Reduct: Deduplicated Distributed File System

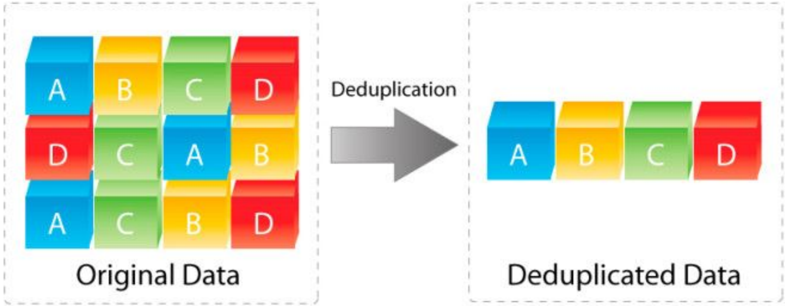
Keonwoo Oh, Sam Rothstein, Ford Johnstone

Background

- Data is Growing:
  - People are generating 2.5 quintillion bytes of data each day
  - Nearly 90% of all data has been created in the last two years
- Real World Examples:
  - Up to 80% of some organizations’ data is duplicated across the corporate network
  - Reducing deduplicated data can save money in terms of storage costs and backup speed
- Files stored in distributed file systems are typically large:
  - Ideal if we could save data storage, but maintain data integrity
- Two main strategies
  - Data compression vs. deduplication
    - Data compression uses an algorithm to reduce the bits required to represent the stored data
    - Data deduplication: next slide

Data Deduplication

- Technique for eliminating redundant/duplicate data in a data set
- File Level vs. Subfile Level
- Fixed Length Block vs. Variable Length Block
- Inline vs. Post Processing
- Fingerprint Indexing vs. Other methods



Fingerprint Indexing

- Deduplication requires means to detect whether a file is a duplicate
- Usually performed by computing & maintaining distinct fingerprint index for each file.
- Uses crytographic hash functions to compute hash values of files
- If two files have the identical hash values, they are most likely duplicates $\rightarrow$ discards one copy and set mapping from both file names to one copy that is stored. Otherwise, they are not duplicates and both files are stored.
- Could be performed at subfile level by using regular-sized data blocks, instead of whole files.

Deduplication in Distributed File Systems

- One system was built on top of HDFS (Hadoop Distributed File System) and HBase, both of which are part of Hadoop.
  - File is broken down to chunks & each chunk is assigned a unique fingerprint.
  - Fingerprint index stored HBase.
  - Each chunk stored as a file in HDFS.
- Droplet internally incorporates deduplication.
  - Consist of fingerprinting servers and storage servers.
  - Client sends data to fingerprinting servers, which break down to blocks and compute hash values.
  - Daemon in fingerprinting servers periodically query storage servers to check if the blocks are already stored → if not stored, send the blocks for storage.
- Other methods
  - eg) Using bloom filter for storing fingerprint index.

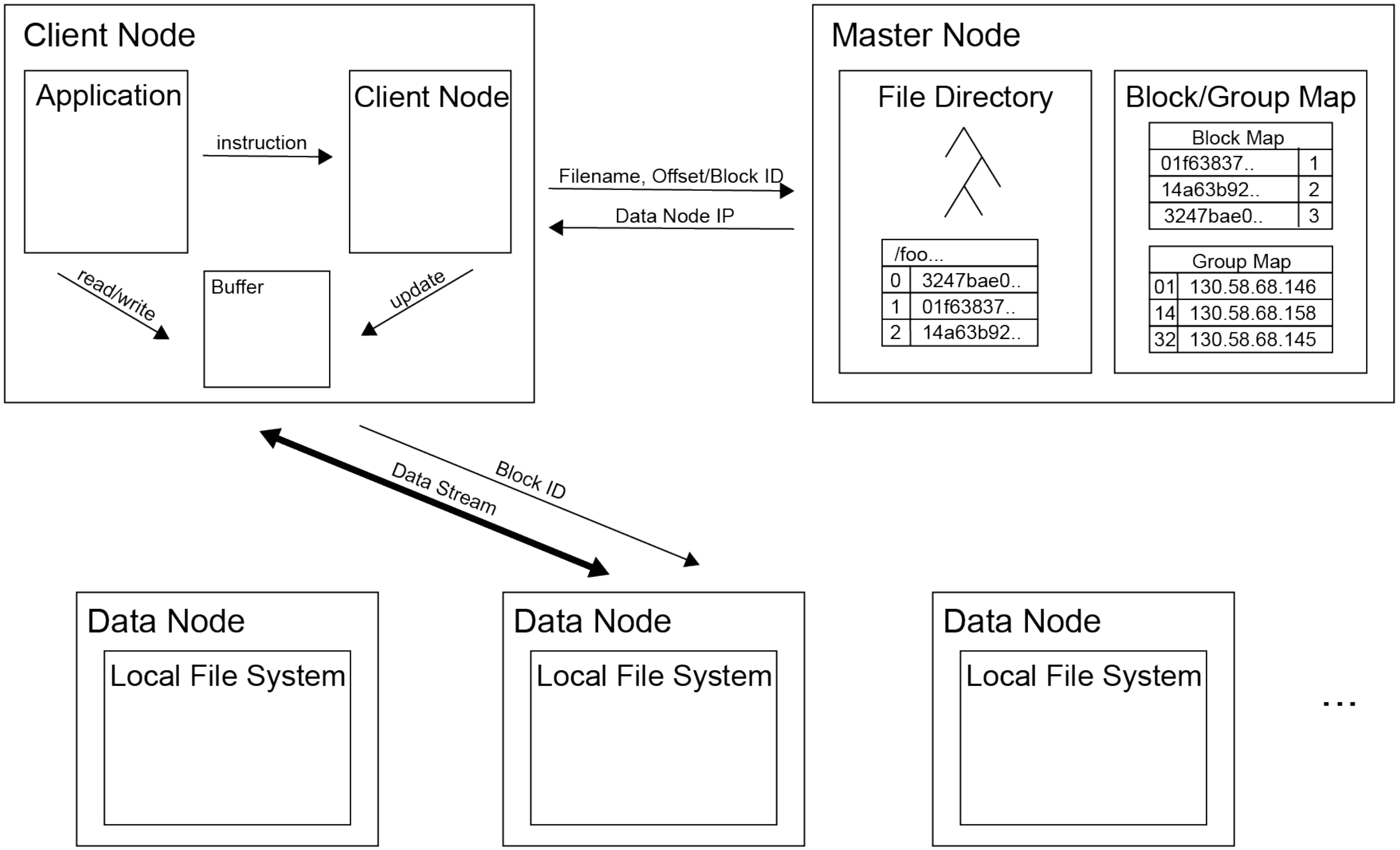
However..

