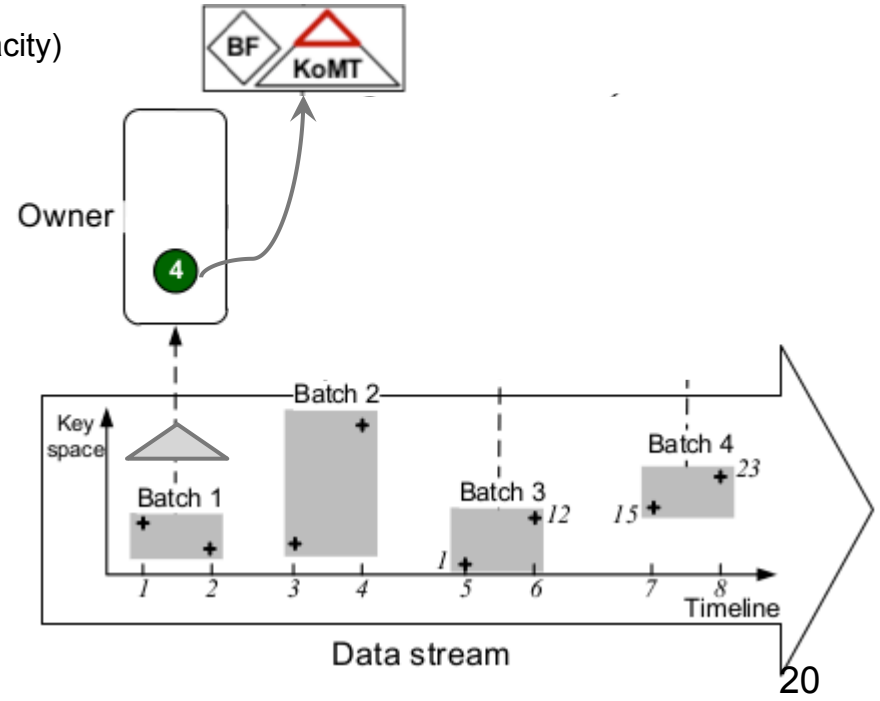
IncBM Tree: Update Workflow (1)

1. Batching data updates
2.
3.



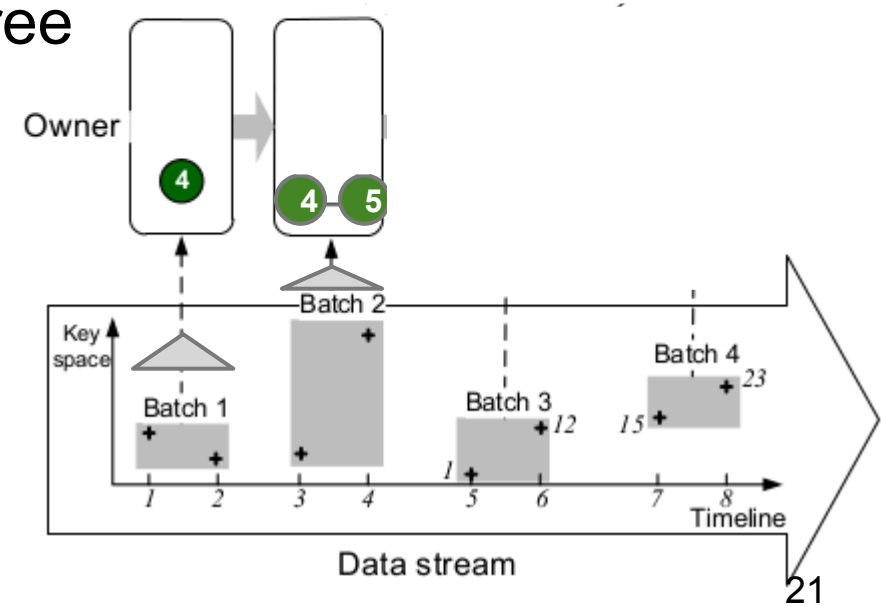
IncBM Tree: Update Workflow (2)

