Revamping Spark™ Shuffle with Apache Celeborn™ at Pinterest Scale

Aria Wang, Nan Zhu

Oct 2024

Pinterest

Agenda

- Introduction
- Apache Celeborn™ @ Pinterest
- Taming Celeborn™ @ Pinterest
- Summary and Future work

About Us

Aria Wang

Software Engineer II - Data Processing Platform. Working on Spark on EKS platform

Nan Zhu

Tech Lead of Spark Team in Pinterest, working on Spark ecosystem in Pinterest

Pinterest owns one of largest Spark™ deployments in the world

4000+ workflows

18K computing nodes

~30 clusters

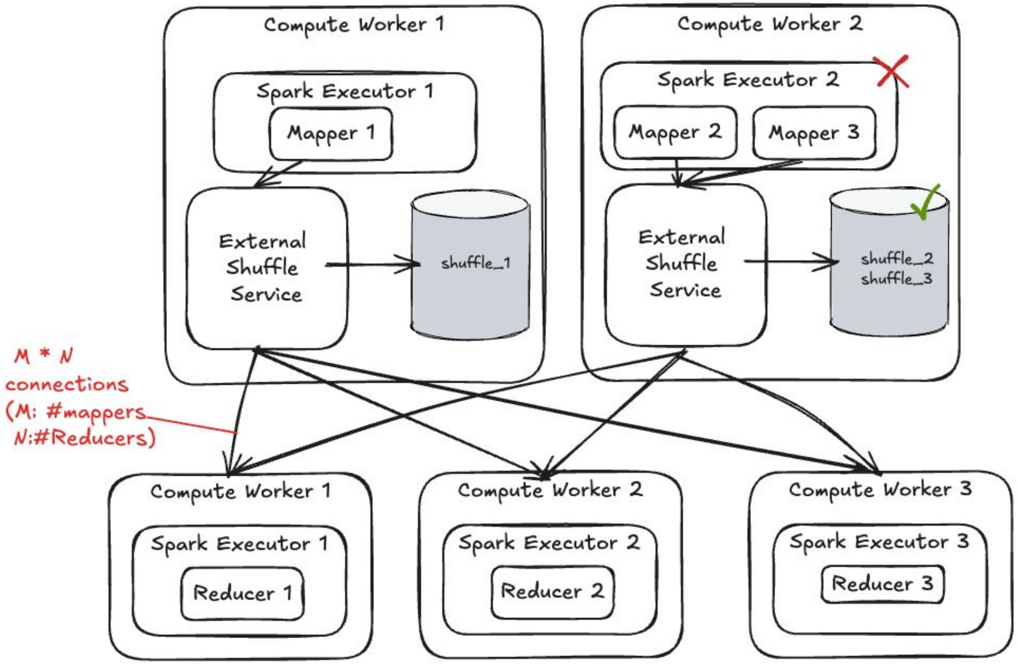
100s of K Jobs (daily)

100s of PB Input (daily)

100s of PB Shuffle (daily)

the usage is expanding in a daily basis

Spark™ on Hadoop® YARN with External Shuffle Service (ESS)

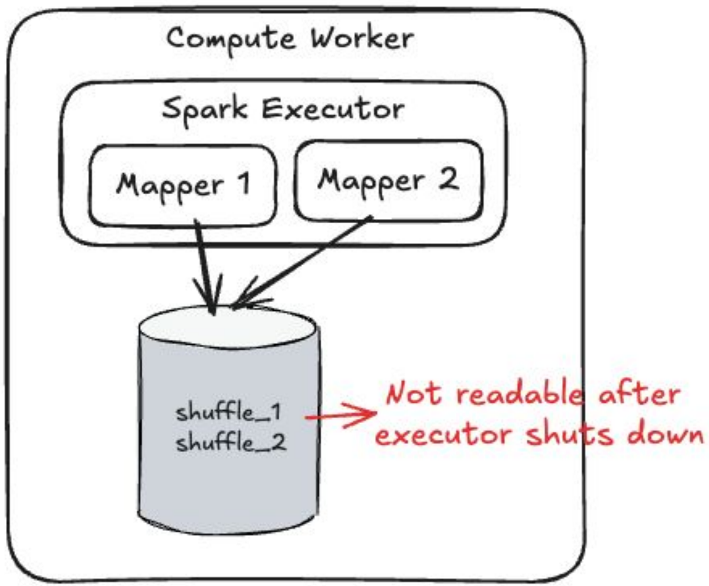


- ESS support is provided in Spark™ on Hadoop® YARN version
  a. scale down executors without losing shuffle files
- Main Challenges
  a. Uneven disk distribution → some nodes get disk full due to large shuffle
  b. Slow shuffle read → caused by busy ESS with M * N full-mesh connections

