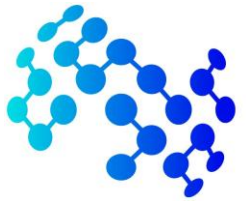
SolFS: An Operation-Log Versioning File System for Hash-free Efficient Mobile Cloud Backup

Riwei Pan[1], Yu Liang[2], Lei Li[1], Hongchao Du[1], Tei-Wei Kuo[3], Chun Jason Xue[4]

[1]City University of Hong Kong [2]ETH Zürich [3]National Taiwan University
[4]Mohamed bin Zayed University of Artificial Intelligence



MOHAMED BIN ZAYED
UNIVERSITY OF
ARTIFICIAL INTELLIGENCE

Outline

香港城市大學
City University of Hong Kong

- Background and Motivation
- Design and Implementation
- Evaluation
- Conclusion

Background and Motivation

Cloud Backup on Mobile Devices

香港城市大學
City University of Hong Kong

- **Mobile cloud backup are widely employed on existing mobile devices for different purposes**
  - Vendor built-in and third-party mobile cloud applications (e.g., Oppo and Dropbox)
  - data security
  - save storage space
  - Smartphone snapshot

- **Importance of Frequent Cloud Backup.**
  - 💰 Financial data
  - 📄 Business documents
  - 🏥 Health-care-related data

There is dire need of a high-performance mobile cloud backup solution

Motivation: Expensive File Hashing

香港城市大學
City University of Hong Kong

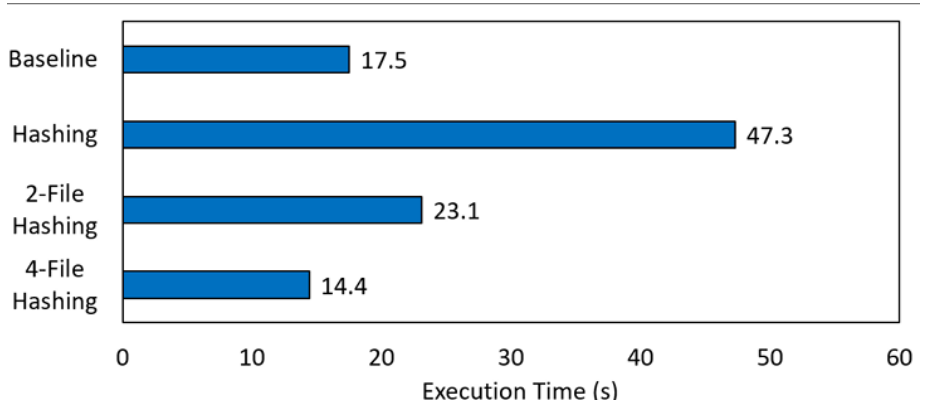
➢ Limitations of existing mobile cloud backup schemes

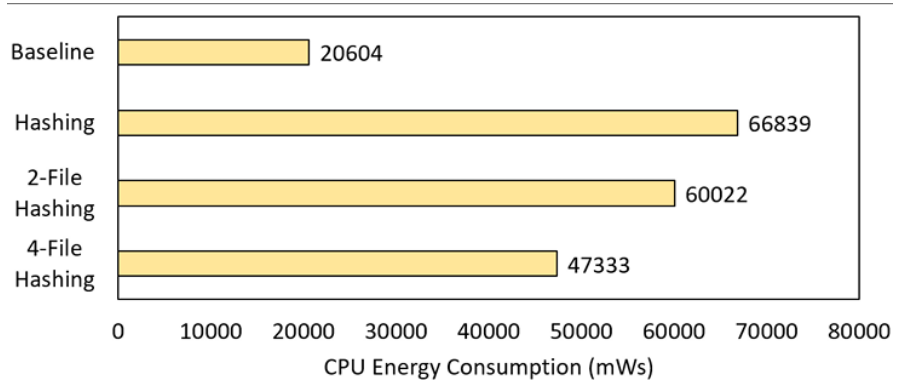
- Direct synchronization → *high network traffic with low CPU usage.*
- Delta synchronization → *low network traffic with high CPU usage.*

➢ Delta synchronization is not satisfactory on resource-limited mobile devices

- ❌ Intensive hash computation.
- ❌ High CPU usage.
- ❌ Compete CPU resources with users' applications.
- ❌ Drains battery.