- Two separate levels of interface and metadata
    - Deduplication is built on top of the file system → extra metadata
    - May compromise the performance of key operations (read, write, etc.)
    - eg) Files are broken down to chunks, which are stored as files in HDFS and further broken down to blocks

- Hash collision → Highly unlikely, but what if?
    - MD5 and SHA-1 have already be deprecated
    - Is SHA-2 safe?
    - Could lower probability by using stronger hash functions, but may compromise performance and increase metadata per byte of data

Reduct: Overall Architecture

- Distributed File System with Block Level Deduplication.
- Write once, read many. No concurrent writes.
- Follows master/worker model, similar to the Google File System.
  - The master node maintains all file system metadata in memory.
  - Each file is broken down to regular size blocks, which are stored in data nodes.
  - Client modifies the metadata and retrieves block location through the master node if needed.
- Each block is assigned a unique block ID, which is a hexadecimal string.
- No fault tolerance, authentication protocol, security measures, random reads, writes, etc. → simplicity due to time constraint.



Client Node
Application
instruction
Client Node
read/write
Buffer
update
Filename, Offset/Block ID
Data Node IP
Master Node
File Directory
/foo...
0 3247bae0..
1 01f63837..
2 14a63b92..
Block/Group Map
Block Map
01f63837.. 1
14a63b92.. 2
3247bae0.. 3
Group Map
01 130.58.68.146
14 130.58.68.158
32 130.58.68.145
Data Stream
Block ID
Data Node
Local File System
Data Node
Local File System
Data Node
Local File System
...