Metric	75th percentile	Max
Duration	37 min	1.3 h
GC Time	39 s	1.5 min
Spill (memory)	5.2 GiB	5.7 GiB
Spill (disk)	1 GiB	1.1 GiB
Output Size / Records	677.2 MiB / 14791550	705.4 MiB / 36720108
Shuffle Read Size / Records	1.1 GiB / 17551514	1.1 GiB / 39478654
Shuffle Read Blocked Time	23 min	1.1 h

Spark™ on EKS without ESS support

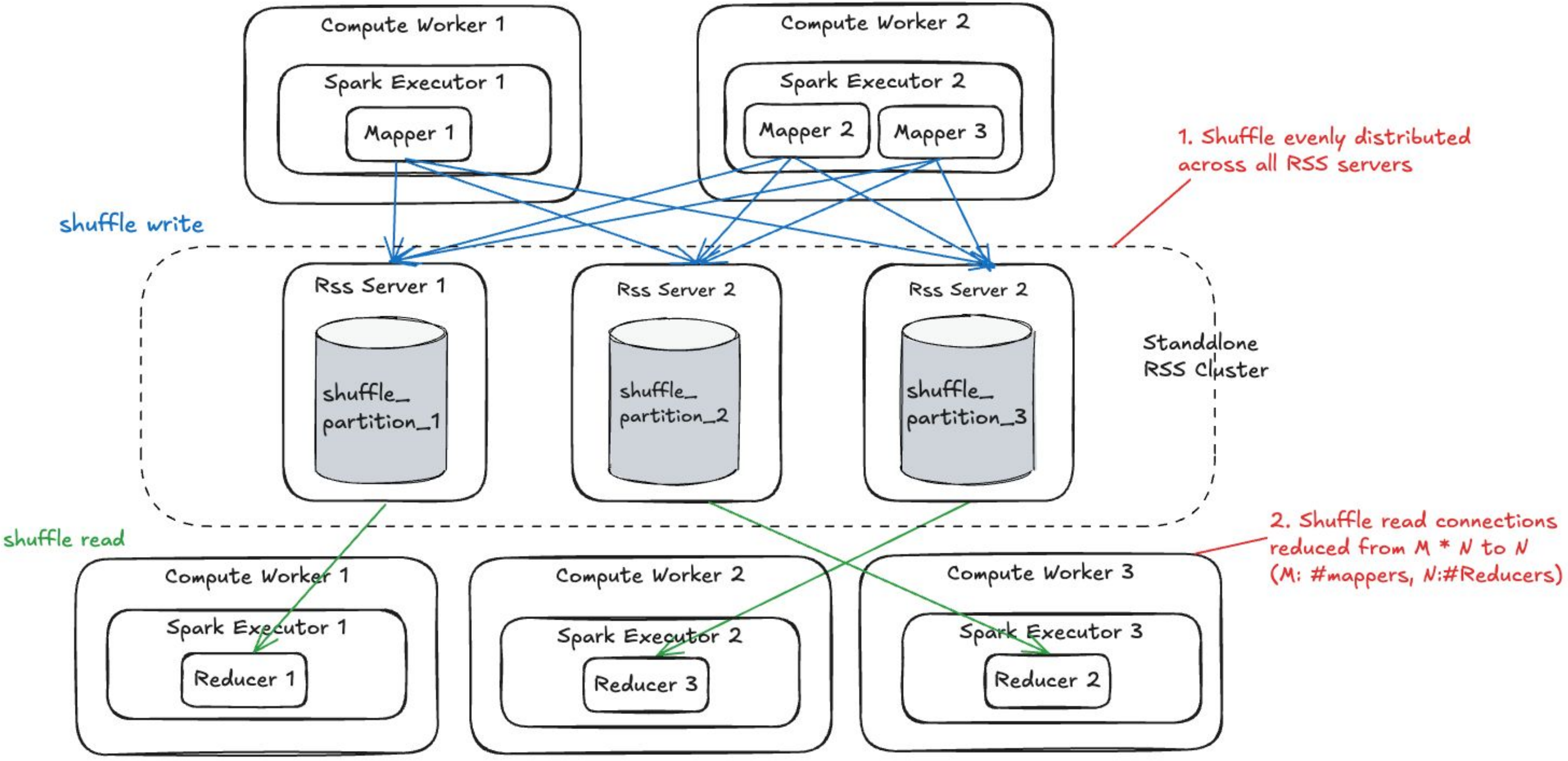
- No support of ESS for Spark™ on K8s
  - only vanilla shuffle
- No shuffle data management to support dynamic allocation
  - Dynamic allocation
    - most effective way to achieve “near-optimal” resource allocation
  - When executor is scaled down by Dynamic Allocation → Shuffle data loss → recompute