1. Batching data updates
2. Build MHT (upon data size reaches memory capacity)
3.



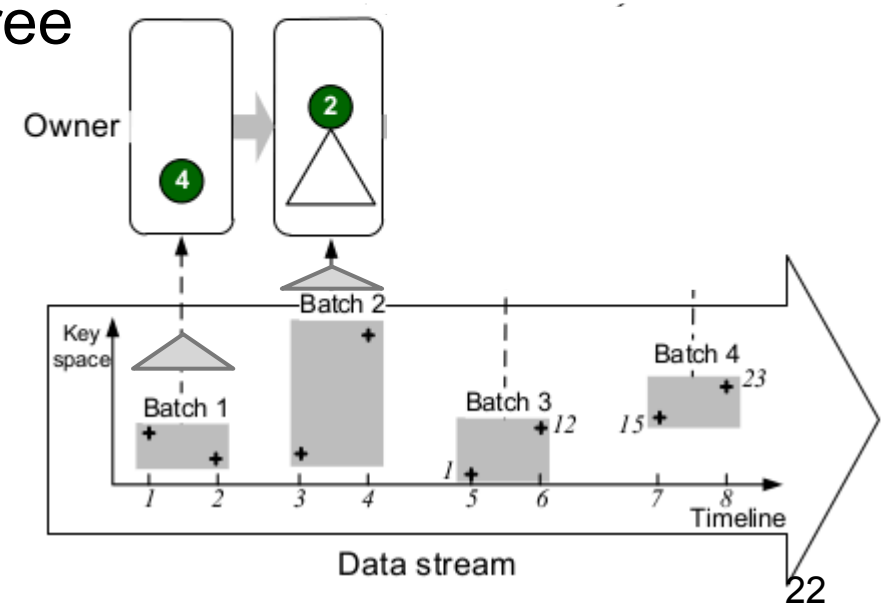
IncBM Tree: Update Workflow (3)

1. Batching data updates
2. Build MHT (upon data size reaches memory capacity)
3. Updating local partial IncBM tree



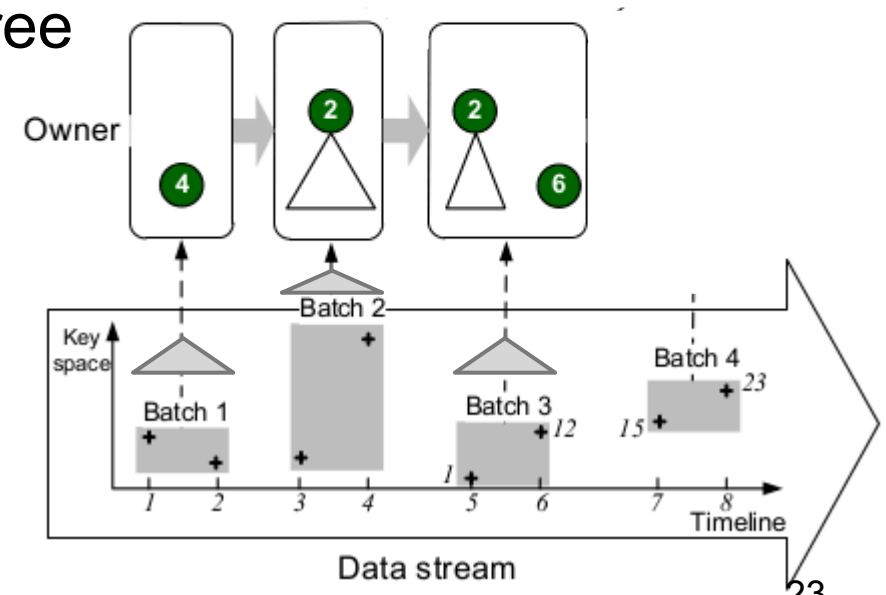
IncBM Tree: Update Workflow (4)