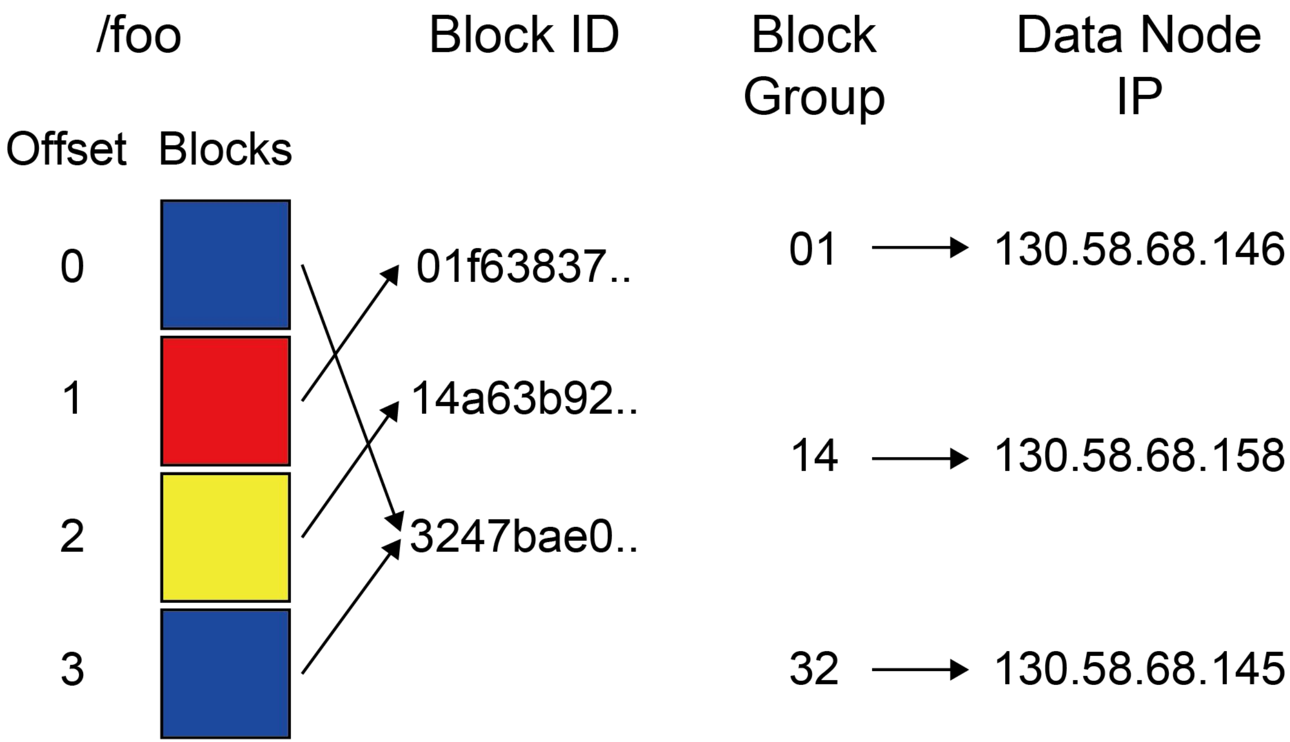
Data Blocks & Deduplication

- Each file is broken down to regular-sized blocks, which are stored in data nodes.
- Each block is assigned a unique block ID and organized into block groups based on the first 2 digits of the ID.
    - eg) A block with block ID, 01f63837.. is assigned to block group, 01.
- Any two blocks in the same block group are stored in the same data node.
- Block ID is generated by concatenating hash value of the data block with an offset, by default, 0.
    - If two blocks are duplicates, they have the same ID and only one copy is stored.
    - If two blocks are not duplicates, but have the same hash value, they have different offsets, therefore assigned different block ID → no data corruption due to hash collision
        - eg) Block A is assigned block ID of 01f63..0 and Block B is assigned block ID of 01f63…1.

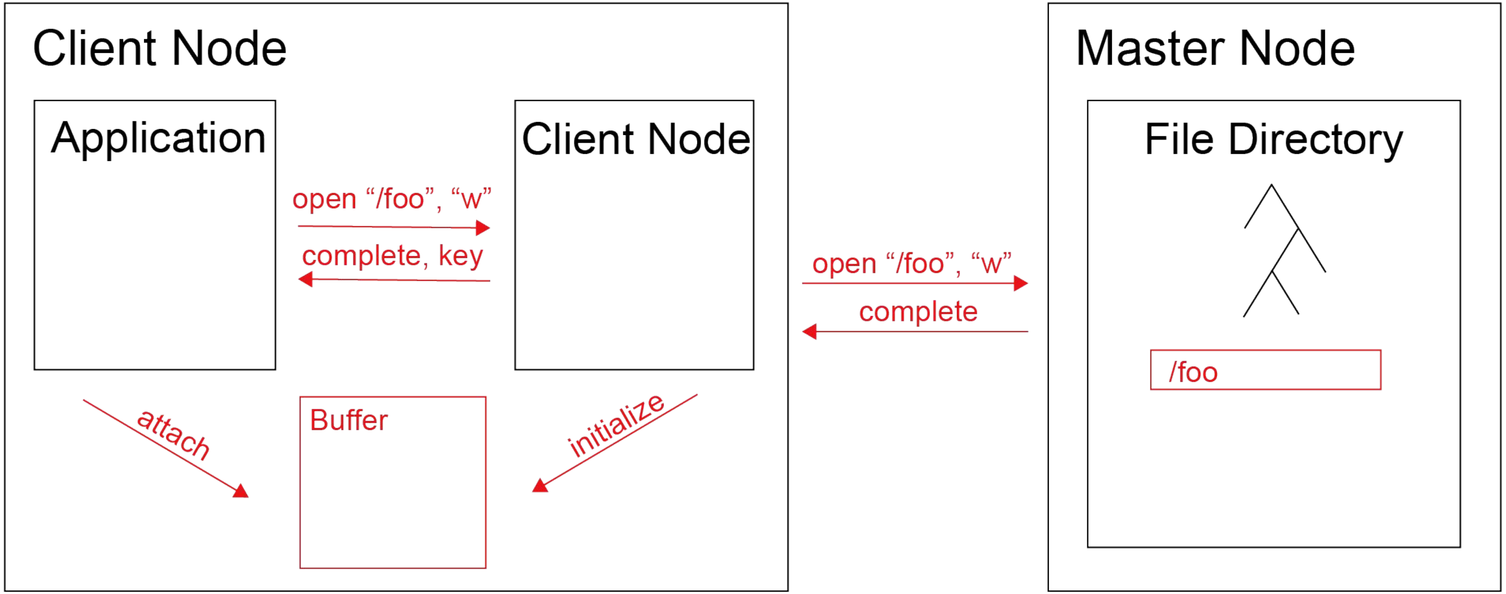
Metadata

- The master node stores all metadata in memory.
- Any modification of metadata requested by a client is atomic.
- Consists of File Directory, BlockMap, and GroupMap
    - File Directory: tree of inodes (directories and files), which provides a mapping from file name and offset to block ID.
    - BlockMap: provides mapping from block ID to the number of duplicates.
    - GroupMap: provides mapping from block group to the data node IP.
- Overall, the metadata provides mapping from file name and offset to block ID and mapping from block ID to its location.
- The metadata serves as the file system metadata as well as fingerprint index for deduplication.