Key to the issues:

Shuffle Data Management System

RSS (Remote Shuffle Service) 1.0 - Zeus based



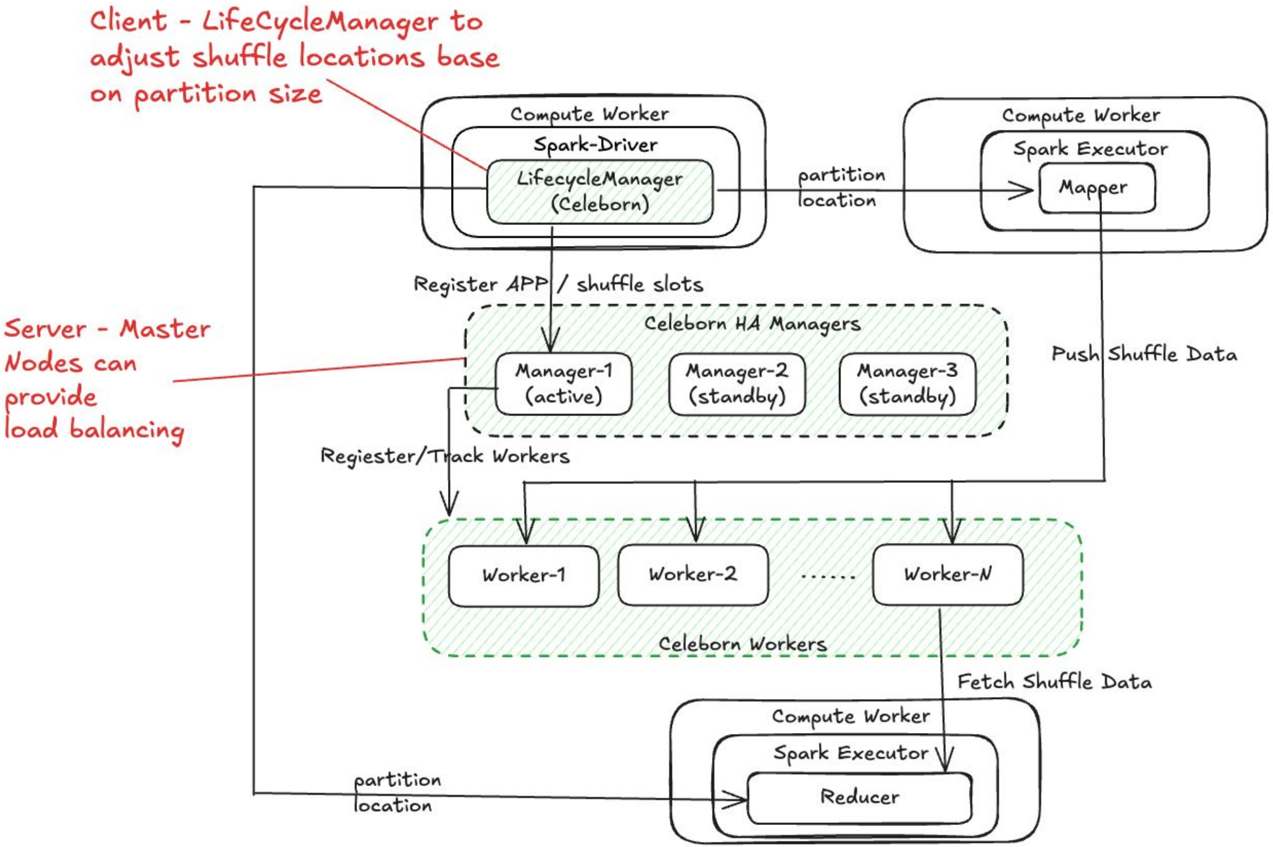
RSS 1.0 - Main challenges

- Large partition skew causing disk issues
    - 1 partition maps to 1 RSS server strategy
    - if 1 partition > 3TB → Rss would fail the app to protect cluster
- Operational overhead
    - requires decommissioning Rss servers to make new deployment
    - usually takes 1-2 days
- Shuffle write performance
    - hash based shuffle writer slowness
- Requires shuffle files replication
    - to ensure job can read backup shuffle file when the primary one is lost (usually due to node termination)
    - doubled the disk usage on RSS cluster

