Dsync: a Lightweight Delta Synchronization Approach for Cloud Storage Services

Yuan He, Lingfeng Xiang, Wen Xia, Hong Jiang,
Zhenhua Li, Xuan Wang, Xiangyu Zou

哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

UNIVERSITY OF
TEXAS
ARLINGTON

清華大學
Tsinghua University

Outline

- Background and motivation
- Design and implementation
- Performance evaluation
- Conclusion

Background and motivation

File synchronization

- Full sync:

  Simple, but not bandwidth-efficient

- Delta sync:

  Suitable for frequently modified files

  1. Fix-Sized Chunking:

     High compute overhead for comparing

  2. Content-defined Chunking:

     Lower compute overhead but require local buffer

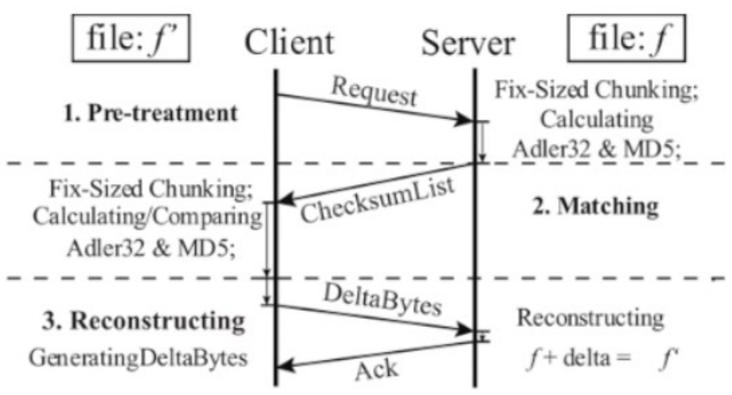
Source	Full Sync[1]	*Delta Sync*	
		Local Buffer	*Chunking*$_2$ *Methods*
DropBox (W/A) [8][3]	×	√	FSC
Seafile (W) [6]	×	√	CDC
Seafile (A) [6]	√	×	×
GoogleDrive (W/A) [8]	√	×	×
OneDrive (W/A) [8]	√	×	×
`rsync` [5]	×	×	FSC
DeltaCFS [34]	×	×	FSC
PandaSync [28]	√*	×	FSC
WebSync [32]	×	×	FSC
QuickSync [8]	×	√	CDC
LBFS [23]	×	×	CDC
UDS [20]	×	×	FSC
Dsync	×	×	CDC

1 √: full sync, ×: delta sync, and √*: selective full sync.
2 FSC: Fix-Sized Chunking, CDC: Content-Defined Chunking.
3 W: windows client, A: android client.

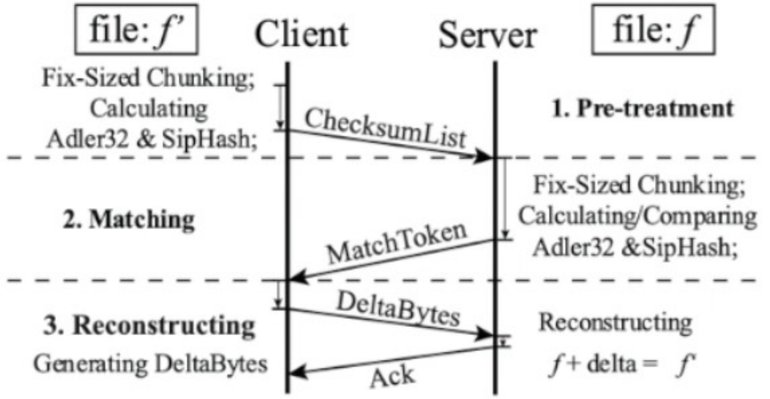
Background and motivation

WebR2sync+:
For low compute and storage devices.

But still need to compare chunks byte by byte!



