Dsync: a Lightweight Delta Synchronization Approachfor Cloud Storage Services

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

Speaker：Liang, Jiacheng

Background

rsync:

- Client sends request (including name of local file f') to server.And received Checksum List
- Slides a fixed-size window on the file f', further byte by byte until a matched chunk is found
- If weak-hash matched chunk is found, its strong hash MD5 will be calculated and checked
- The client generates Delta Bytes and sends to server

➢Problem:

- The byte-by-byte sliding window takes a long time
- It needs client to calculate the hash

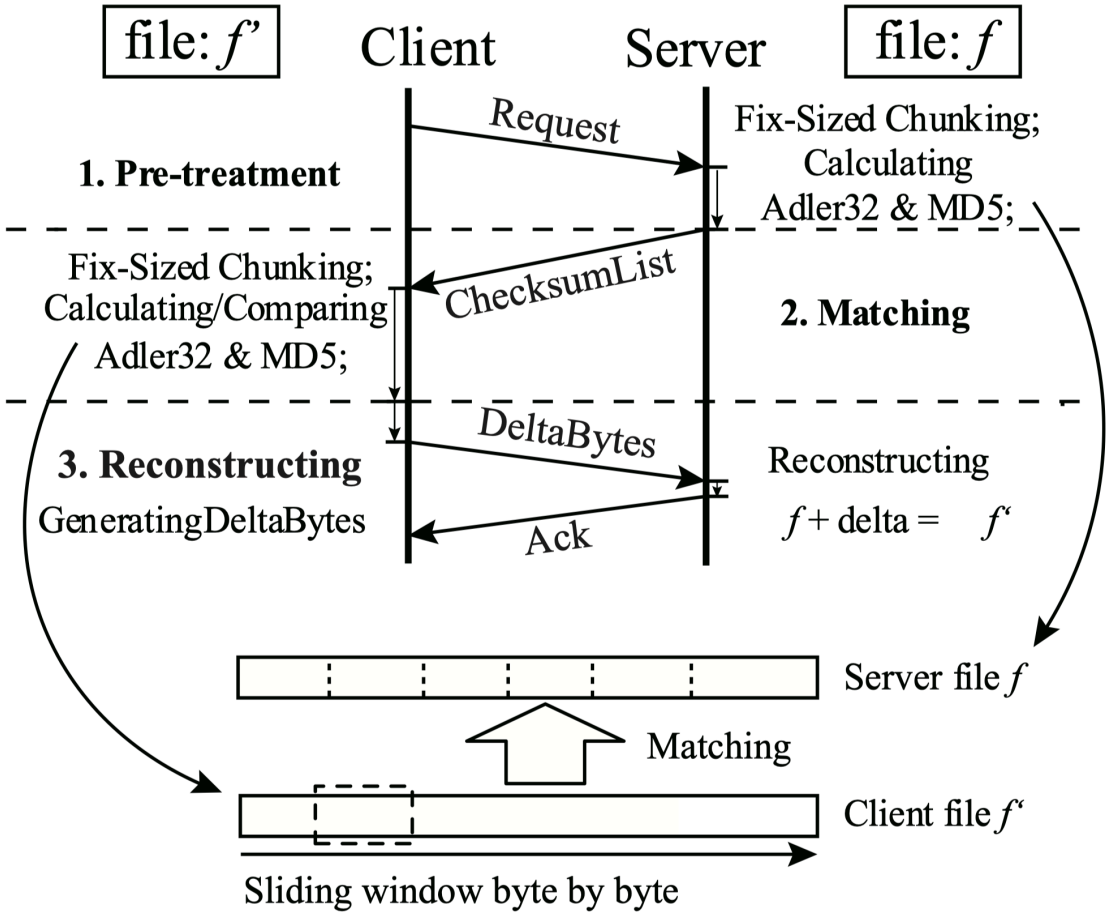


Fig. 1. Workflow chart of `rsync`.

Background

WebR2sync+:

- Client generates the chunksumlist,and replace MD5 as SipHash(faster)
- the chunk–matching process is moved from the client to the server

➢Problem:

- The time–consuming byte by byte chunk matching remains in WebR2sync+

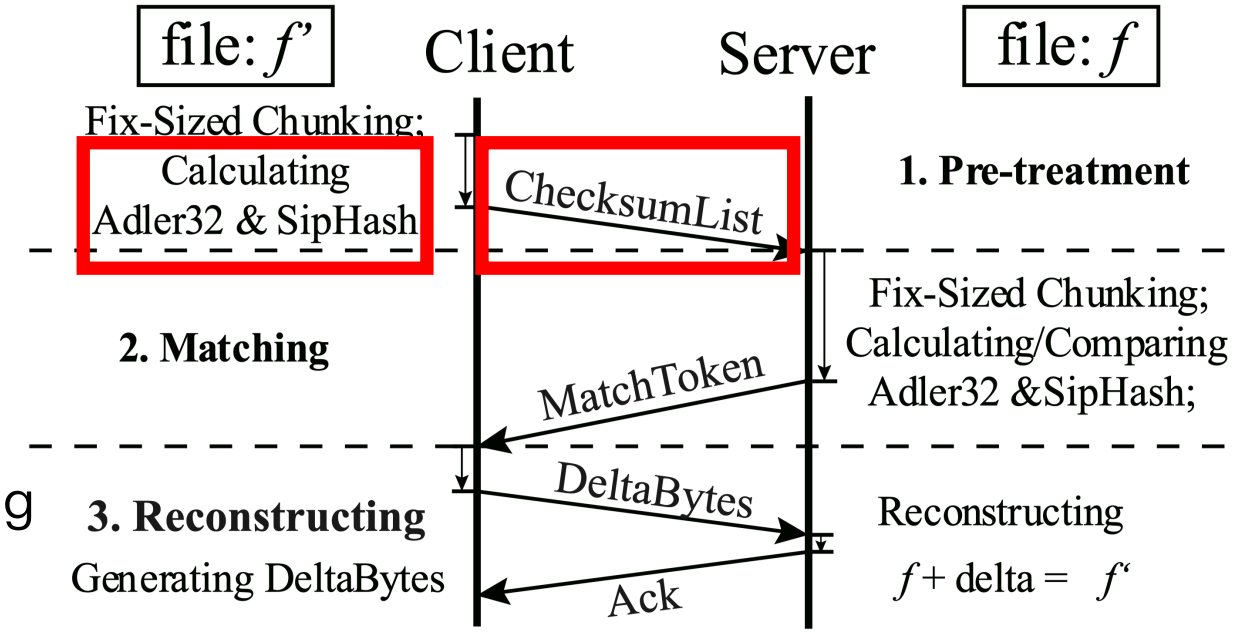


Fig. 2. Workflow chart of WebR2sync+.

Background

CDC:

- CDC uses a sliding–window technique on the content of files and computes a hash value of the window.
- A chunk cut–point is declared if the hash value satisfies some pre–defined condition.

➢Problem:

- CDC introduces additional compute overhead for delta synchronization
- Fail to eliminate redundancy among similar but non–duplicate chunks

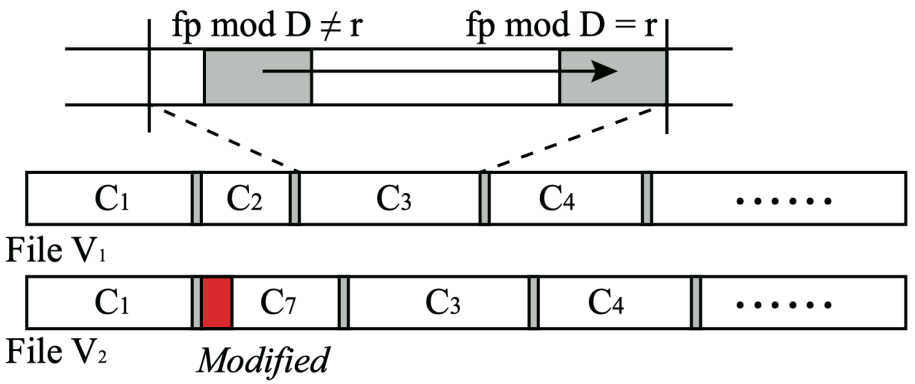


Fig. 3. The CDC technique for the chunk-level data deduplication. A chunk cut-point is declared if the hash value "fp" of the sliding window satisfies a pre-defined condition.

FASTCDC（ATC16）：

- the client splits the file into chunks via FastCDC
- replace Siphash to SHA–1(more reliable)
- A chunk cut–point is declared if the hash value satisfies some pre–defined condition.

➢Problem:

- Still have additional compute overhead from CDC
- low redundancy detection ratio due to the coarse–grained chunkmatching

➢Solve:

- Utilize the hash values generated by Content–Defined Chunking in FastCDC, as the weak hash
- Redesign the communication protocol to minimize the size of metadata for the Match Token