RSS 2.0 – Apache Celeborn™

- Open Source project started by Alibaba Corp (website link)
- Intermediate data service for Big Data compute engines to boost performance, stability, and flexibility
- Integrated with Spark™, Map-Reduce, and Flink®, to provide remote shuffle service
- Used by LinkedIn, Stripe, and other peer companies
- Active community and support channels

RSS 2.0 – Apache Celeborn™



Server: scalability improvement

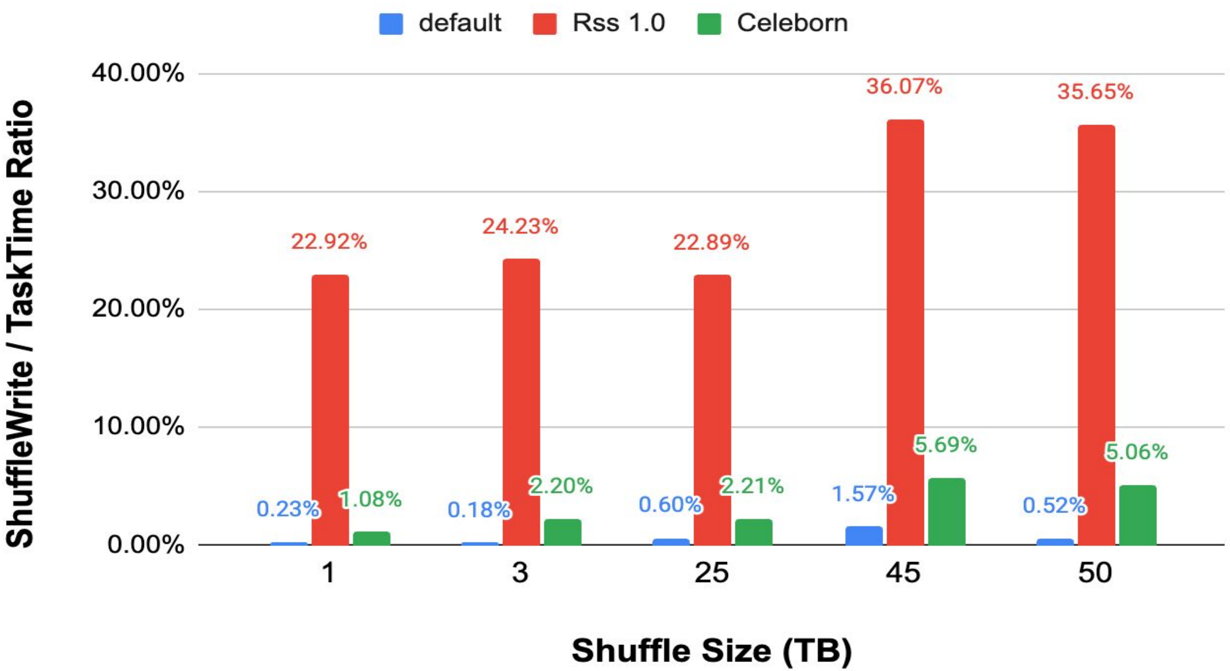
Rss 1.0

- 1 shuffle partition need 2 copies on Server side
- Server upgrade takes 1-2 days
- Heavily skewed jobs can cause server disk full
- No load balancing

Rss 2.0: Apache Celeborn™

- stage retry to recover shuffle files → 1 shuffle partition 1 copy, reduced disk & network by **50%**
- Server rolling upgrade takes within **10 mins**
- Evenly spread skewed partition to multiple workers
- Load balancing supported by manager nodes