1. Batching data updates
2. Build MHT (upon data size reaches memory capacity)
3. Updating local partial IncBM tree



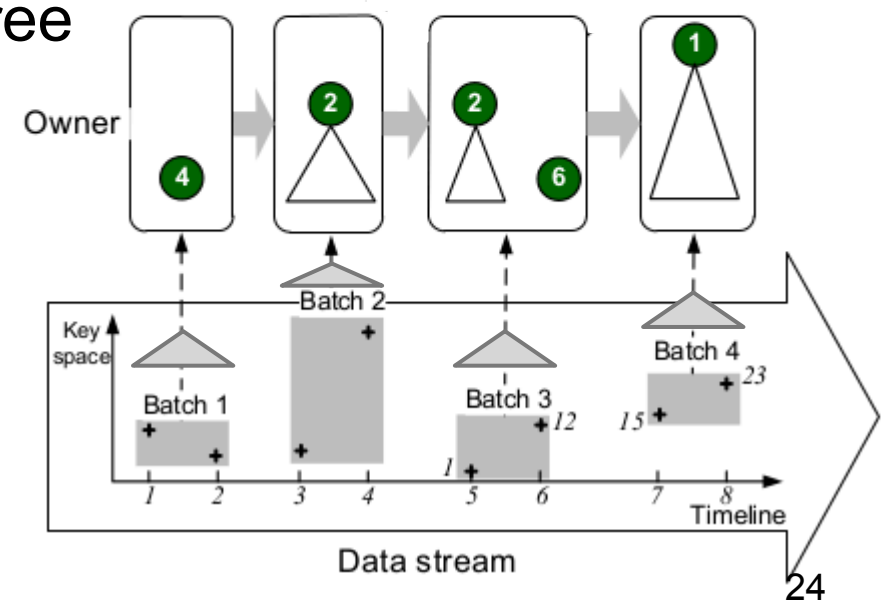
IncBM Tree: Update Workflow (5)

1. Batching data updates
2. Build MHT (upon data size reaches memory capacity)
3. Updating local partial IncBM tree



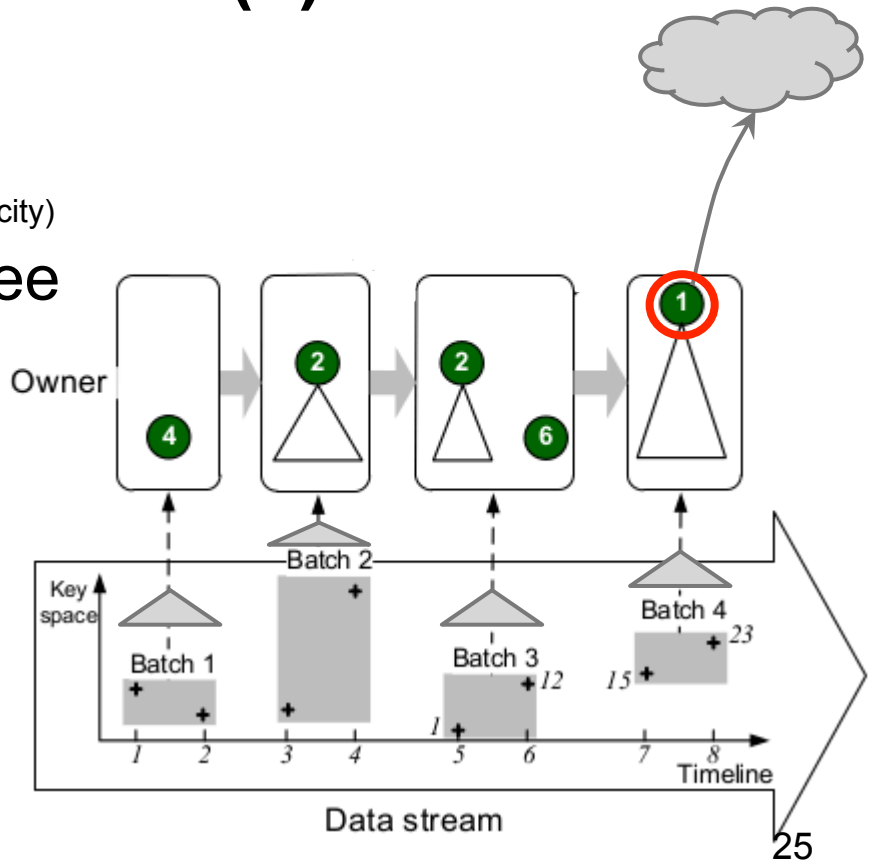
IncBM Tree: Update Workflow (6)

1. Batching data updates
2. Build MHT (upon data size reaches memory capacity)
3. Updating local partial IncBM tree
4.



IncBM Tree: Update Workflow (7)

1. Batching data updates
2. Build MHT (upon data size reaches memory capacity)
3. Updating local partial IncBM tree
4. Sign the root of local IncBM tree, and upload it to the cloud