Hashing introduces 170% execution latency and 224% CPU energy consumption on average.





Challenges

香港城市大學
City University of Hong Kong

- **Goals:**
  - ✓ Upload only modified data (like Delta Sync) ***WITHOUT*** expensive hashing.

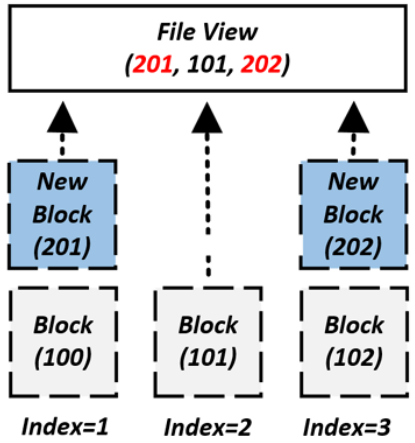
- **Insight:** If we can track all the modified ranges of a file since the last backup, we could directly upload the modified range of data without hash computation.
  - File System is the suitable component to achieve this goal.

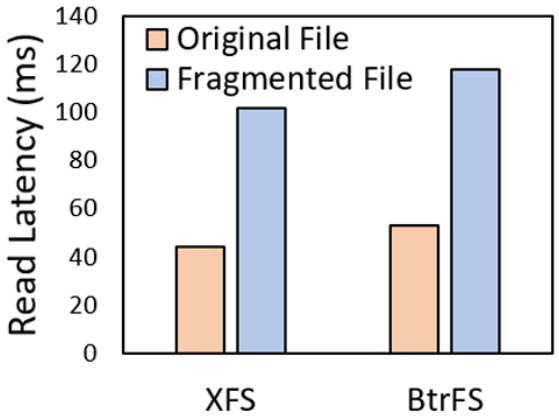
Challenges

香港城市大學
City University of Hong Kong

Limitations of file versioning techniques:

- Copy-on-write (CoW) in XFS and BtrFS:
  - ✓ The reflink feature help identify the file modified data range.
    - ❌ Result in File fragmentation issue.
    - ❌ Require changing file system layout.
- Undo Logging in WaybackFS:
  - ❌ It has to store and manage multiple versions of file data, resulting in high storage overhead.





Existing FS techniques are unsuitable for efficient, low-overhead mobile backup.

Design and Implementation

Design Goals

香港城市大學
City University of Hong Kong

Our solution: operation log versioning file system (SolFS)

The main design goals:

- **Hash-free Identification**: identify modified ranges of file data without hashing.
- **Overhead Minimization**: No file fragmentation issue and no significant and memory and storage overhead.
- **Generality**: Support multiple backup APPs (e.g., built-in and third-party) on the same device to manage their respective file differences without interfering with each other.
- **Backward Compatibility**: There are over 6.5 billion of mobile devices in the world and we expect that all existing mobile devices could benefit from SolFS. → Do not change file system layout.

Design: Overview

SolFS involves three primary design points:

- **Per-file Mergeable Operation Logging**
  - Track modified file data range via operation log (i.e., a pair of **each write's offset and length to a file**)
- **Mlog Versioning**
  - Manage different operation logs from different backups and different backup applications.
- **Hash-free Delta Synchronization**
  - Calculate a file's modified file data ranges without hashing since the last backup based on multi-version operation logs.