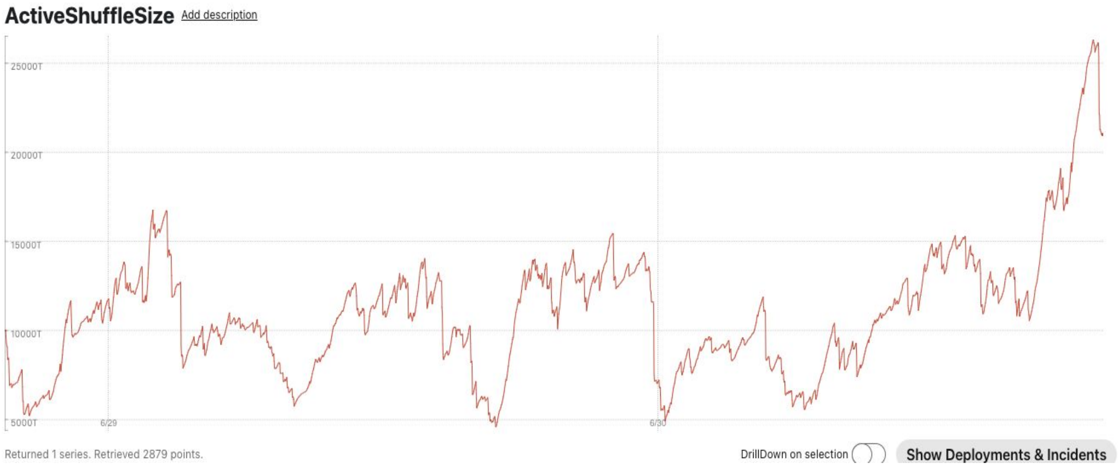
Client: shuffle write improvement



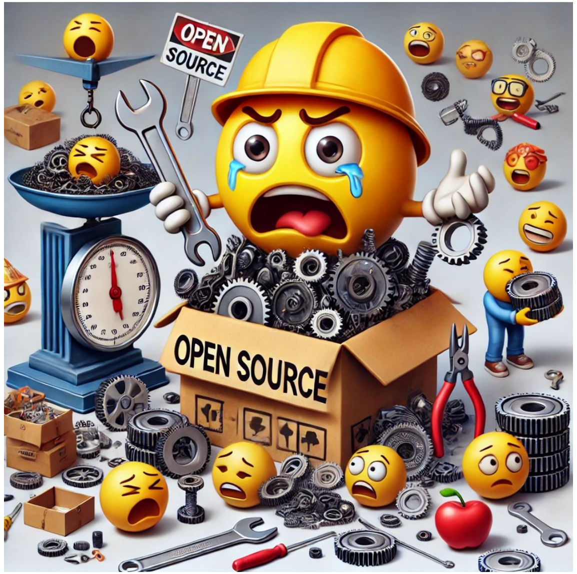
- Significantly reduced the shuffle write overhead, and solved the bottleneck of Rss 1.0

Celeborn™ Adoption in Pinterest

- Set up dedicated cluster for Celeborn™ on EKS
  - 3 Managers for HA, serving shuffle request
  - 500 Workers, serving shuffle data
- Cluster serving ~25,000 TB shuffle each day
- By switching to Celeborn™ from RSS 1.0
  - Decreased shuffle disk usage by 50%
  - Reduced computation resources by ~40%
- Current largest shuffle job ~700TB



Open-source solutions are rarely usable out of the box at Pinterest's scale



Taming Apache Celeborn™ @ Pinterest

Main Challenges of Using Apache Celeborn™ in Pinterest

- Results validation failure
  - Inconsistent results of shuffle bytes , output file counts

- Significant performance regression comparing to ESS (for some applications)