MetaData


/foo
Block ID
Block Group
Data Node IP
Offset
Blocks
0
1
2
3
01f63837..
14a63b92..
3247bae0..
01
130.58.68.146
14
130.58.68.158
32
130.58.68.145

Open



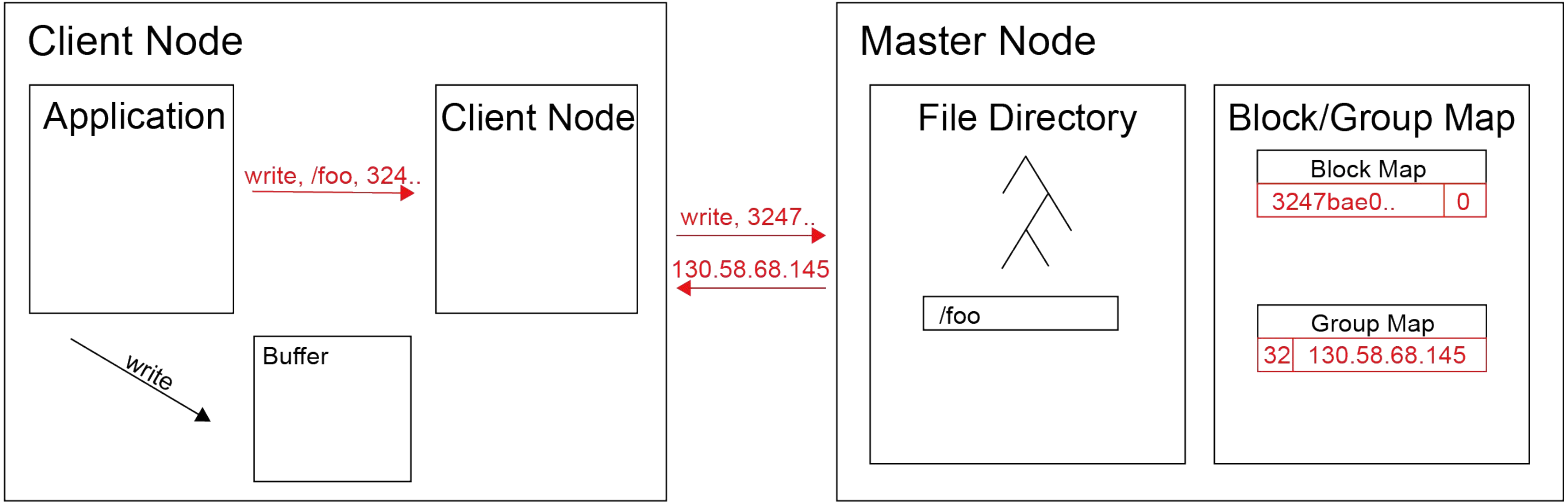
Client Node
Application
open “/foo”, “w”
complete, key
Client Node
attach
Buffer
initialize
open “/foo”, “w”
complete
Master Node
File Directory
/foo

Open

- If the mode is write and the file already exists, the operation fails.

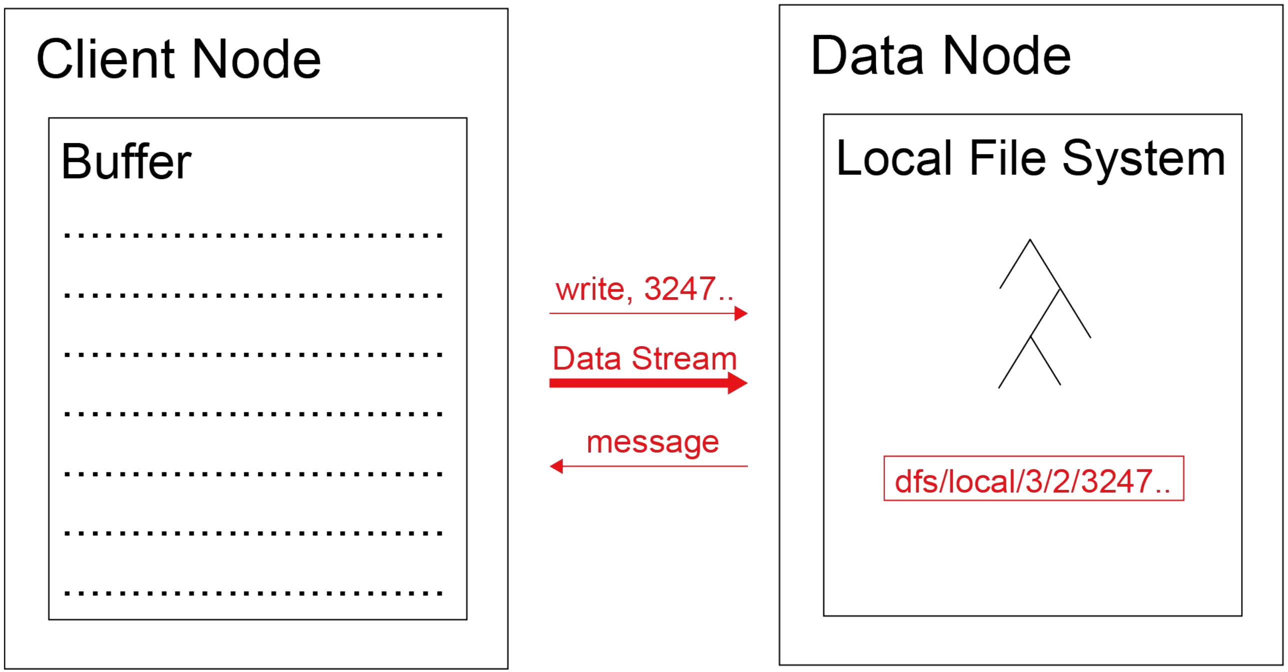
- If the mode is read and the file does not exist, the operation fails.