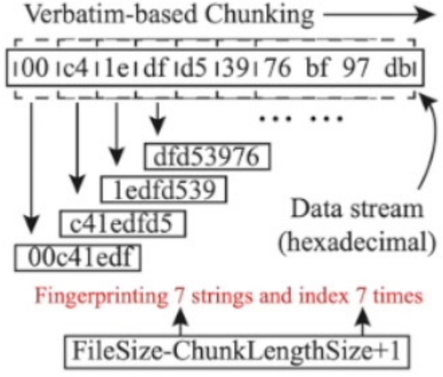
rsync

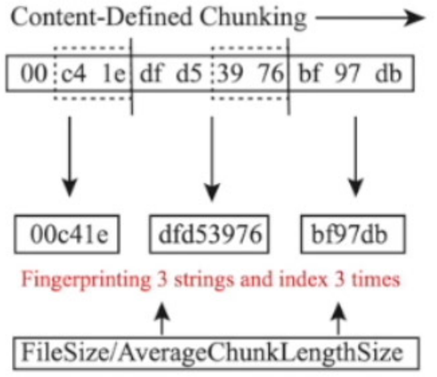


WebR2sync+

Insight of Dsync:

CDC-based comparing to decrease overhead



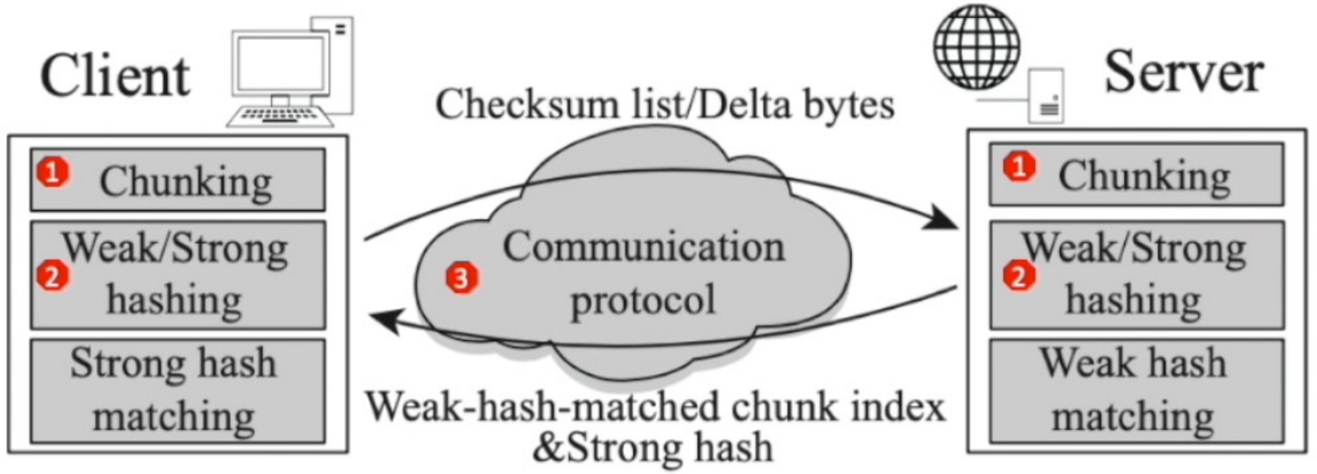


Design and implementation - Overview

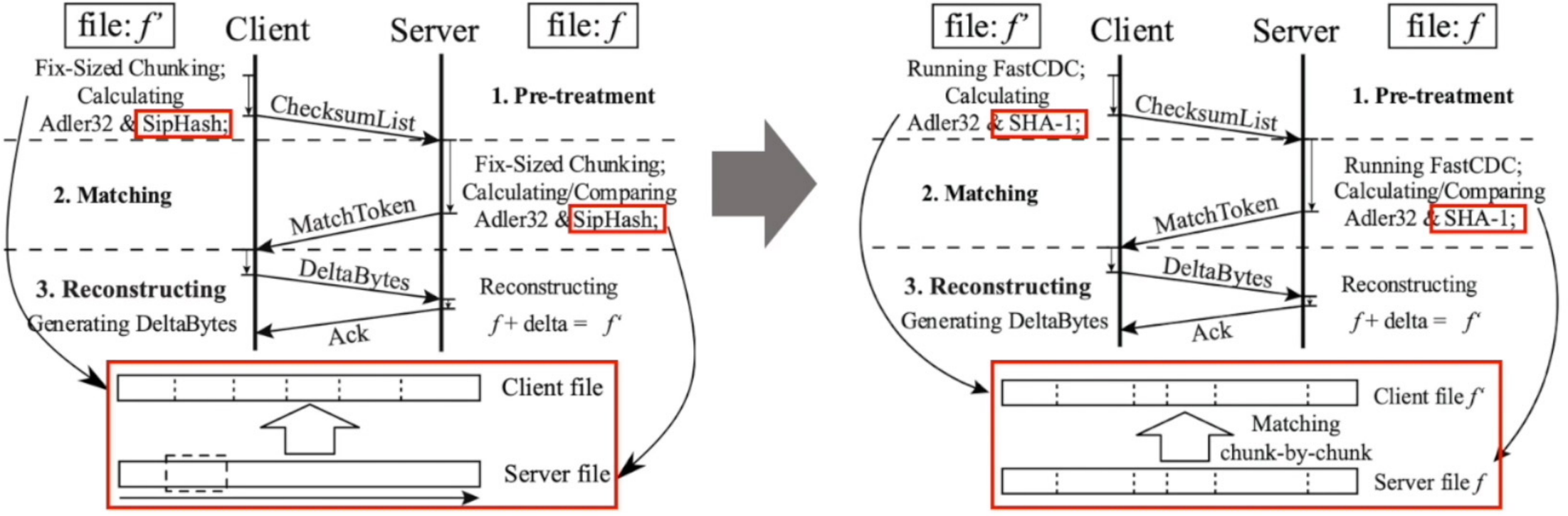
1. Fix-size chunking to content-defined chunking