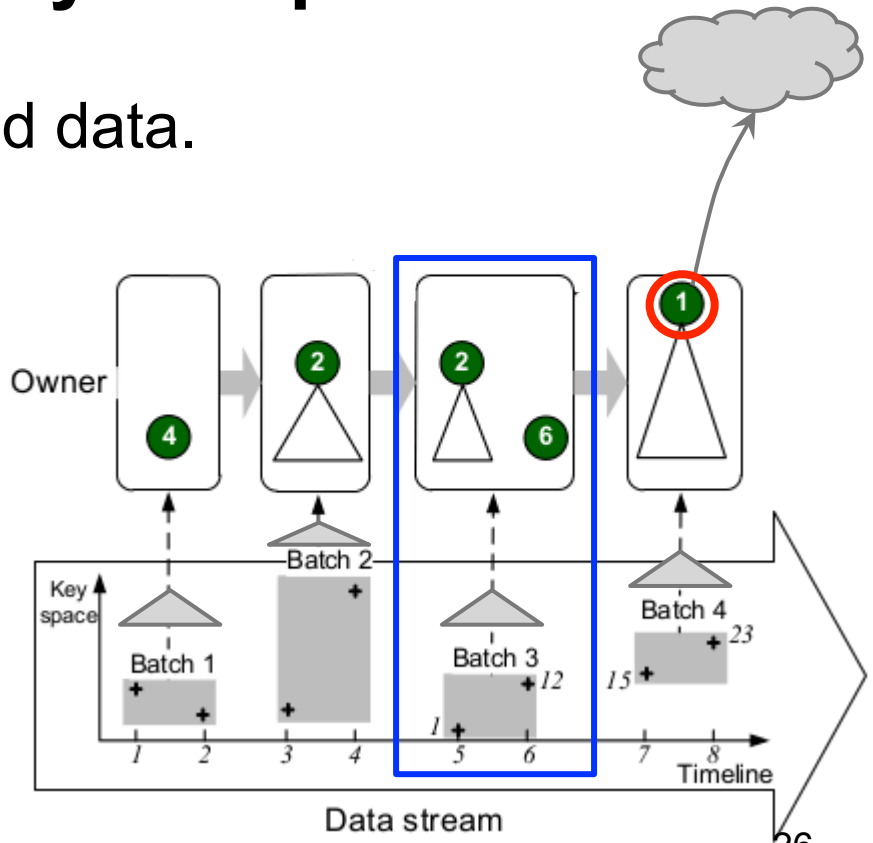


IncBM Tree: Small Memory Footprint

Node ① signs all **4** batches of sorted data.

Yet, at any time, it only stores
at most **1** batch with extra digest.
Data batch as large as mem-cap.

Using **1** memory space it signs **4**
mem worth of data.