Write



Client Node
Application
write, /foo, 324..
Client Node
write
Buffer
write, 3247..
130.58.68.145
Master Node
File Directory
/foo
Block/Group Map
Block Map
3247bae0..
0
Group Map
32
130.58.68.145

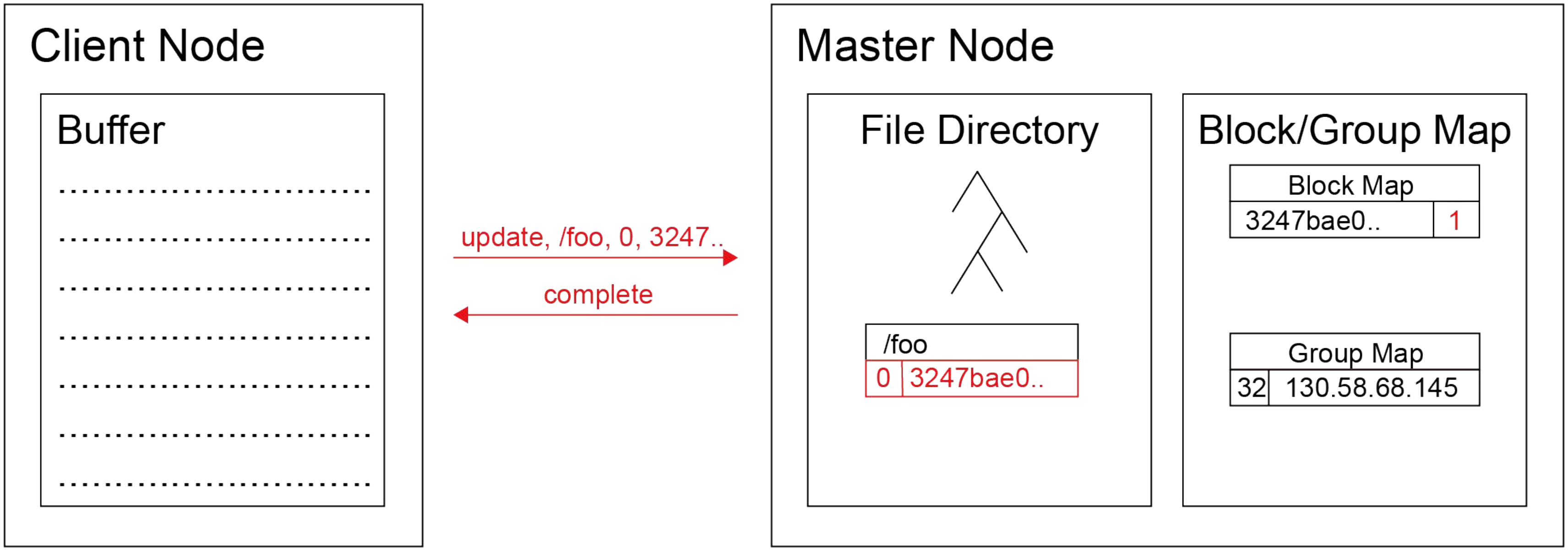
Write


Client Node
Buffer
write, 3247..
Data Stream
message
Data Node
Local File System
dfs/local/3/2/3247..

Write

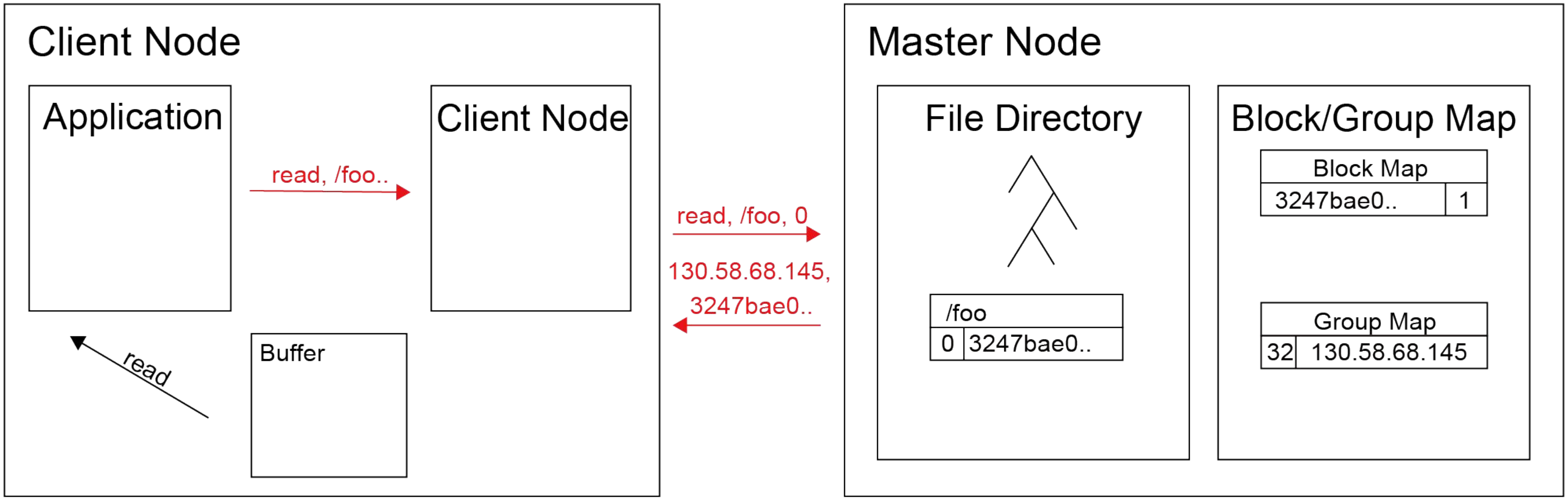
- If a data block with the same block ID does not exist, data node creates a new file in the local file system and writes the data into the disk and sends a complete message.
- If a data block with the same block ID exists, the data node performs byte-by-byte comparison between the blocks with the same block ID:
  - The data node sends a message that indicating whether the received block is a duplicate.
    - If the data block is not a duplicate, the client node restarts the entire write operation with a new block ID of the same hash value concatenated with an incremented offset.
- If there is a duplicate or no hash collision, the client node proceeds to the next step.