2. Newer hash function with low overhead

3. Communication protocol
   - Reduce compute overhead
   - Reduce metadata transmission



Design and implementation - Baseline



1. Replace SipHash with SHA-1

2. Replace Fix-Sized Chunking with Content-Defined Chunking (CDC)

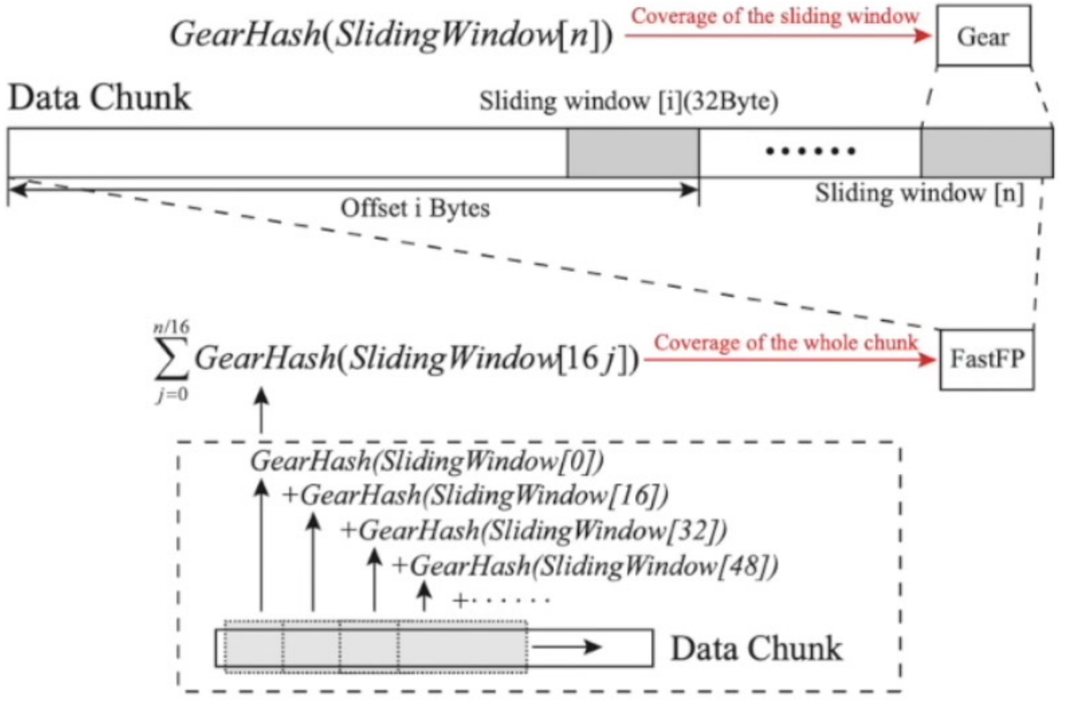
Design and implementation - FastFp

1. Reduce hash compute overhead:

Generate chunk fingerprint from rolling hash

2. Lower hash collision:

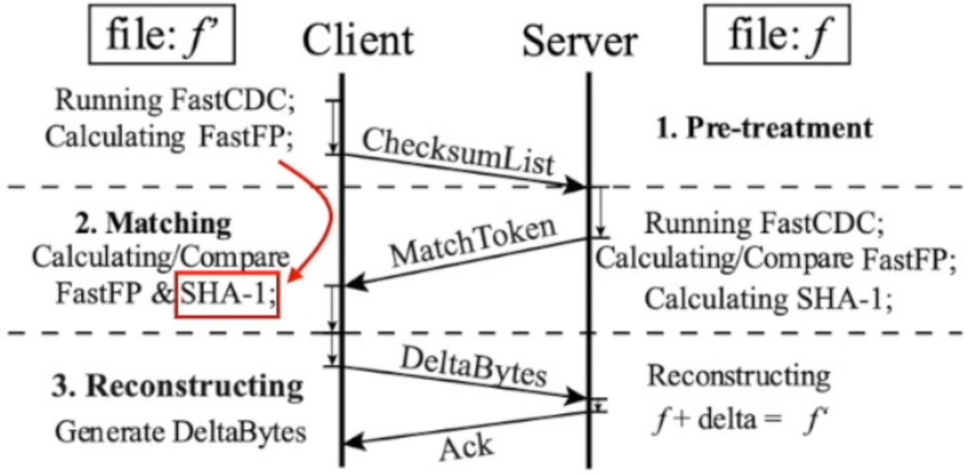
Integrate the influence of sliding windows by summing them up