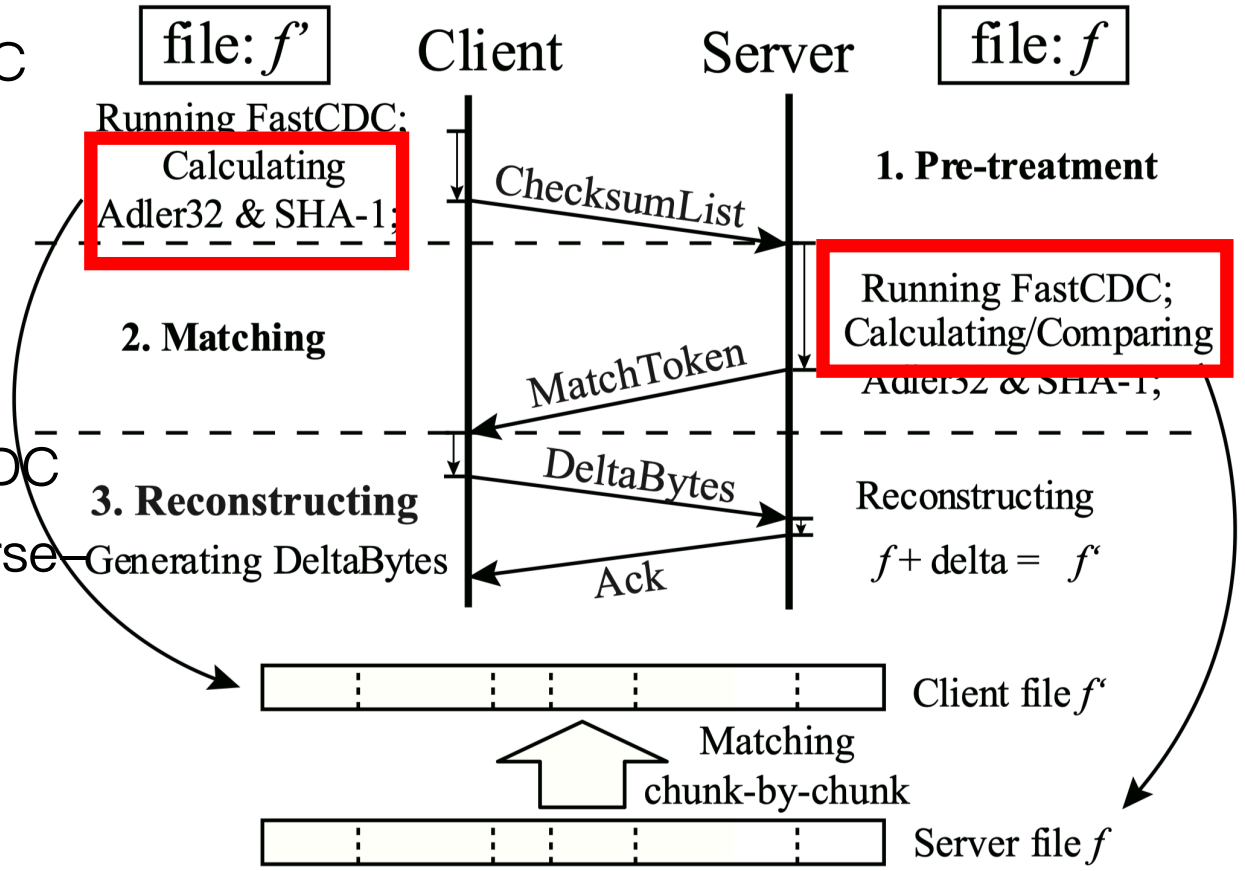


Fig. 6. Preliminary FastCDC-based Dsync prototype.

Solution1: FASTFP

- FastCDC is very fast, but still incurs extra calculation

➢How to use it?

- CDC is sliding on the data by using the rolling hash algorithm
- it is feasible to quickly obtain the weak hash for Dsync according to the rolling hashes during CDC

➢FASTFP:

- the original Gear hash is only relevant to the content that is 32–Byte
- FastFP is related to all the contents of the chunks by using an extra operation '+' to combine many Gear hashes into one hash value