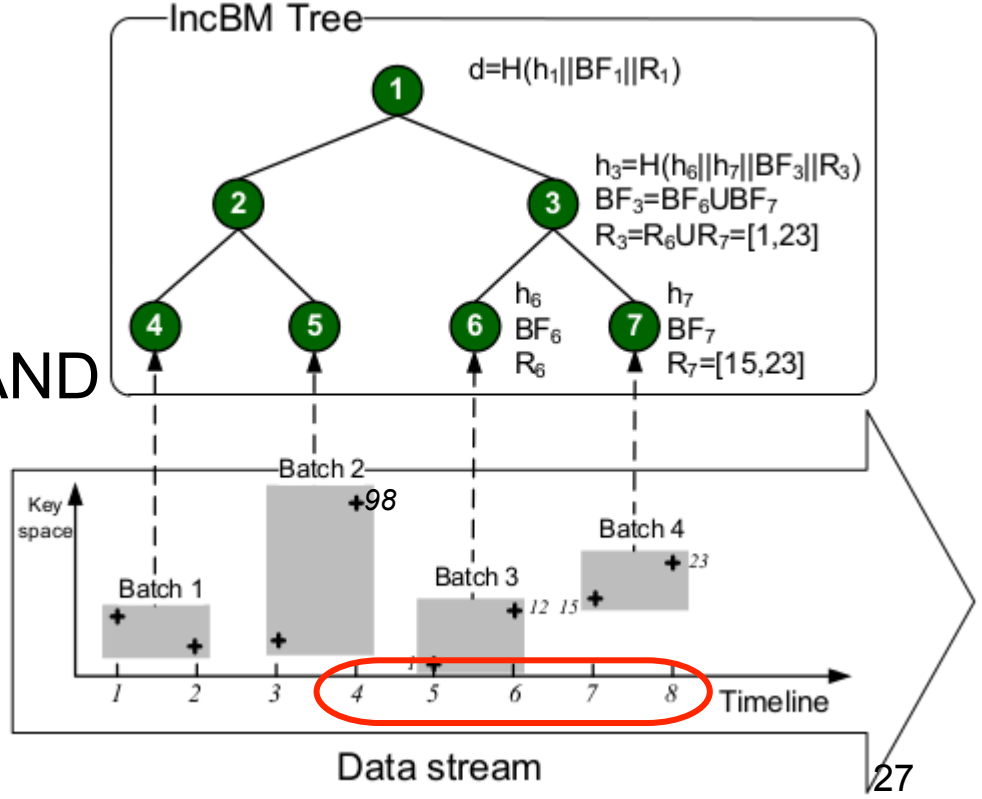
IncBM Tree: Query Proof Construction

Prove:

"Key 98 is fresh as of time 8"

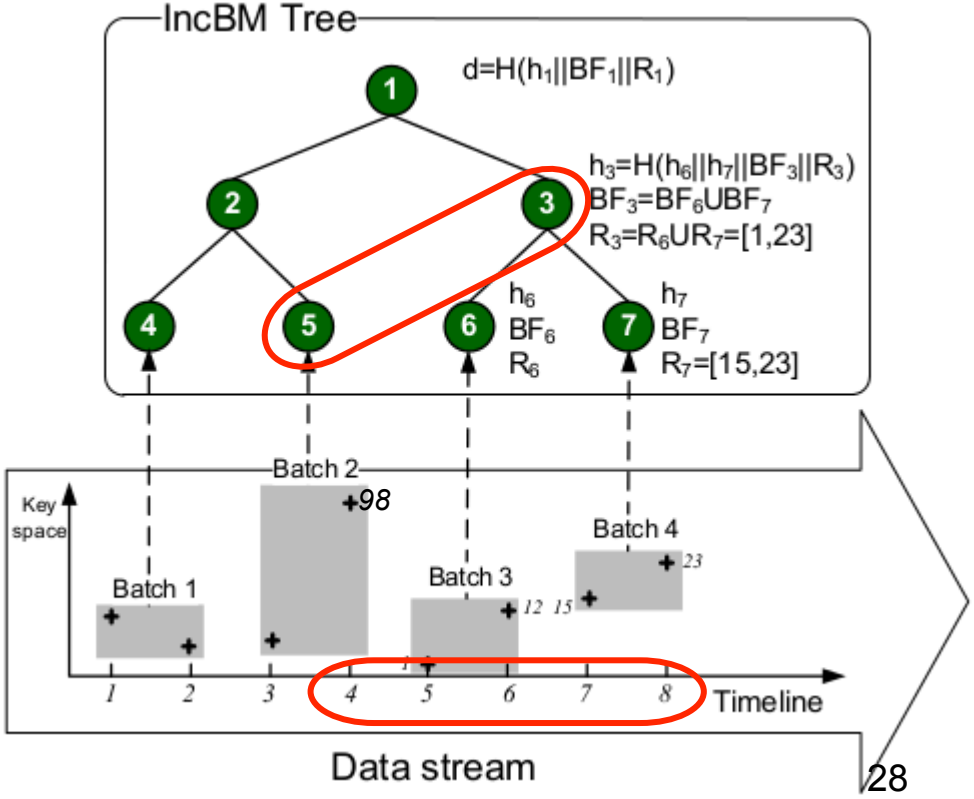
Equivalent to say:

1. "There is key 98 at time 4" AND
2. "There is no key 98 in time range [5,8]"



IncBM Tree: Query Proof Construction

1. "Key 98 is there at time 4"
   can be proved by node 5
   using Merkle root
2. "Key 98 is not in [5,8]"
   maybe proved by BF
   by node 3
   if not, go down to
   node 6 AND 7



Prototype implementation: HBase

Digest and sign (**OpenSSL**)