Design and implementation – Communication protocol

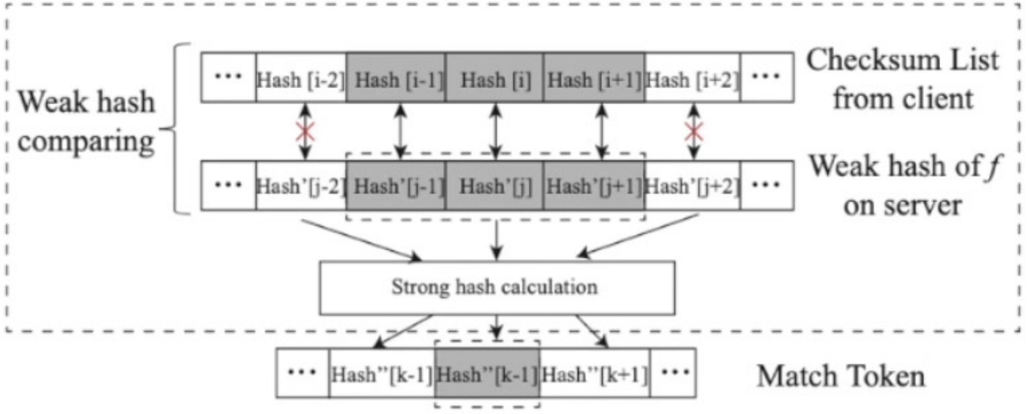
1. Reduce compute overhead:

   Shifting client strong hash calculation



2. Reduce metadata transmission:

   Merging contiguously duplicate chunks



Evaluation

Platform:

Server
NodeJS v12.8@Quard-core virtual machine(16GB memory) ,100Mbps network

PC client
Chrome v76.0(Windows)@Intel i7-7700 CPU, 16GB memory PC

Mobile phone client
Chrome v74.0(Android)@6GB RAM, Huawei Honor V10

Evaluation

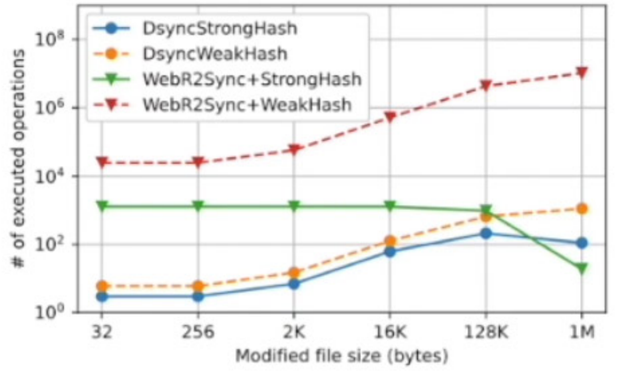
Compared with Adler32, FastFp:

Equivalent collision ratio,
But higher throughput

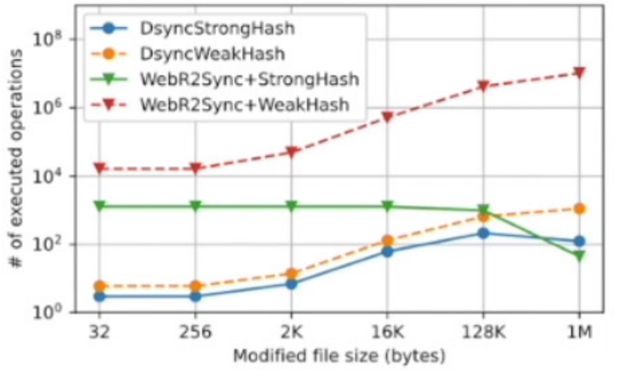
THROUGHPUT AND COLLISION RATIO COMPARISON. HERE FASTFP IS EVALUATED WITH DIFFERENT CONFIGURATIONS: "⊕" IS FOR "XOR", "+" IS FOR "PLUS", WINDOW SLIDING DISTANCE IS OF 8B, 16B, 32B.

Algorithms	Thpt (MB/s)	Hash collision ratio under different random file size		
		100GB	10GB	1GB
Adler32	295.6	2.3×10^{-10}	2.3×10^{-10}	2.4×10^{-10}
Gear	527.8	6.8×10^{-7}	6.8×10^{-7}	6.8×10^{-7}
FastFp(⊕ 8B)	460.0	2.2×10^{-10}	2.2×10^{-10}	3.1×10^{-10}
FastFp(⊕16B)	459.2	2.3×10^{-10}	2.3×10^{-10}	1.5×10^{-10}
FastFp(⊕32B)	460.4	2.2×10^{-10}	2.4×10^{-10}	3.1×10^{-10}
FastFp(+ 8B)	396.9	2.3×10^{-10}	2.4×10^{-10}	2.8×10^{-10}
FastFp(+16B)	397.5	2.3×10^{-10}	2.2×10^{-10}	1.8×10^{-10}
FastFp(+32B)	409.6	2.3×10^{-10}	2.3×10^{-10}	4.0×10^{-10}

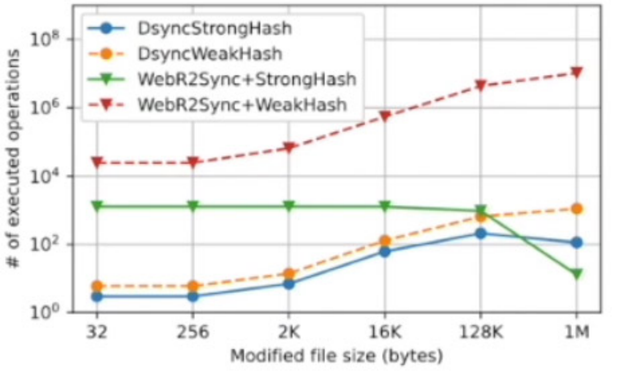
Evaluation



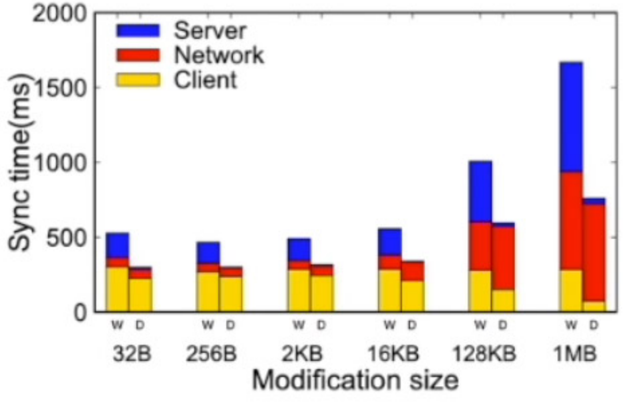
(a) Cut.