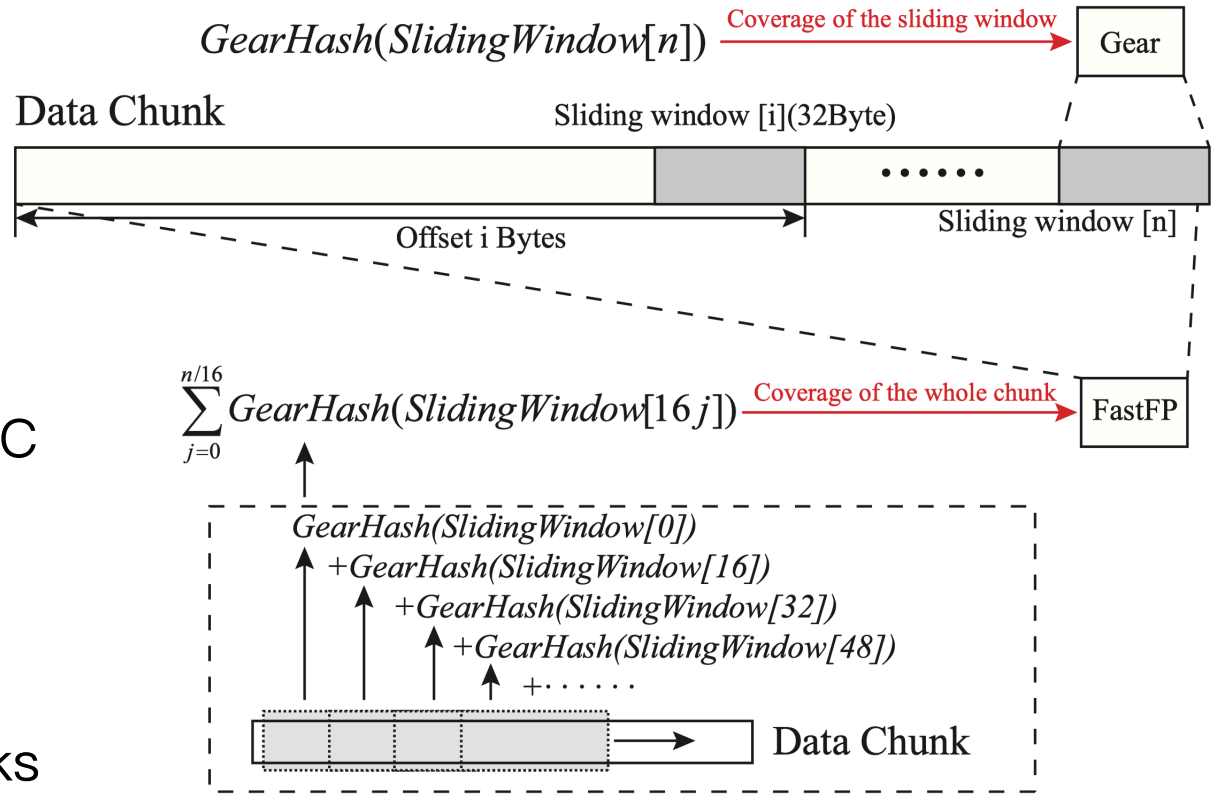


Fig. 7. A schematic diagram of FastFp using Gear hash.

Solution2:Server Communication Protocol

➢Not calculate strong hash at first

- Dsync splits the client file f' into several chunks via FastCDC with their weak hash (FastFp) values generated.
- Sent Checksum List (not include strong hash SHA1 now) and sent to the server.
- If weak hash is matched,then sent the SHA–1 to the client for compare

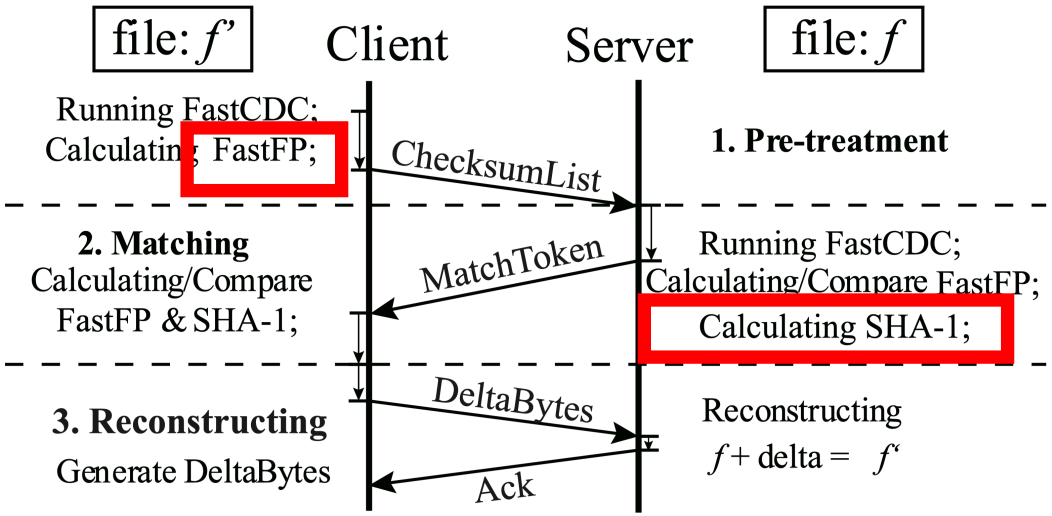


Fig. 8. The redesigned protocol of Dsync to minimize the strong hash calculation.

➢Big Chunk

- Generated the continual same weakhash value,only one strong value will be compared in the client.

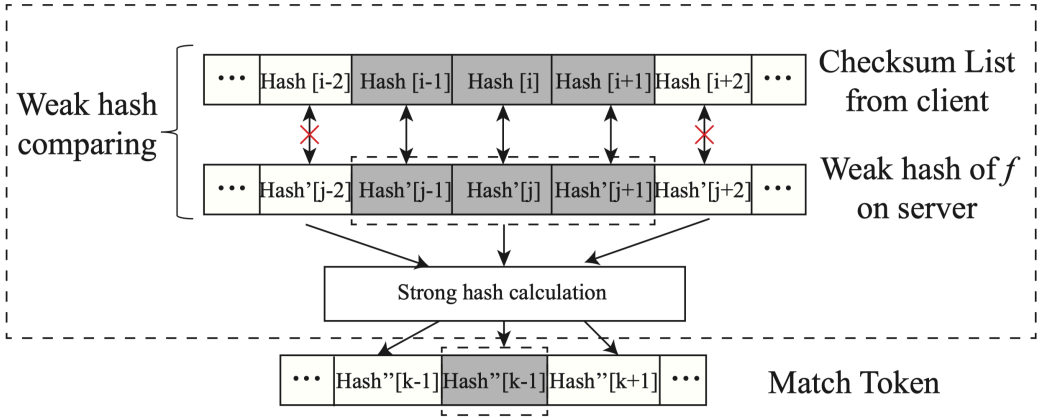


Fig. 9. Merging several consecutive weak-hash-matched chunks into one chunk to reduce metadata overhead.