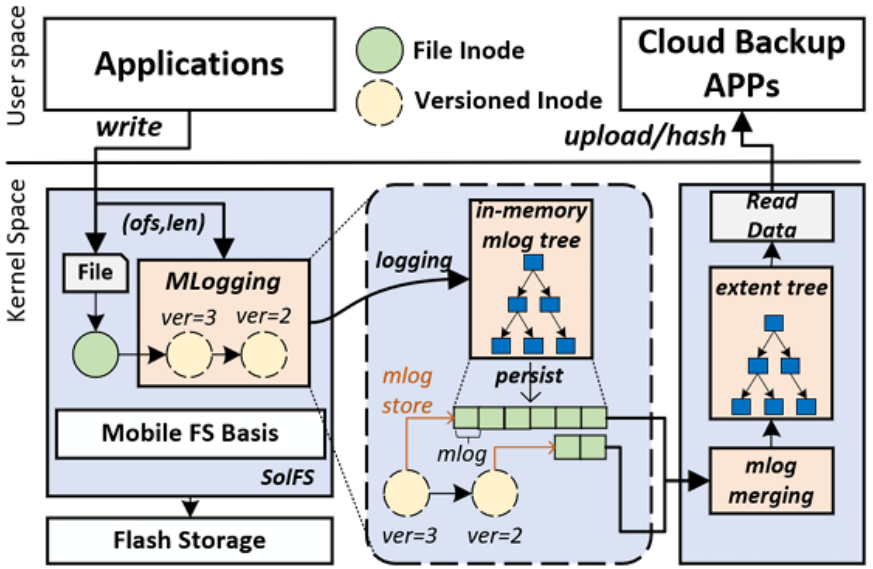


Figure 3: SolFS architecture (shown in blue).

Design: Per-file Mergeable Operation Logging

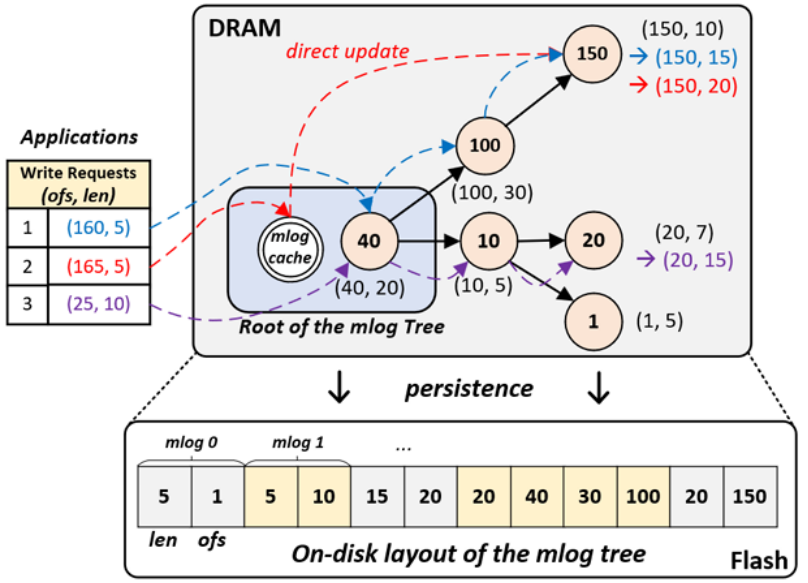
香港城市大學
City University of Hong Kong

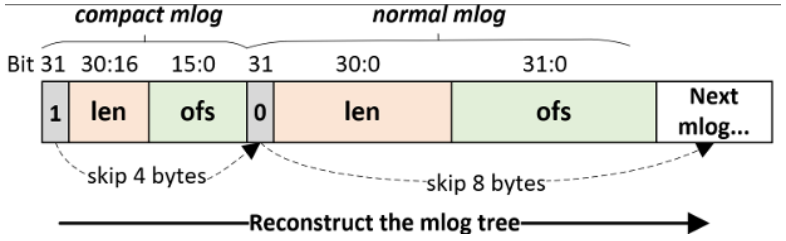
Per-file Mergeable Operation Logging (MLogging)

- **Mergeable Operation Log Tree**
  - Record and merge a file's operation log (or mlog for short) based on an in-memory extent tree.
  - Fast path optimization for append and sequential writes.