Write

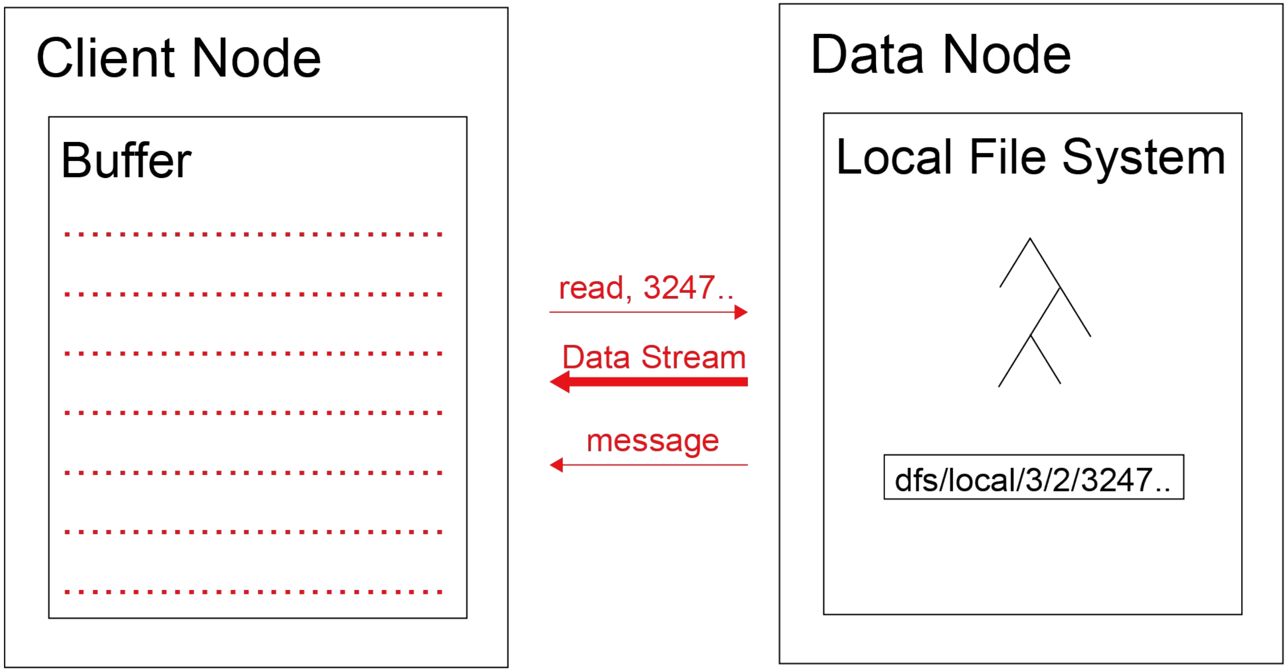


Client Node
Buffer
update, /foo, 0, 3247..
complete
Master Node
File Directory
/foo
0
3247bae0..
Block/Group Map
Block Map
3247bae0..
1
Group Map
32
130.58.68.145