- Vulnerability to Driver OOM with big shuffle size

Two Major Learnings

IO Buffer Management
in ESS v.s. Apache Celeborn

Heavy control flow of Celeborn
in Driver side

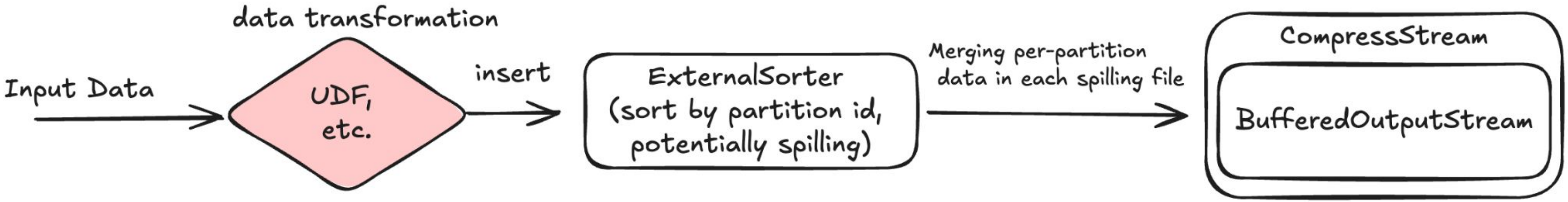
Learning 1: IO Buffer Management

IO Buffer Management
in ESS v.s. Apache Celeborn

Heavy control flow of Celeborn
in Driver side

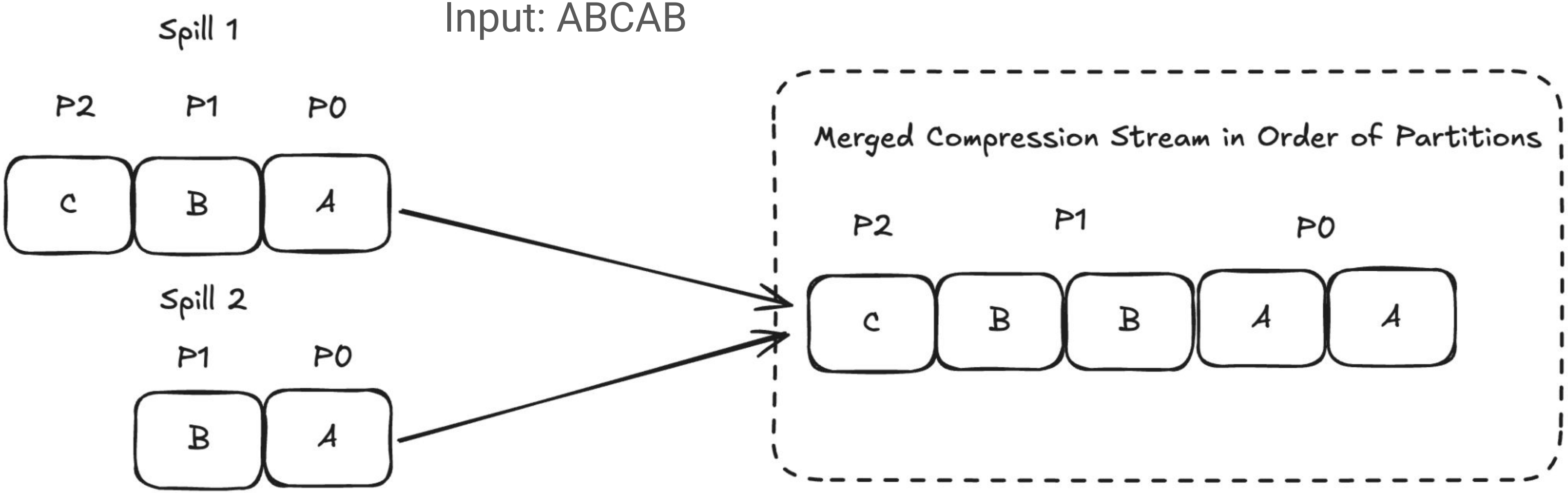
Persisting Shuffle Data (1)

- Persisting Shuffle Data to Local Disk (using ESS or Spark™ native Shuffle)