Signer produces signed root

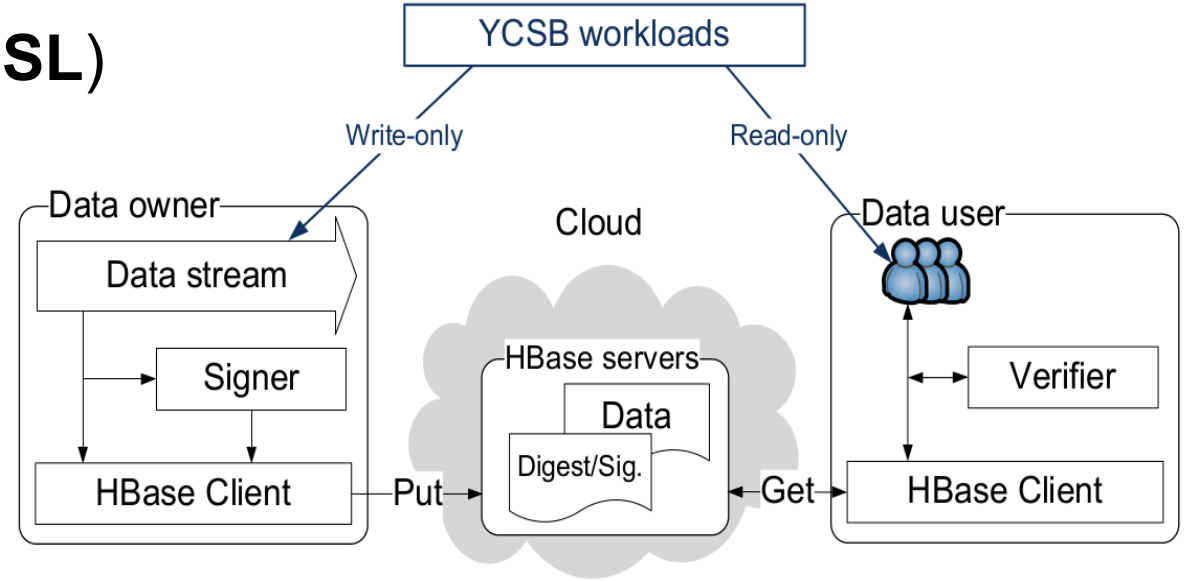
submit to cloud thru *Put* call

HBase in cloud:

Key-value stores:

Write-optimized

Strong consistency



Tables: Meta-data sharded by time (digest, signatures), base data by key

Proof verification by users

Proof construction in cloud, send back along w *Get* call

Get result verified by user

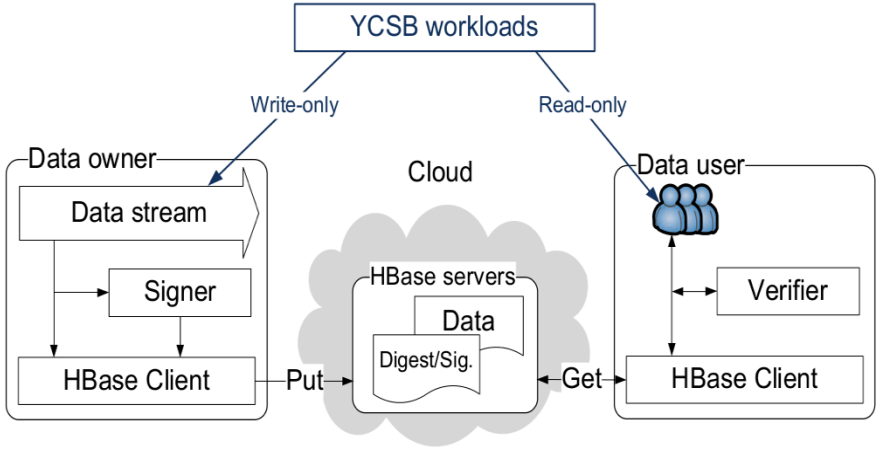
Performance study: Setup

Data generated by **YCSB**

*Get*s thru. read-only workload

*Put*s thru. write-only workload

500 GB data poured into a 10-node HBase cluster



Evaluated in **Emulab**

Owner, HBase cluster, users are on separate machines

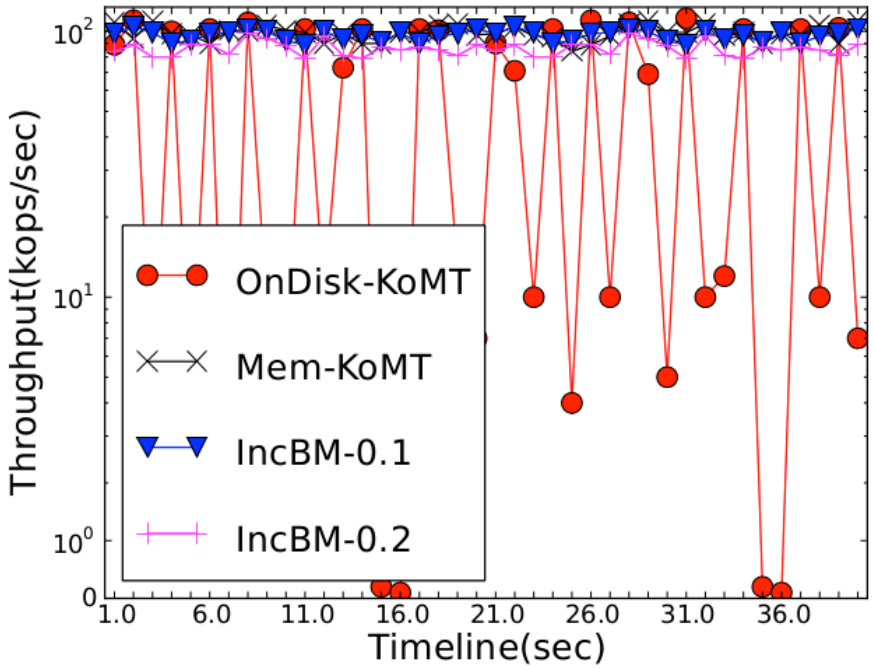
Commodity machines: 2GB memory

Performance of Put (write and sign)

KoMT-onDisk: MHT size triples mem. cap.
KoMT-mem: MHT size equals mem. cap.
IncBM-0.1: 10% mem for local partial IncBM
IncBM-0.2: 20% mem for local partial IncBM

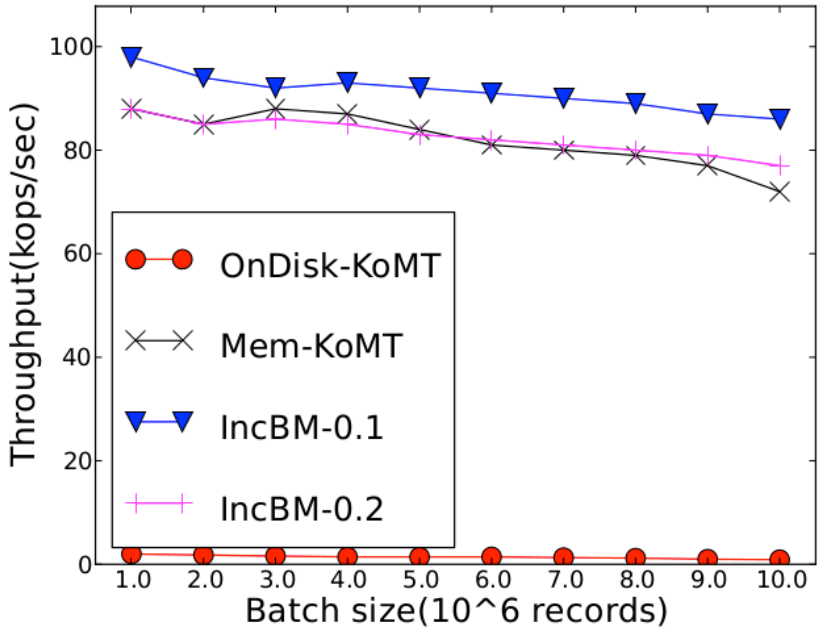
Spikes in KoMT-onDisk due to flushing data onto disk.

Others' write-performance is similar.

