Row level repair

Asias He <asias@scylladb.com>

Sep 2019, Las Vegas

Next Generation Cassandra Conference 2019

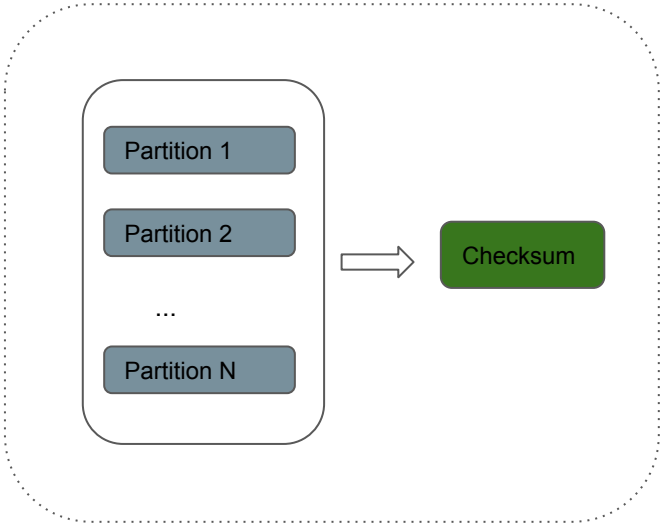
Row level repair introduction

What is repair

- Important maintenance operation
- Detect mismatch between replicas on different nodes
- Fix the mismatch

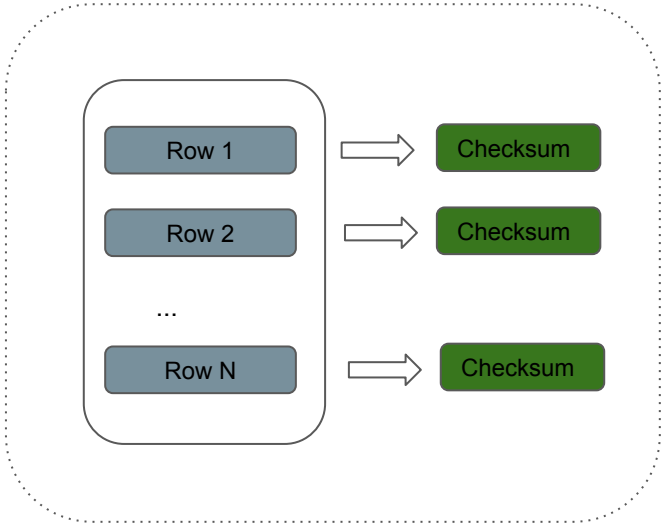
Problems in partition level repair

- Partition level
  - Calculate checksum for group of partitions
    - Merkle tree leaf node in Cassandra
    - Roughly 100 partitions in Scylla
  - Single row mismatch causes whole group partitions to be synced
  - Even a single partition can be large
  - Unnecessary on wire data transmission
- Two stage repair
  - Checksum to find the mismatch
  - Streaming to fix the mismatch
  - Read data twice, checksum + streaming



Introduce the row level repair

- Row level
  - Calculate checksum for each row
  - Set reconciliation algorithms
  - Only exchange mismatch rows
  - Avoid unnecessary data transmission
- Single stage repair
  - Reuse data read from sstables for checksum
  - Use RPC framework to send the mismatched row directly from the buffer without streaming service
  - Read data once only from disk