- **Persistence with Compact Mlogs**
  - Persist operation logs into an additional inode for continuous tracking.
  - Use compact form of mlog to minimize storage overhead

- **Dynamic Granularity Alignment**
  - Byte-level or page-level mlogs
  - Convert byte-level mlogs to page-level mlogs to further reduce the memory and storage overhead.





Design: Mlog Versioning

香港城市大學
City University of Hong Kong

Mlog Versioning

- **Versioned Inodes for Mlogs**
  - Mlogs are stored in versioned inodes.
  - Four additional attributes, *ino_ver, ver_link, ino_flag and next_ino,* are added into the inode's extended file attribute area to allow inode versioning and ensure backward compatibility.

- **Backup-driven Versioning**
  - The inode version increases when performing a new backup.
  - For each new version, SolFS will insert a new inode and form a versioned inode chain.
  - Multi-versioned inodes allow different cloud backup applications to manage their own file differences to ensure generality.

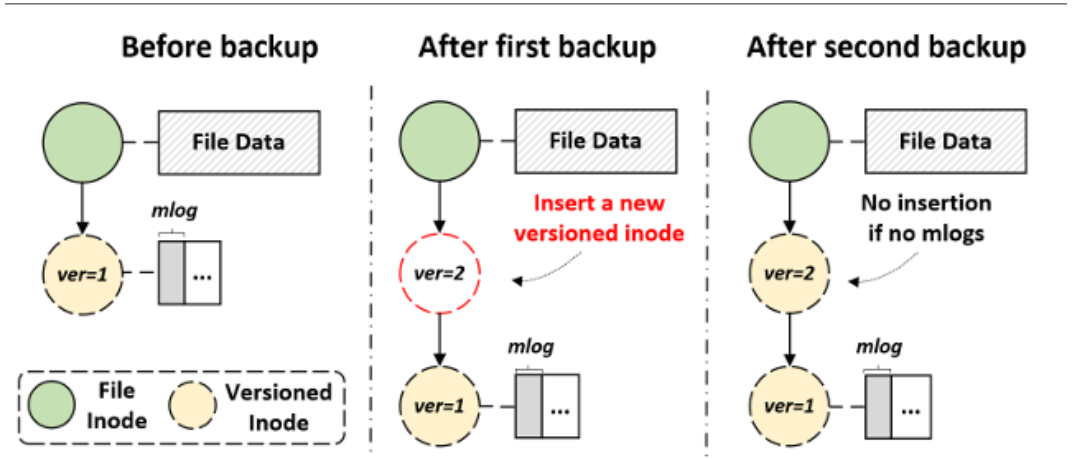


Figure 6: Backup-driven versioning.