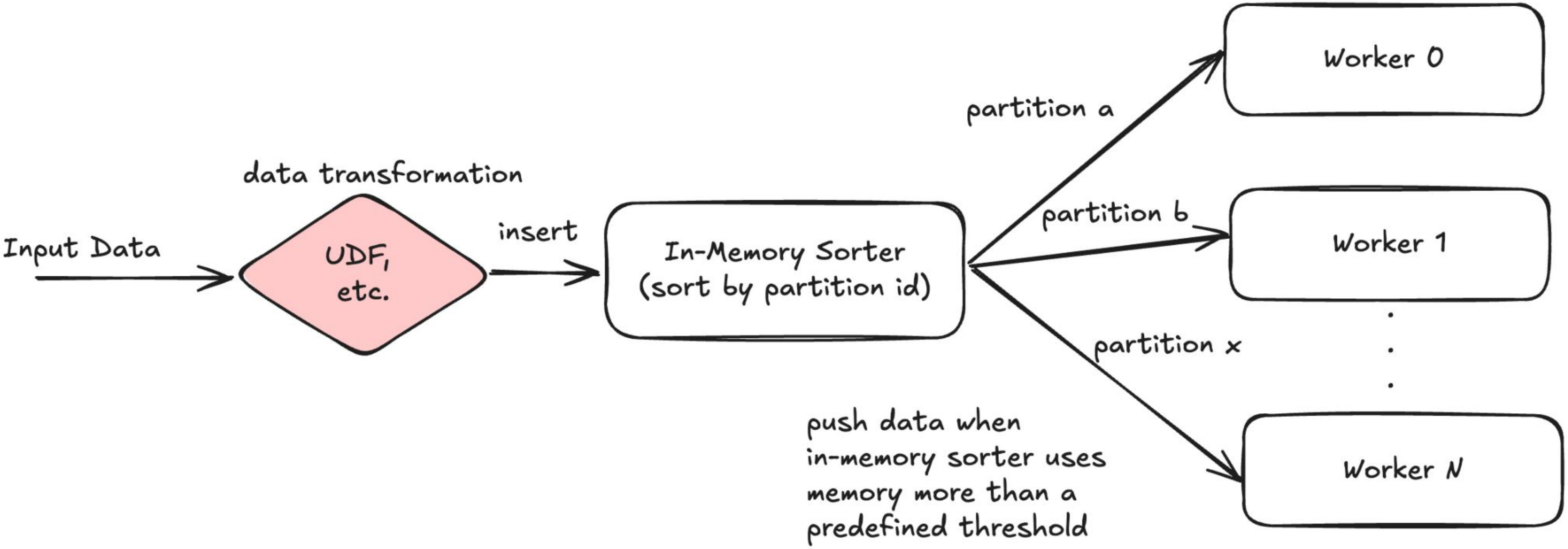
Compression Input of Local Disk Shuffle

Input: ABCAB



Persisting Shuffle Data (2)

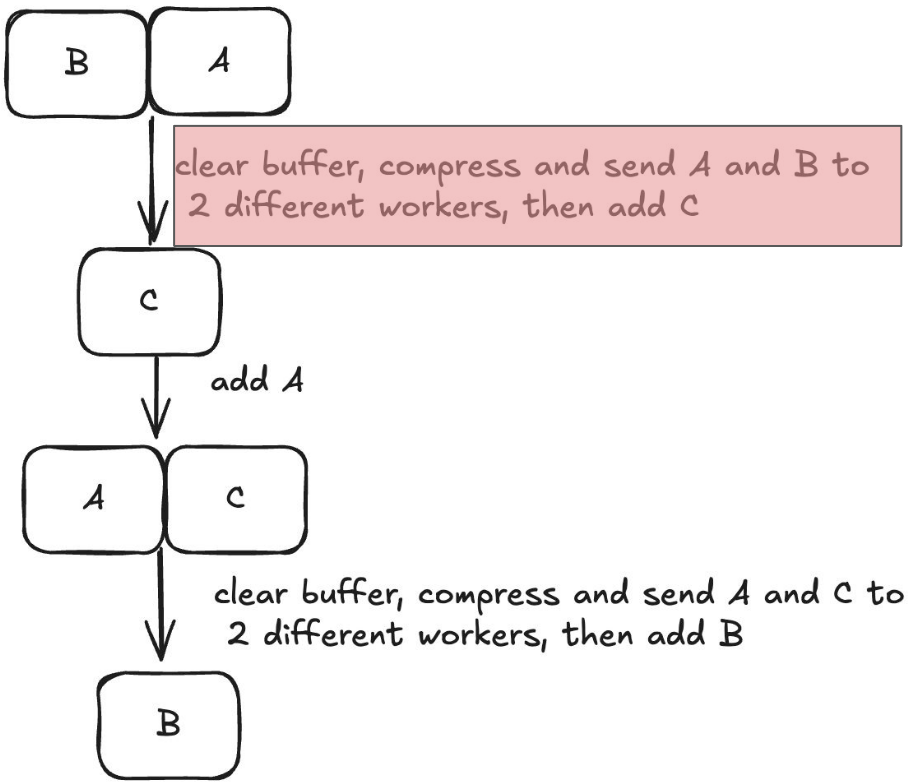
- Persisting Shuffle Data to Remote Disks (using Apache Celeborn™)