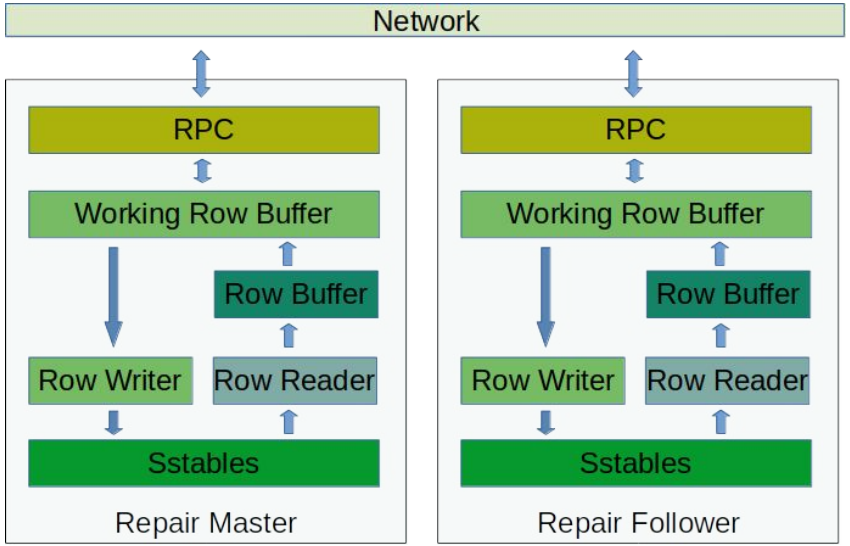
Row level repair details

Row level repair algorithm overview

- Repair master and followers
  - Master is the node running the nodetool repair command
  - Followers are the node holds a replica for a token range
- Major steps
  - Negotiate sync boundary so that master and followers agree on the range to work on
    - Sync boundary define a range contains rows that fits in memory
    - The range can be smaller than a single partition, so we can work on only part of a big partition
  - Repair master gets missing rows from followers so that master contains all the rows
    - The missing rows can be rows with same key but different content or the row with the key is actually missing
  - Repair master sends missing rows to followers so that followers contain all the rows

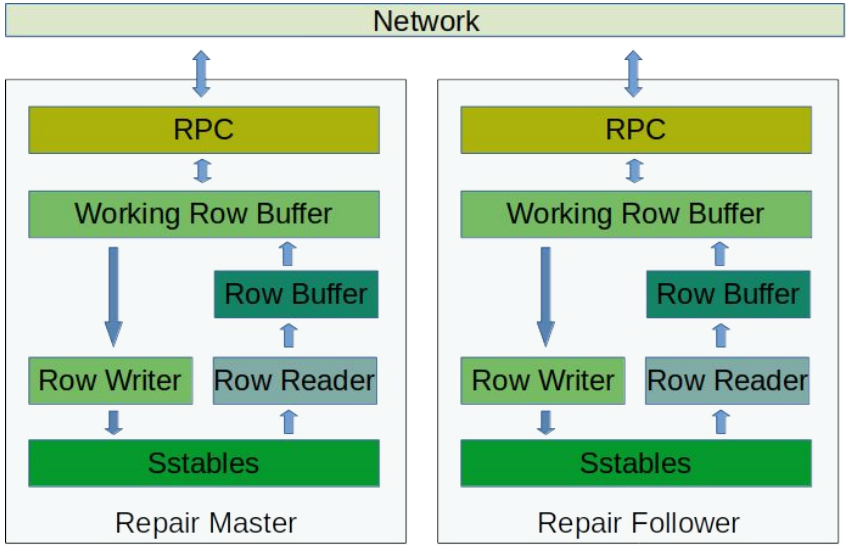
Step 1: Negotiate the sync boundary

- Master and followers
  - Fill the row buffer until it is full
  - Calculate hash for each row in row buffer
  - Return combined hash for all hashes
  - Return proposed the sync boundary to use
    - Rows are ordered in row buffer
    - Boundary of last row in row buffer is the proposed boundary
- Master
  - Checks if combined hash and proposed sync boundary are the identical for all nodes, if so, rows in row buffer are already synced, move to next range
  - Decides the sync boundary to use which is the smallest of all proposed boundaries