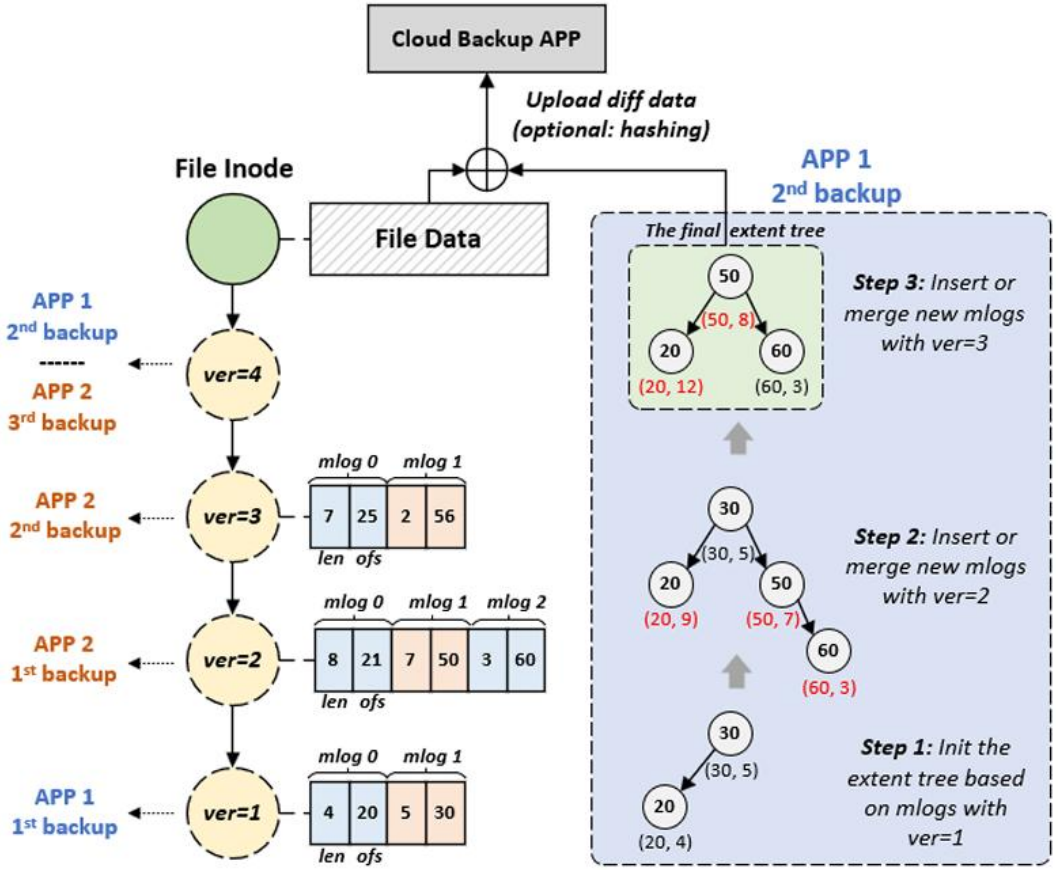
Design: Hash-free Delta Synchronization

香港城市大學
City University of Hong Kong

Hash-free Delta Synchronization

- **Collaborative Version Management:**
  - The version number is managed by the local device and the remove server.
  - The version difference between the local device and the remote server hints SolFS to find the correct modified data ranges and upload the corresponding data.

- **Upload with the Extent Tree**
  - Based on the version difference, SolFS traverses the versioned inode chain to combine corresponding mlogs to form an extent tree. → File modified data range.
  - The data range helps cloud backup APPs to upload modified data without hashing.



Evaluation

Evaluation

香港城市大學
City University of Hong Kong

- **Mobile Platform:** Pixel 8 with Android 14 & Linux 5.15
- **Remote Cloud Server Platform:** Ubuntu 20.04 platform with a 24-core Intel i9 14900k and 32GB of memory
- **Baselines**:
  - **HashSync:**
    - chunking a file; performing MD5 hashing for each chunk; uploading modified chunks.
  - **Rsync:**
    - similar to HashSync and do rolling checksum on the local application side.
  - **WebR2sync+:**
    - similar to HashSync and do rolling checksum on the remote server side.

Evaluation: Cloud Synchronization Efficiency

香港城市大學
City University of Hong Kong

➢ **Cloud Sync Time compared with existing solutions**

- SolFS reduces the sync time by 88.8% on average in random updates.
- This is due to the hash-free design that mitigates intensive hash computation.

➢ **Network Traffic**

- SolFS shows remarkable low network traffic compared with existing solutions due to two reasons:
  - SolFS transmits version number for backup and it does not requires transmitting the chunk list that is related to the file size.
  - SolFS can accurately upload data in byte level instead of chunk-level data of previous solutions.