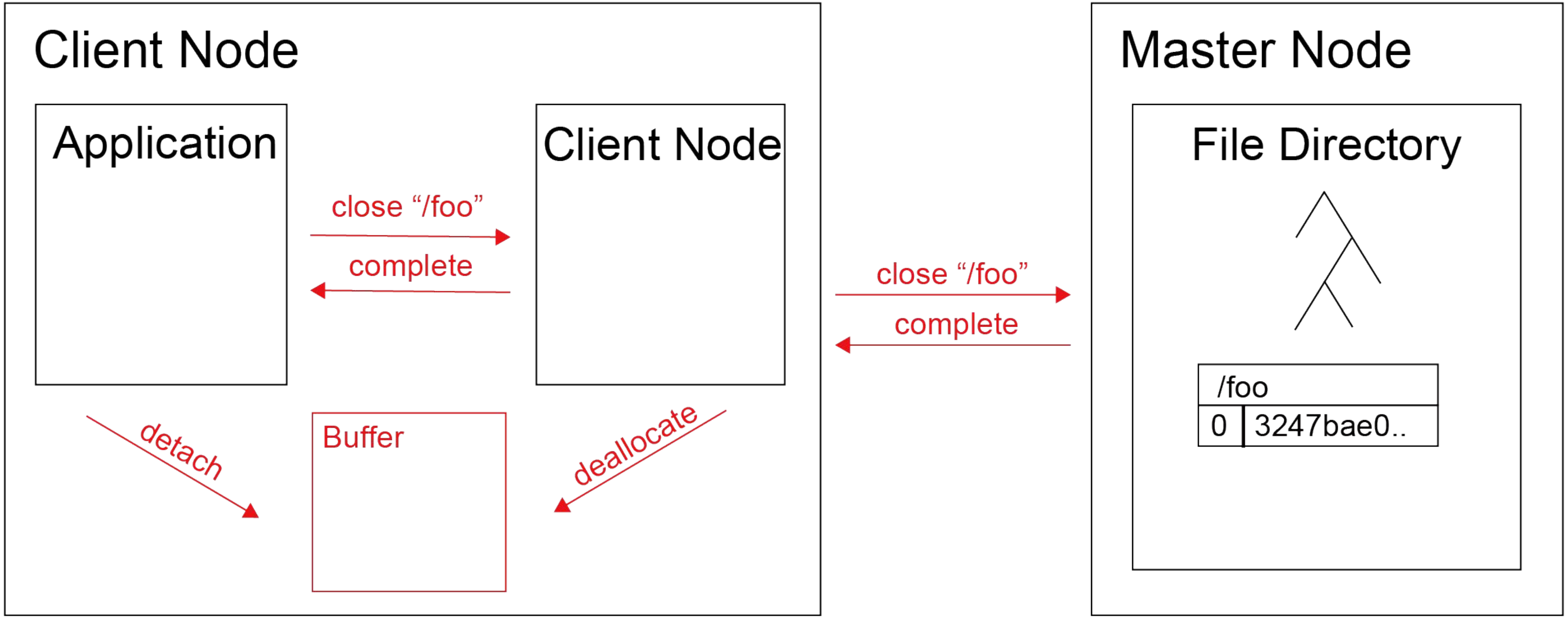
Read


Client Node
Application
read, /foo..
Client Node
read
Buffer
read, /foo, 0
130.58.68.145,
3247bae0..
Master Node
File Directory
/foo
0
3247bae0..
Block/Group Map
Block Map
3247bae0..
1
Group Map
32
130.58.68.145

Read


Client Node
Buffer
read, 3247..
Data Stream
message
Data Node
Local File System
dfs/local/3/2/3247..

Close


Client Node
Application
close "/foo"
complete
Client Node
detach
Buffer
deallocate
close "/foo"
complete
Master Node
File Directory
/foo
0
3247bae0..

Experiments

- Tested deduplication ratio, read, and write speeds.
- Conducted 3 experiments with 1 master node, 4 data nodes, and 4 client nodes.
- In first 2 experiments, each client wrote 256MB of data into its own file.
- In the Experiment 3, each client wrote about 250MB of data into its set of files.
- In Experiment 1, each client wrote an array of one repeated character.
- In Experiment 2, each client wrote an array of randomly generated characters.
- In Experiment 3, each client wrote arrays of characters into a set of files where the 34.57% of the total set were duplicates (file level).

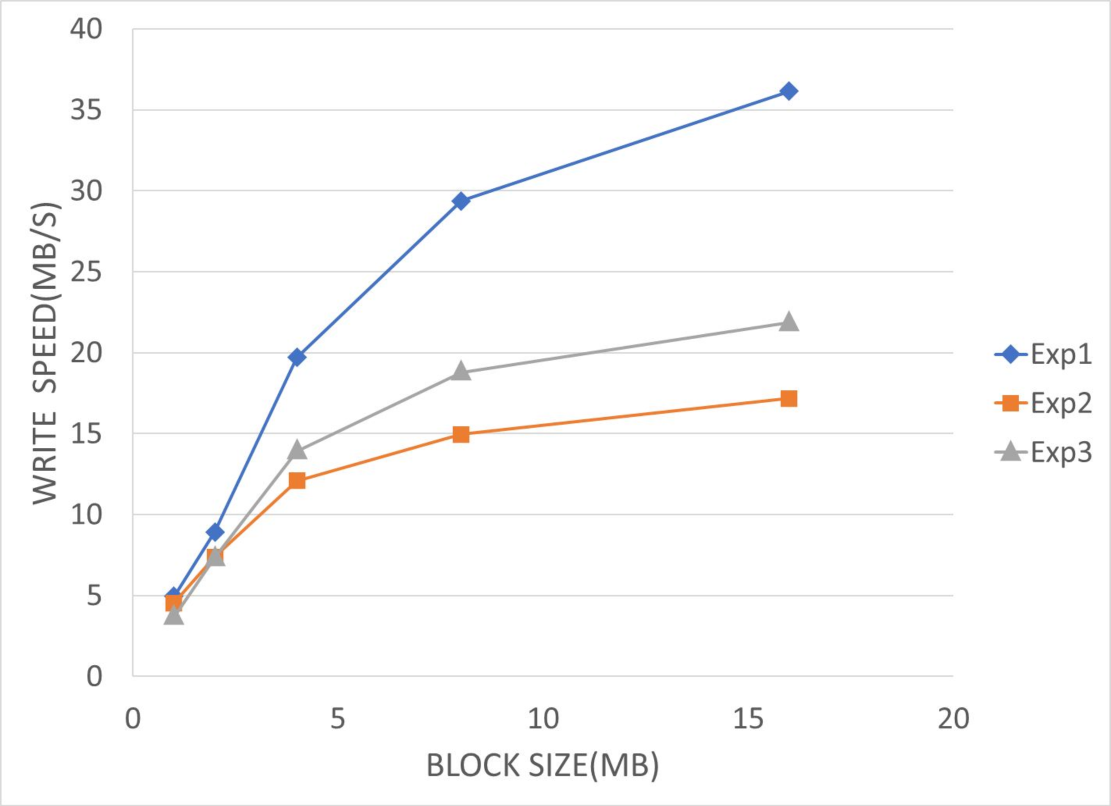
Results(Deduplication)

- In Experiment 1, decreasing deduplication ratio with increasing block size.