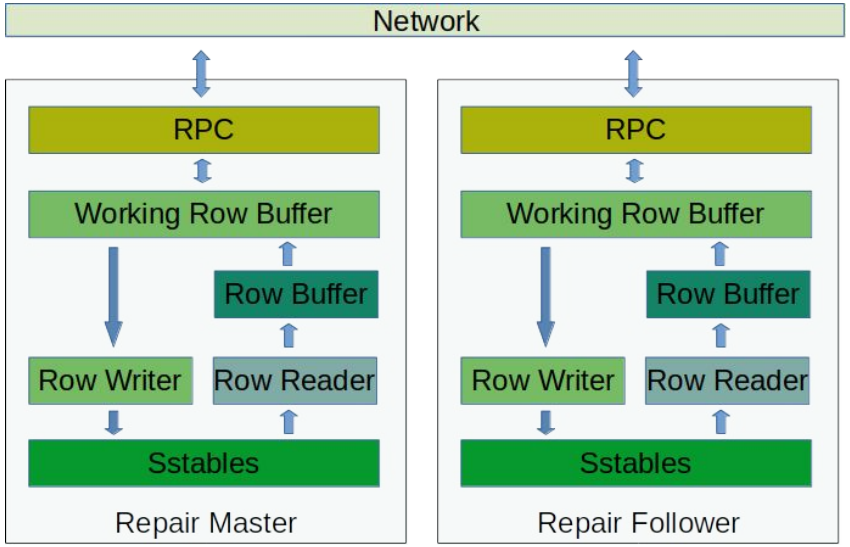
Step 2: Move rows to working row buffer

- Master and followers
  - Move rows whose boundary is smaller than the sync boundary to working row buffer
  - Return the combined hash of all rows in working row buffer
- Master
  - Checks if the combined hash of all nodes are identical, if so, rows in the working row buffer are already synced, move to next range



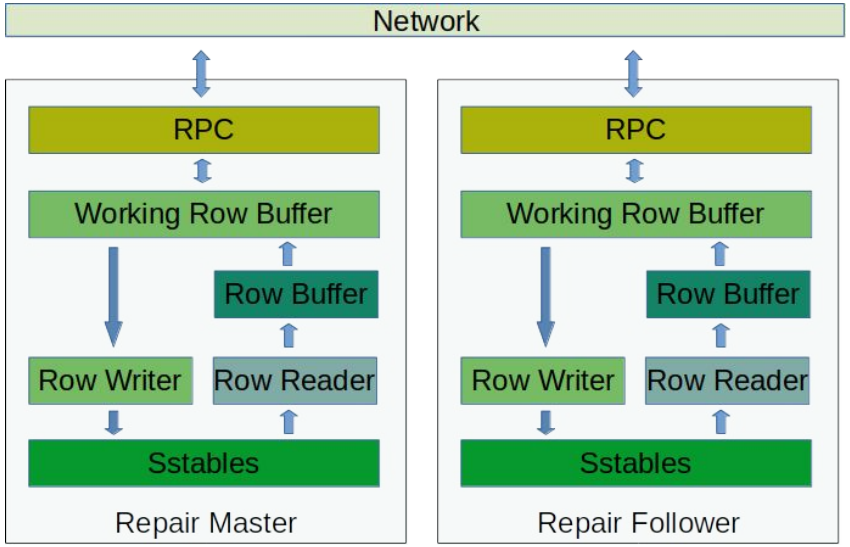
Step 3: Get the full row hashes in working row buffer

- Master and followers
  - Master asks followers to send all the hashes of th rows, not all the rows, in working row buffer to master
    - Simplest implementation
    - Work best if the difference is big
  - Other method like using better set reconciliation algorithm to get hashes of followers
    - Avoids send the full hashes
    - Work best if the difference is small
- Master
  - Knows exactly what rows each node has and misses