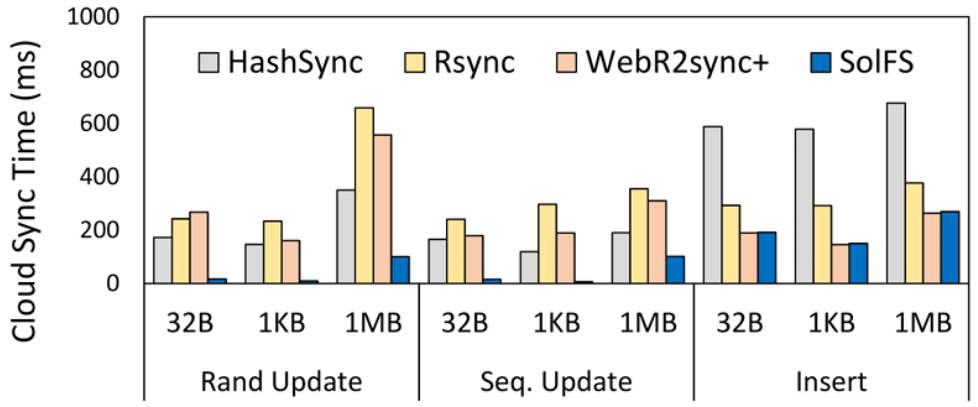


Figure 10: Cloud sync time on different types of file modifications.

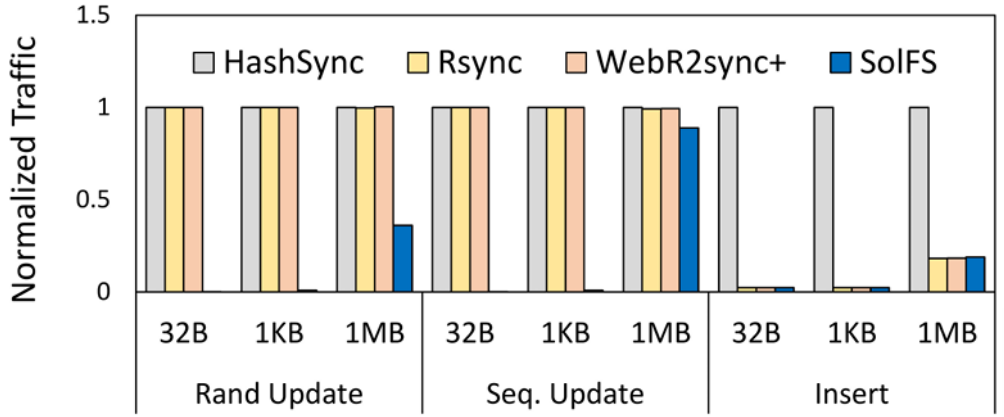


Figure 11: Normalized network traffic on different types of file modifications. HashSync serves as the baseline.