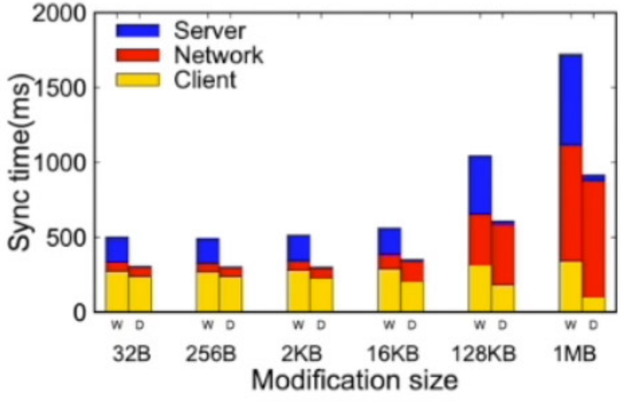
(b) Insert.



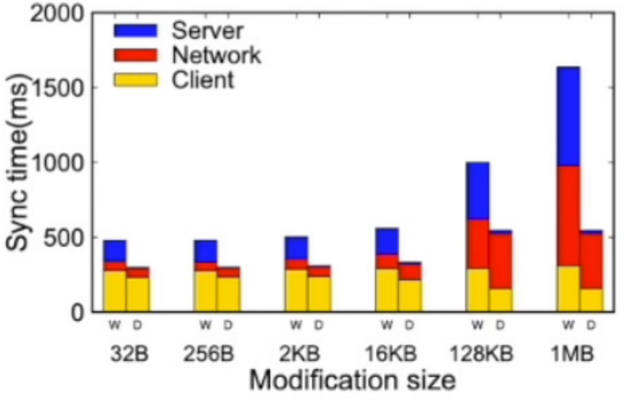
(c) Inverse.



(a) Cut.



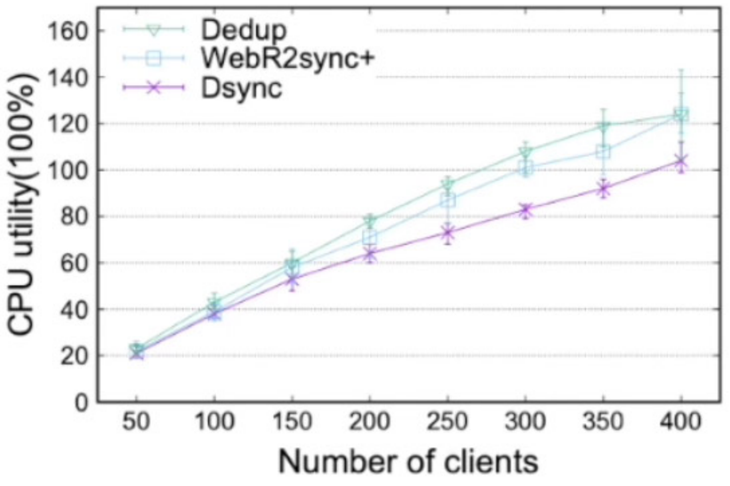
(b) Insert.



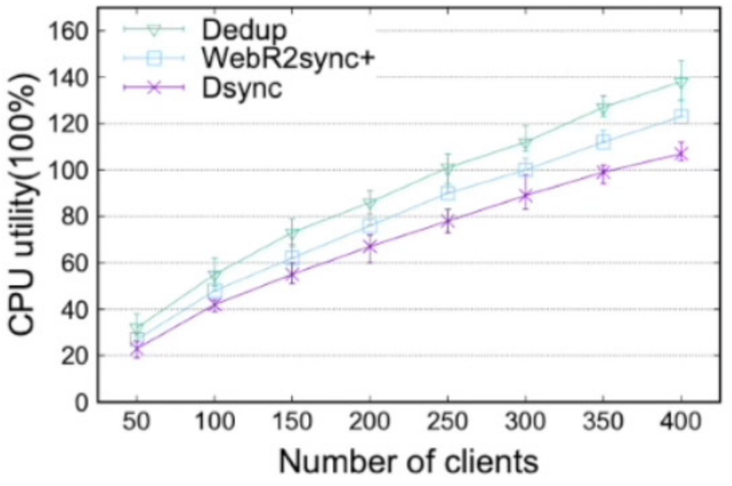
(c) Inverse.