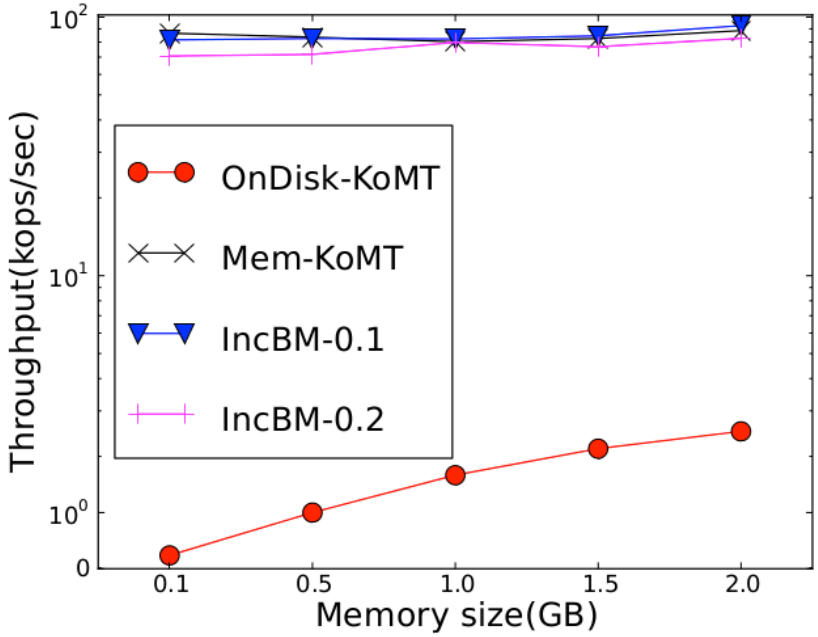


(a) Time series

Performance of Put (write and sign)



(b) Varying batch sizes

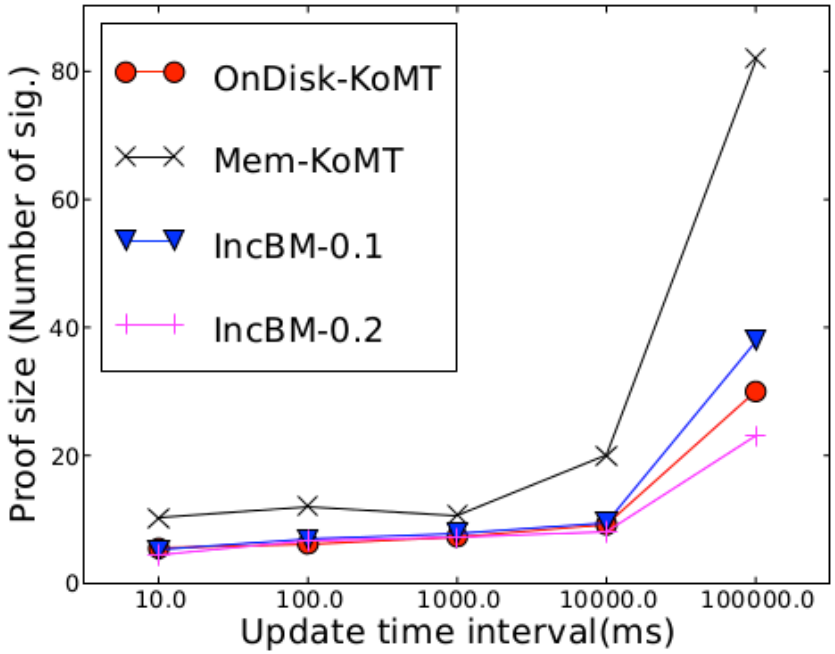


(c) Varying memory sizes

Performance of Get (proof and vrfy)

Mem-KoMT is slowest because small batch size (Verifying the same time interval requires more digital signature verifications)

IncBM is the same efficient to OnDisk-KoMT on query performance



(b) Proof size

Summary

Articulated the problem of providing data freshness assurance for outsourced multi-version key-value stores.

Proposed INCBM-TREE:

1. lightweight for both data owners and end users,
2. optimized for intensive data update streams,
3. immediate authentication of data freshness in the presence of real-time and historical data accesses.

Referenced Work

[1] Feifei Li, et al, Proof-infused streams: enabling authentication of sliding window queries on streams, VLDB'07

[2] Raluca Ada Popa, et al, Enabling security in cloud storage SLAs with CloudProof, USENIX-ATC'11

[3] Emil Stefanov, et al, Iris: a scalable cloud file system with efficient integrity checks, ACSAC'12

Questions?

Thank you

Contact:
Yuzhe (Richard) Tang
Syracuse University
Email: *ytang100@syr.edu*
Web*: ecs.syr.edu/faculty/yuzhe*

Backup Slides

Near-Real-Time Authentication

Fresh as of now

- owner signs updates every r minutes
- assumption: sync. time service (e.g. by NIST)

Freshness authentication relaxed by r-minute interval

- absence of a proper freshness proof for data > r minute ago means misbehavior of cloud

Auth. Framework (1): Baseline

Authentication data flow

Digest and sign (periodically)

Proof construction

Proof verification