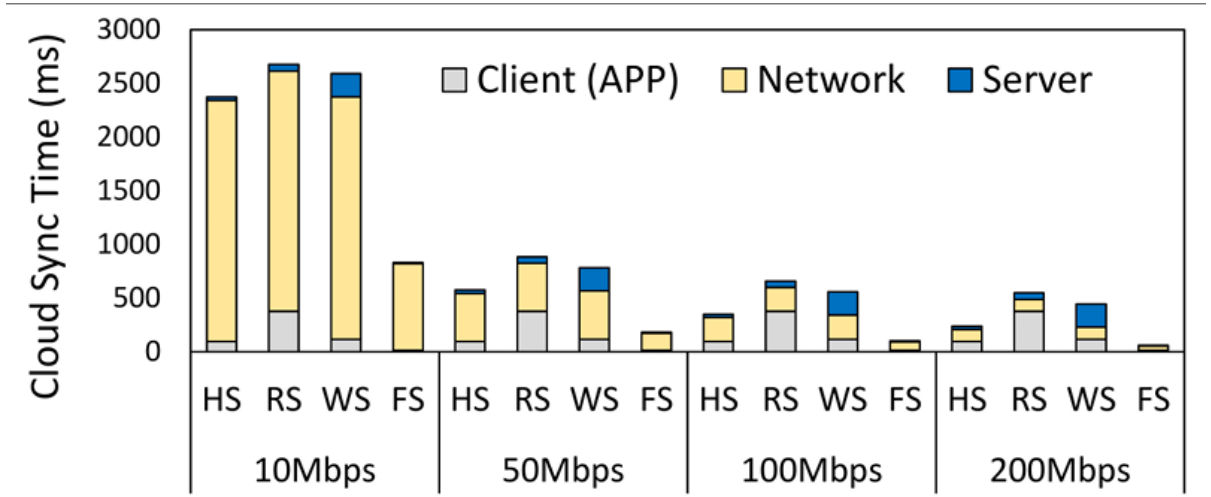
Evaluation: Sync Time Breakdown

香港城市大學
City University of Hong Kong

➢ **Sync Time Breakdown**

- SolFS outperforms other approaches by significantly reducing hash computation overhead on both the client and server sides (92% on average).

- **Software Overhead (Client + Server) Contribution:**
  - **50Mbps:**
    - HashSync: 21.8%
    - Rsync: 49.3%
    - WebR2sync+: 42.2%
    - SolFS: 10.8%
  - **200Mbps:**
    - HashSync: 52.7%
    - Rsync: 79.5%
    - WebR2sync+: 74.5%
    - SolFS: 32.7%



☐ As network bandwidth grows, computation on both the client and server sides lead to significant software overhead, becoming the bottleneck.

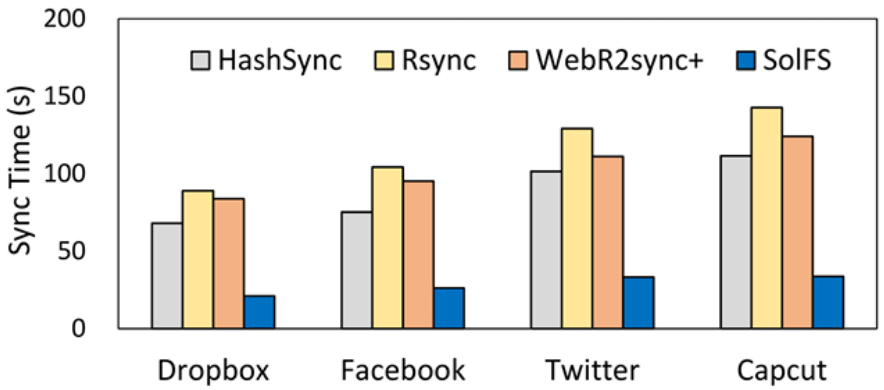
✓ SolFS mitigates this issue by minimizing hash computation.

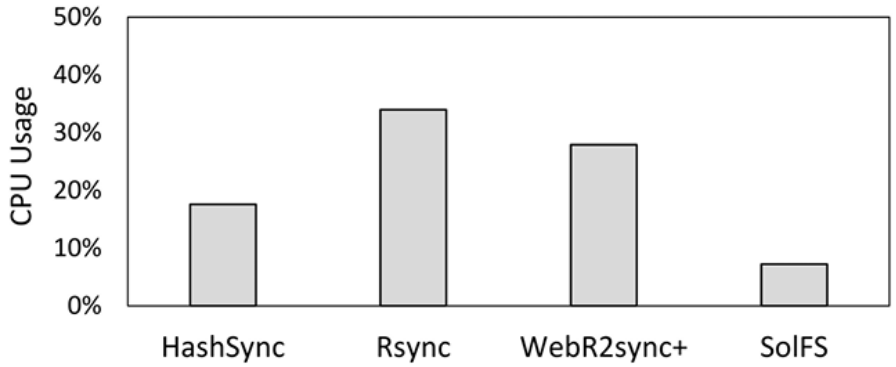
Evaluation: Mobile APP Workloads

香港城市大學
City University of Hong Kong

➢ Mobile APP Workloads

- Improve 71% of sync time on average among four applications.
- Reduce around 70% CPU usage on average during cloud backup.