Experimental setup

- A quadcore Intel i7–7700 CPU, 16GB memory PC with Windows 10 operating system&Chrome
- A 6GB RAM, 64GB ROM mobile phone with Huawei honor V10& Chrome

- A quadcore virtual machine @3.2GHz (installed Ubuntu Server 16.04 with 16GB memory and 128GB disk).

➢ Benchmark Datasets.:

Silesia is a widely acknowledged dataset for data compression covering typical data types that are commonly used, including text, executables, pictures, htmls and et cetera.

Weak Hash FastFp

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}

- The hash speeds of FastFp reaches almost 400–460MB/s, about 50% higher than Adler32.
- The bigger the dataset is, the higher the probability of hash collision will be

Redesigned Protocol ——Server

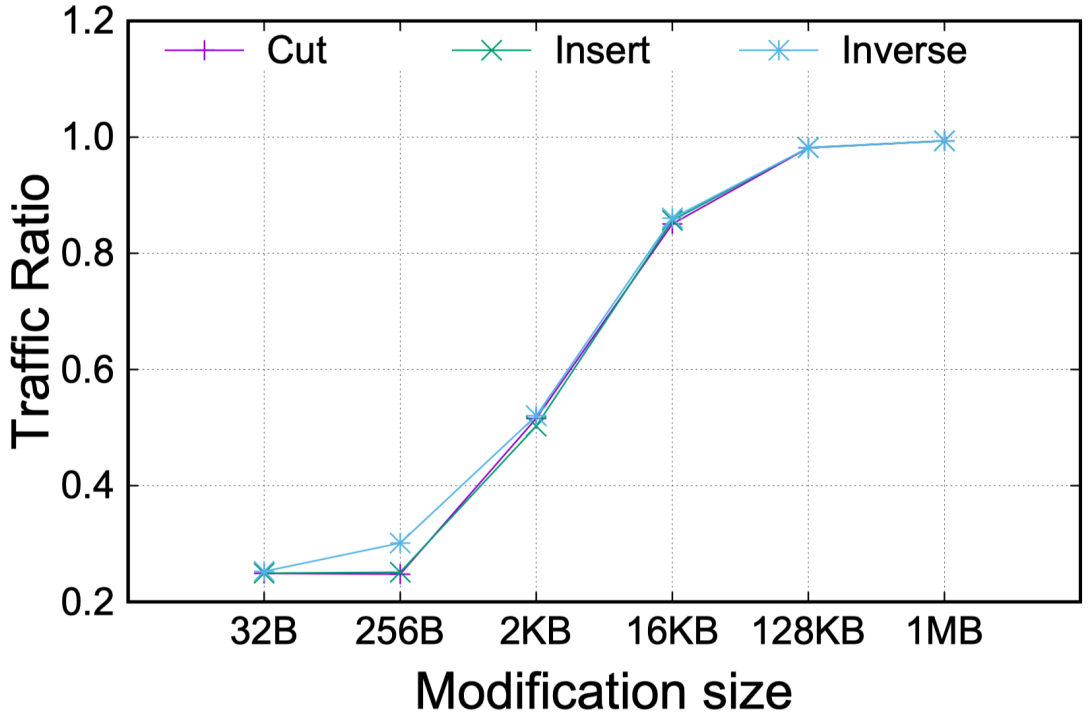
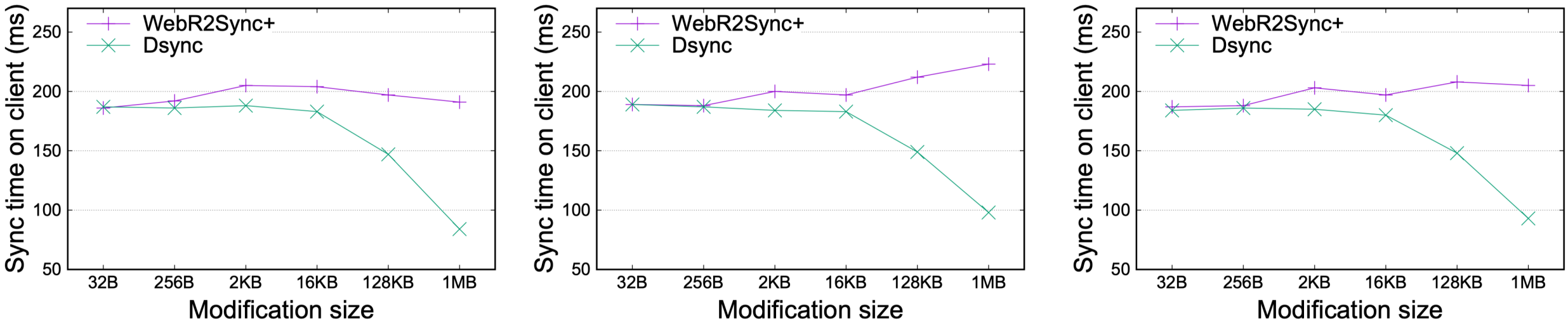