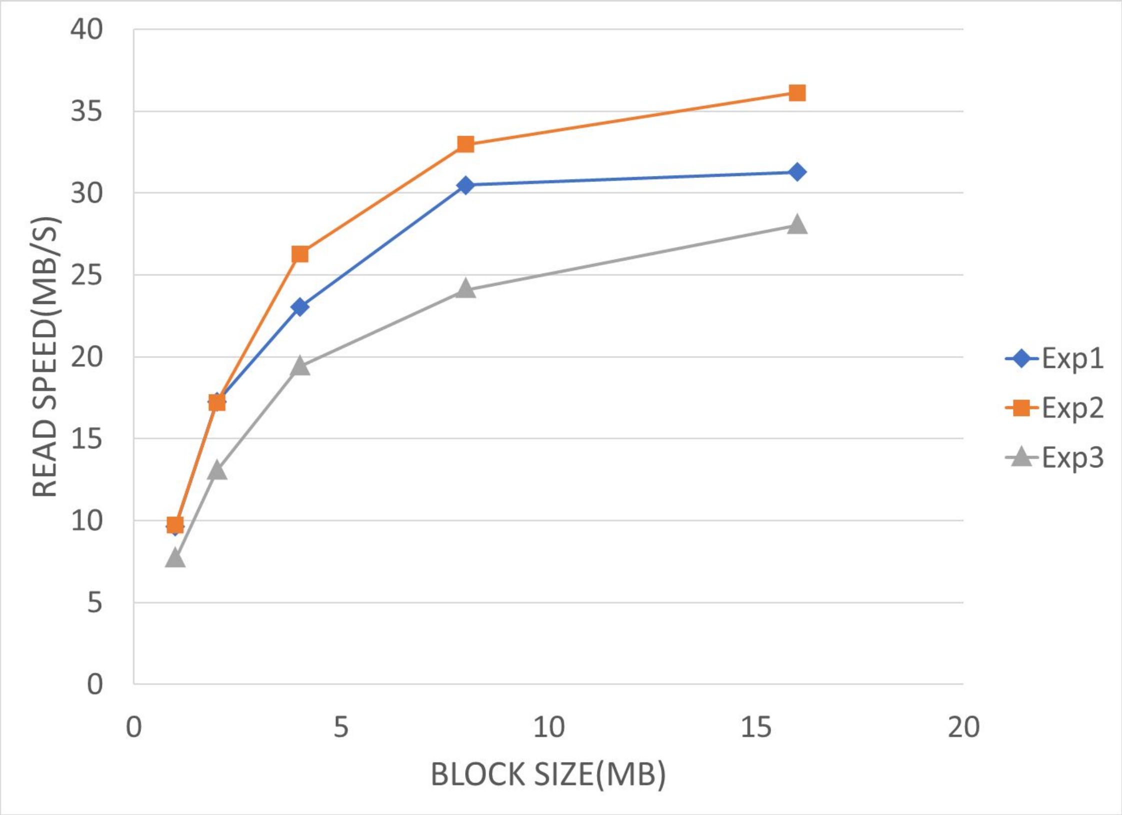
Block Size(MB)	1	2	4	8	16
Deduplication ratio	256:1	128:1	64:1	32:1	16:1

- In Experiment 2, no deduplication.
- In Experiment 3 ,deduplication ratio of 1.53:1 across different block sizes.

Results(Write Speed)



Results(Read Speed)



Implications

- The results for write speed suggest that there is a trade off between performance and deduplication.
    - Smaller block size provides finer granularity for detecting duplicates but also increases communication cost per byte of data.
- However, dependent on data type:
    - No benefit from smaller block sizes if there are file level duplicates or no duplicates at all.
- Large block size may be appropriate given the importance of read/write speeds.

Implications

- Two major sources of additional overhead:
  - Computing the hash value of a data block
  - Performing byte-by-byte comparison between two data blocks with the same hash value
- While not explicitly measured, we are able to infer the impact of hashing and byte-by-byte comparison through our experiments
  - In experiment 1, read and write speeds are almost the same
    - This suggests that the computations of hash values and byte-by-byte comparisons do not significantly hinder performance

Implications

- Data deduplication could improve write speed
- A comparison of experiments 1 and 2:
  - **In experiment 1, every operation involves a byte-by-byte comparison**
    - Every block is a duplicate
  - **In experiment 2, no operation involve byte-by-byte comparisons**
    - No duplicate blocks
- Write speeds in experiment 1 are significantly faster than in experiment 2
  - This suggests duplicate detection using byte-by-byte comparison is at least as fast as writing a new block of data if not faster.
  - Byte-by-byte comparison + read is faster than write.

Conclusion & Future Directions

- We built a functioning system that address the issues of conventional systems.
- Some promising experimental results → suggests minimal overhead due to deduplication.
- Various limitations
  - Simple model without fault tolerance, security, load balancing etc.
  - No standard benchmark tests
- But our model is worth further investigation.
- Future research should focus on building and testing a full scale system that is much more reliable and robust.

Meta-discussion

- Challenges
  - Time constraint
  - COVID-19
  - Using C++
  - Debugging
- But, we are done!

Thank you!!!