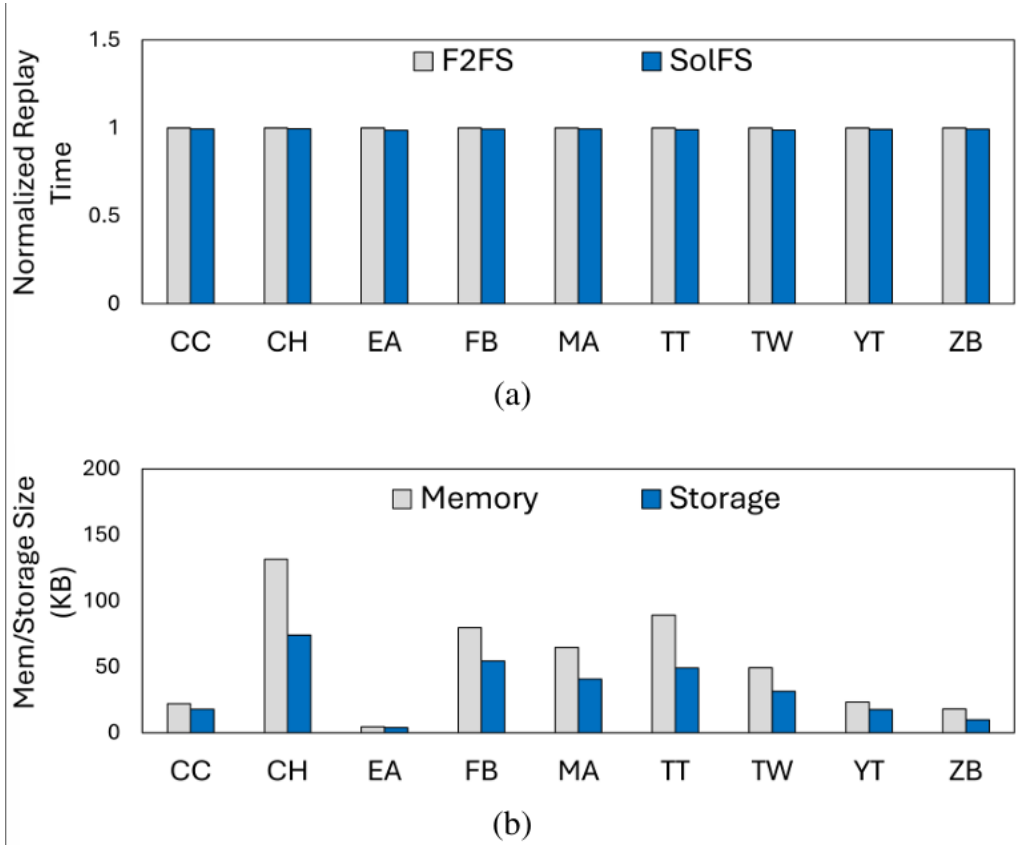




Evaluation: Overhead Analysis

香港城市大學
City University of Hong Kong

- **SolFS does not introduce significant overhead on normal operations:**
  - Replaying applications' operation traces.
  - < 1.5% read & write performance loss.
  - ~ 470KB additional memory consumption.

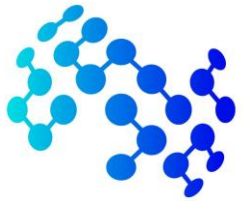


Conclusion

香港城市大學
City University of Hong Kong

- We identify the dilemma of mobile cloud backup between CPU overhead and cloud backup efficiency.
- We indicate that tracking write operation's offset and length can address the dilemma on mobile cloud backup.
- We design and implement SolFS to achieve hash-free file cloud backup while minimize its memory and storage overhead.
- We evaluate SolFS on mobile devices and show that SolFS can significantly improve mobile cloud backup efficiency.

Thanks for Listening



MOHAMED BIN ZAYED
UNIVERSITY OF
ARTIFICIAL INTELLIGENCE