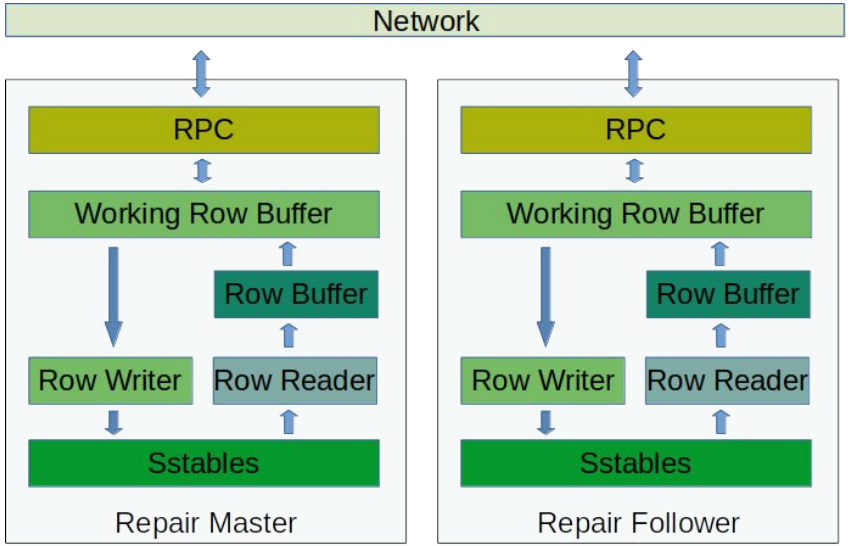
Step 4: Get missing rows from repair followers

- Master
  - Compares hashes in working row buffer between master and followers to decide which rows to fetch from followers, e.g.,:
    - n1={1,2,3}, n2={1,2,4}, n3={1,4,5}
    - n1 pulls {4} from n2
    - n1 pulls {5} from n3
  - Adds the rows get from followers to working row buffer and writes them to sstables
  - Now, master contains all the possible rows



Step 5: Send missing rows to repair followers

- Master and followers
  - Master compares hashes in working row buffer between master and followers to decide which rows to push to followers, e.g,:
    - Originally n1={1,2,3}, n2={1,2,4}, n3={1,4,5}
    - After previous step: n1={1,2,3, 4, 5}, n2={1,2,4}, n3={1,4,5}
    - n1 sends {3,5} to n2
    - n1 sends {2,3} to n3
  - Followers write the rows received from master to sstables
  - Now, both master and followers contain all the rows and are in sync
  - Move to next range until all the ranges are repaired



Row level repair test results

Repair test results

Test case	Description	Time to repair		Ratio
		Partition Level Repair	Row Level Repair	
0% synced	One of the nodes has zero data. The other two nodes have 1 billion identical rows.	49.0 min	37.07 min	x1.32 faster
100% synced	All of the 3 nodes have 1 billion identical rows.	47.6min	9.8 min	x4.85 faster
99.9% synced	Each node has 1 billion identical rows and 1 billion * 0.1% distinct rows.	96.4 min	14.2min	x6.78 faster

Test setup:
- Three-node Scylla 3.0/3.1 cluster
- AWS i3.4xlarge instances
- Each node has 1 billion rows and 1TB data
- Each row is 1 KB
- Each partition only has 1 row
- No background read or write workload

Repair test results

Description	Partition Level Repair	Row Level Repair	Transfer data ratio
Bytes sent on wire for 99.9% synced case on master	120.52 GiB	4.28 GiB	3.6%
Bytes received on wire for 99.9% synced case on master	60.09 GiB	2.14 GiB	3.6%

Repair test results discussion

- The speed up comes from 3 factors:
  - No overstreaming
    - Only 3.6% of partition level repair in 99.9% synced case
    - The workload is one row per partition, with multiple rows per partition workload, even better results
  - Faster hash algorithm
    - 256-bit cryptographic SHA256 hash v.s. 64-bit no-cryptographic xxHash
  - More parallelism
    - Repair one token range in parallel v.s. 16 token ranges in parallel
    - Especially helpful for cluster with high latency network

Summary

- Better repair resolution
  - Improved from group of partitions resolution to single row resolution
- Avoid overstreaming
  - Significantly reduces the amount of data transferred on wire
  - Reduces repair time
- Less disk access
  - Read once for both checksum and sync between nodes

Thank you!