Lower overhead and runs faster than WebR2sync+

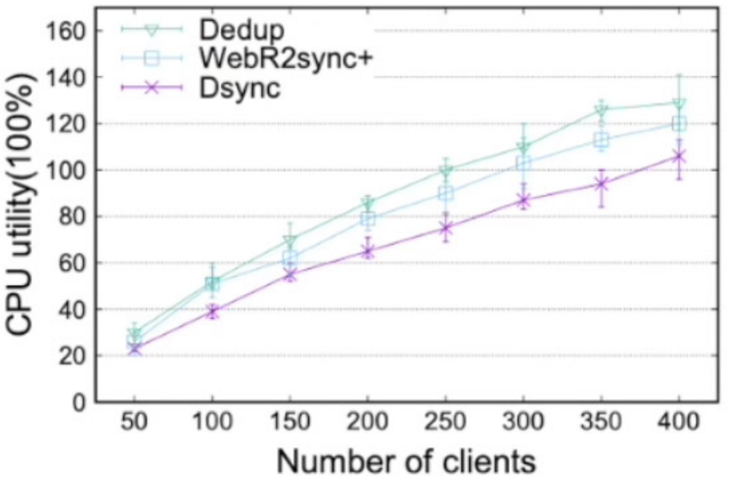
Evaluation



(a) Cut.



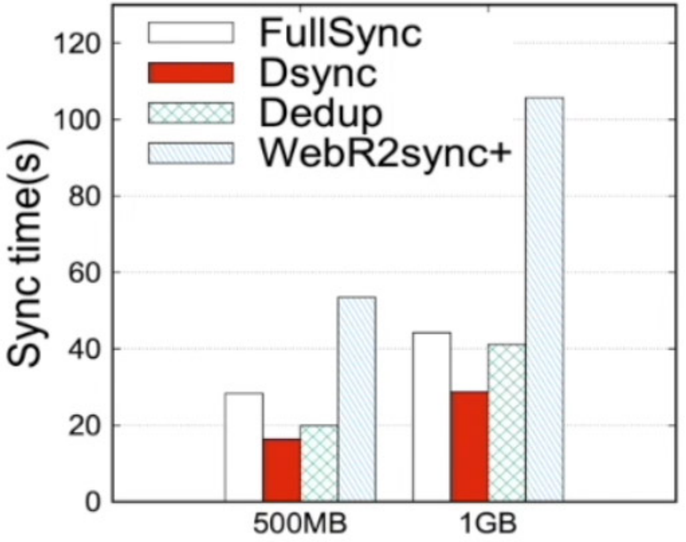
(b) Insert.



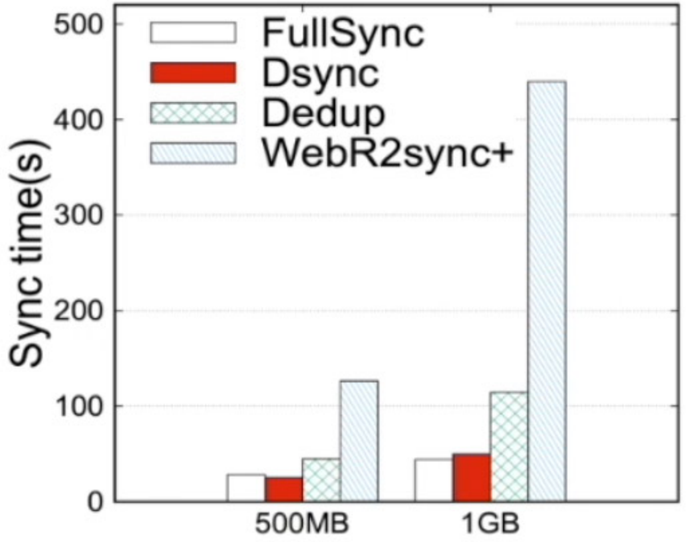
(c) Inverse.

Better scalability than LBFS-based (Dedup) and rsync-based approach (webR2sync+)

Evaluation – Under high bandwidth network



(a) Modification ratio of 1%.



(b) Modification ratio of 30%.

Dsync runs no worse than full sync in any condition!

Conclusion

- Develop a fast, weak hash named FastFp
- Redesign delta sync protocol by exploiting deduplication locality, and hash properties
- 2X~8X faster and 30%~50% higher concurrency than either state-of-the-art rsync-based or deduplication based approach

Thanks for watching!