Fig. 11. Total traffic ratio as a function of the modification size, after/before applying the Dsync protocol that merges consecutive weak-hash-matched chunks.

- When there are fewer modifications (e.g., only inserting 32B), the total traffic is reduced by 70% ~ 80% in Dsync

Redesigned Protocol ——Client

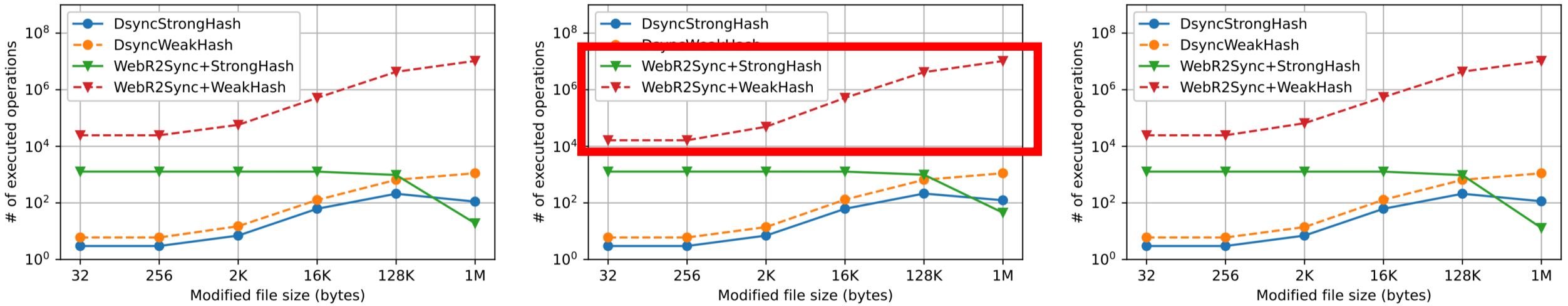


(a) Cut. (b) Insert. (c) Inverse.

Fig. 12. The sync time spent on client as a function of modification size in WebR2sync+ and Dsync.

- Initially, the two are similar, but as the modification size increases, Dsync's time gradually decreases due to the reduction in strong hash calculations

Putting It All Together



(a) Cut. (b) Insert. (c) Inverse.

Fig. 13. Numbers of hash-comparing operations of Dsync and WebR2Sync, including both strong and weak hashes.

- When file modifications are minor, Dsync compares fewer strong hash fingerprints because of our merging strategy and WebR2Sync+'s ability to find more chunks than Dsync.
- However, when modification size is 1MB, it's hard to find unmodified chunks for WebR2Sync+,

Putting It All Together

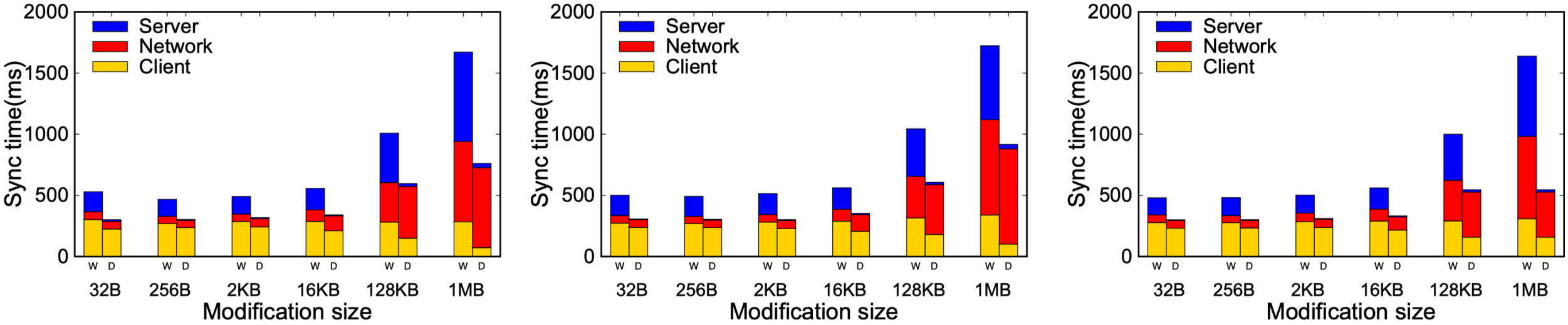


Fig. 14. Sync time breakdown of WebR2sync+ and Dsync as a function of modification sizes.

- Compared with WebR2sync+, the time consumed by both the server and the client has been greatly reduced and the time spent on the network has been reduced as well,

Real Dataset

TABLE III
SYNC PERFORMANCE OF THE THREE APPROACHES ON BOTH WINDOWS/ANDROID CLIENT ON FOUR REAL-WORLD DATASETS. NOTE THAT WINDOWS/ANDROID PLATFORMS HAVE THE SAME SYNC TRAFFIC IN THE LAST THREE COLUMNS SINCE THEY ONLY DIFFER IN THE COMPUTE CAPACITY.

	Sync Time(Seconds) (Windows/Android)			*Sync Traffic(MB) (Windows/Android)*		
Dataset	***Dsync***	***WebR2sync+***	***Dedup***	***Dsync***	***WebR2sync+***	***Dedup***
Pictures	17.49/ 53.45	20.27/ 83.90	20.61/ 59.74	94.3	105.3	104.4
PPT	11.14/ 44.69	20.81/ 84.22	15.22/ 56.39	162.2	162.4	164.0
Mail	27.19/117.40	90.46/271.74	48.67/160.06	635.4	638.0	637.6
GLib	41.98/157.46	75.05/299.74	52.03/186.88	497.8	455.4	532.7

- Both PC and mobile phone clients (about 1.5–3.3× faster).
- The sync traffic volumes of the three approaches are almost the same.

Conclusion

- FASTCDC: This paper propose to use FASTCDC(ATC16),and replace SipHash with SHA1 in WebR2sync+
- FASTFP: This paper uses the weak hash according to the rolling hashes during CDC to generate weak hash.
- Client/Server Communication Protocol

Only FastFP(without the strong hash) will be sent to the server at first. If the weak hash is a match, the client will calculate the strong hash.

- BigChunk: continuous match chunk will be merged as a big chunk and just compare the big chunk's strong hash.

Comments

➢ Strengthens

• Parts of fastCDC is used to generate FASTFP, which greatly saves the cost of hashing calculation.

• Big Chunk saves a lot of strong hash computation overhead

➢ Weaknesses

• Both deduplication and FASTCDC are existing things.

• The main contributions of this paper are piecing them together and proposing the fastFP hash algorithm.

➢ Comments

• The overall design of the article is not complicated, the writing is clear and fluent, and the testing part is also relatively perfect

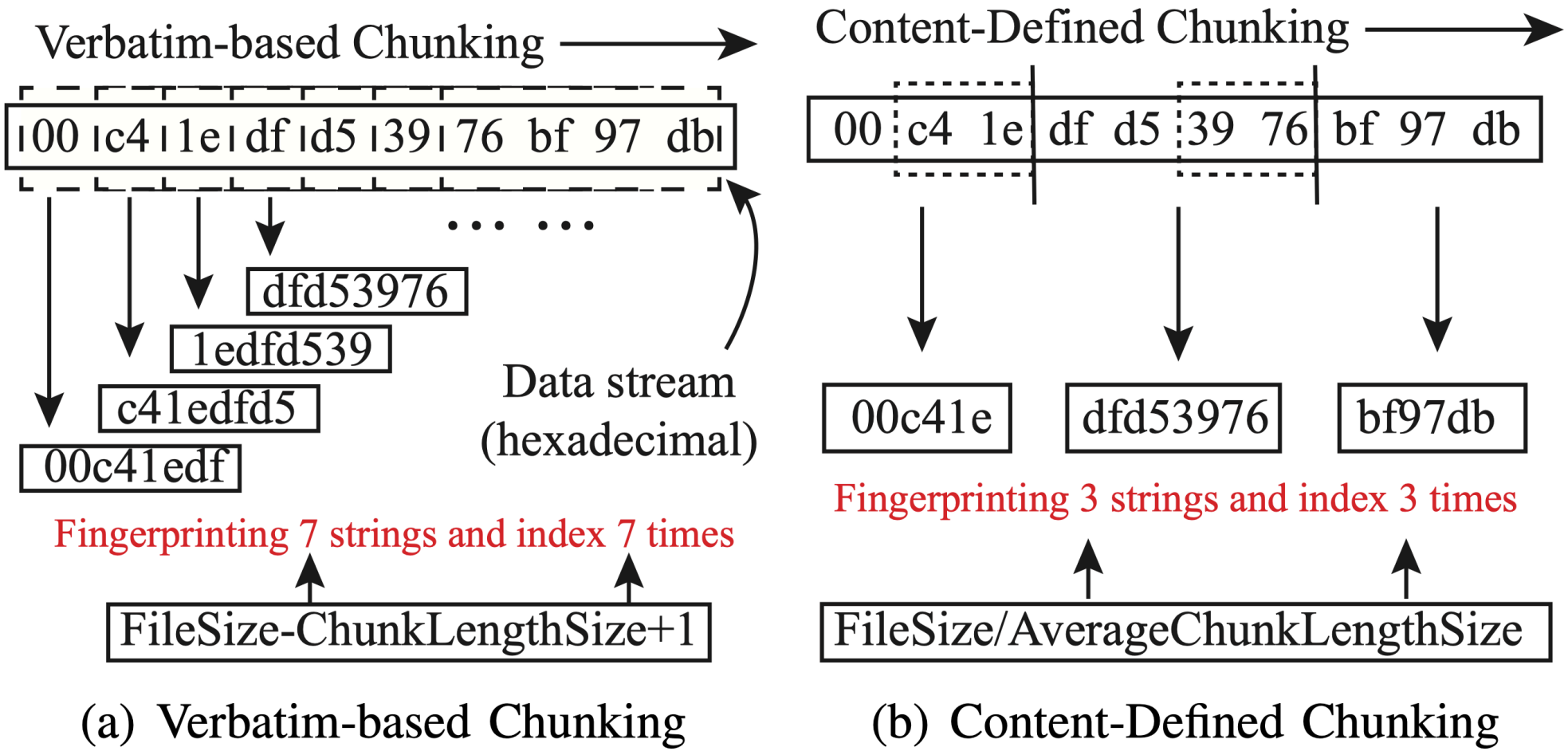
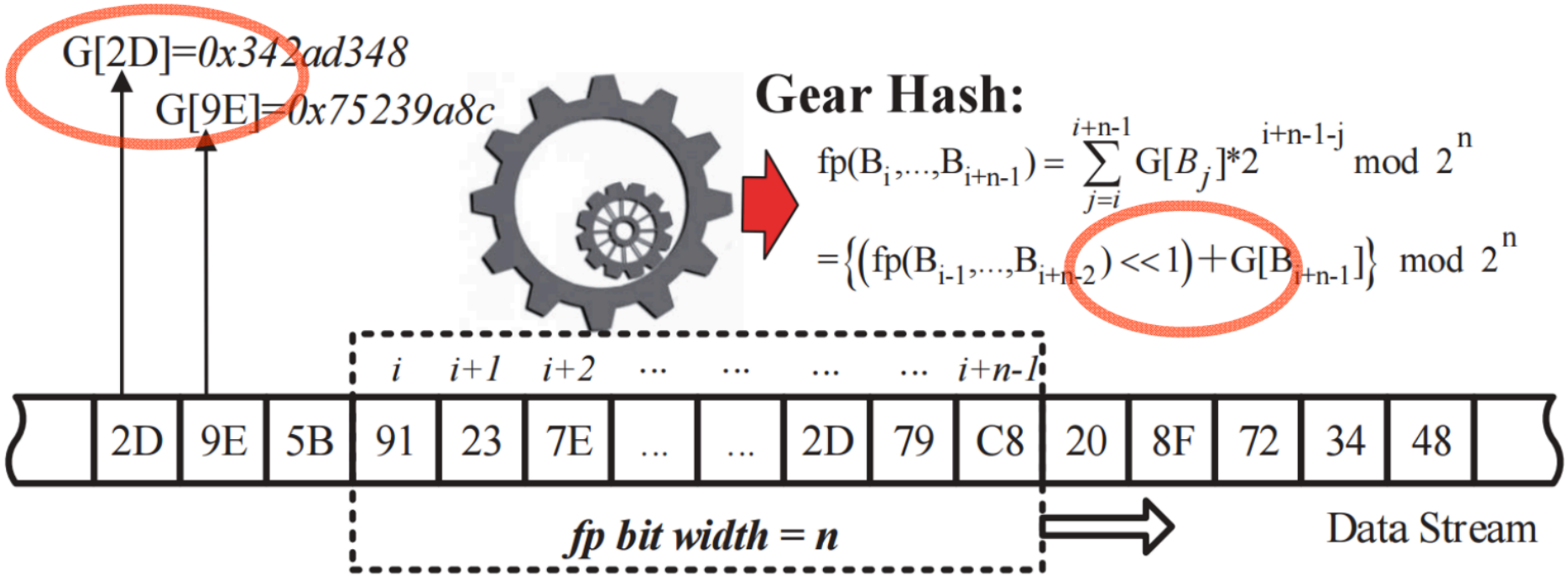


Fig. 4. Comparison of Verbatim-based Chunking and Content-Defined Chunking used for string matching. CDC-based approach executes much fewer fingerprinting and searching operations.

Gear: a very fast rolling hash for CDC



Implementation: ***fp = (fp << 1) + G(b)***

- One array lookup
- One "+" operation
- One left-shift operation

Ddelta: A dedup-inspired fast delta compression approach @ Performance'14