Compression Input of Remote Shuffle

Input: ABCAB

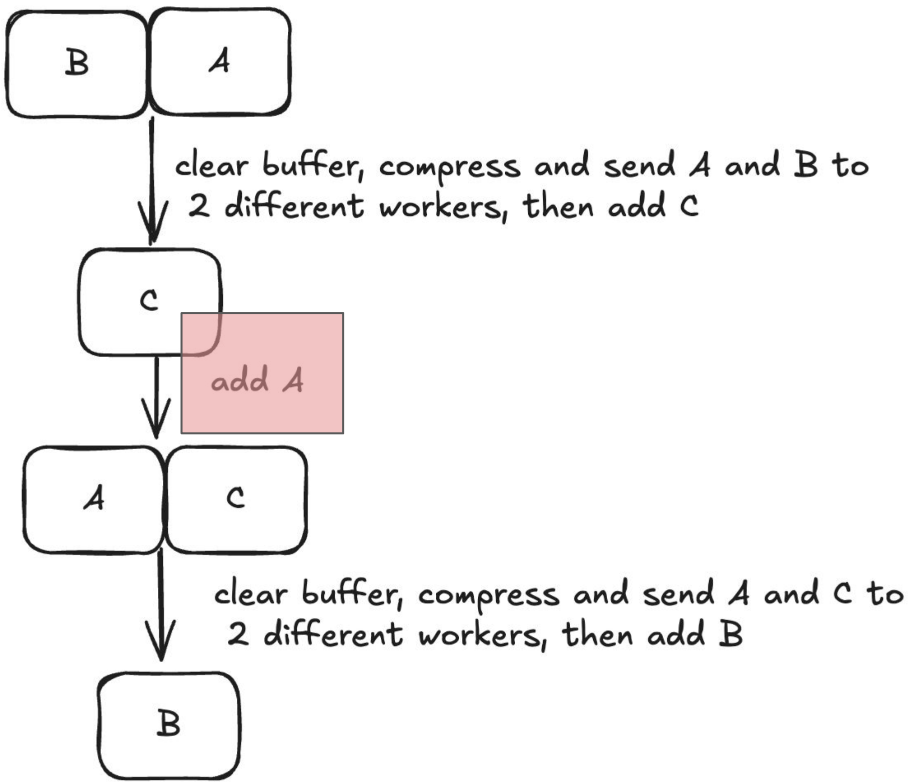
buffer size is 2 records



Compression Input of Remote Shuffle

Input: ABCAB

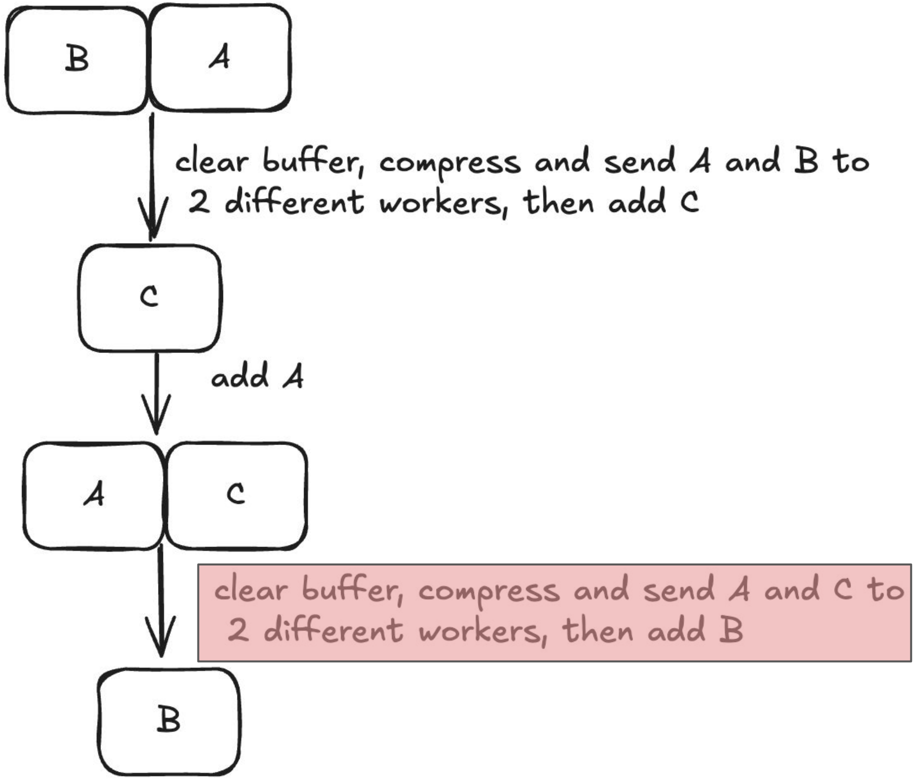
buffer size is 2 records



Compression Input of Remote Shuffle

Input: ABCAB

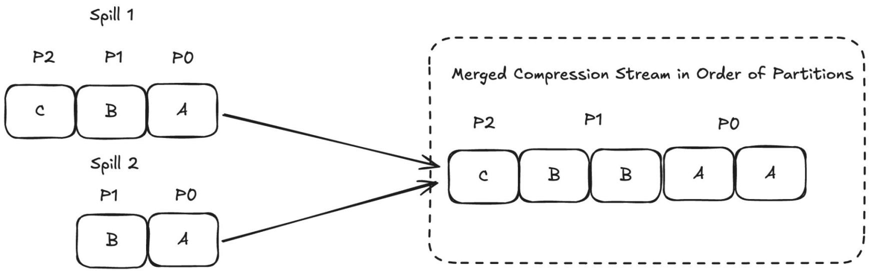
buffer size is 2 records



Different Input to Compression - Reason to Shuffle Stats Disparity

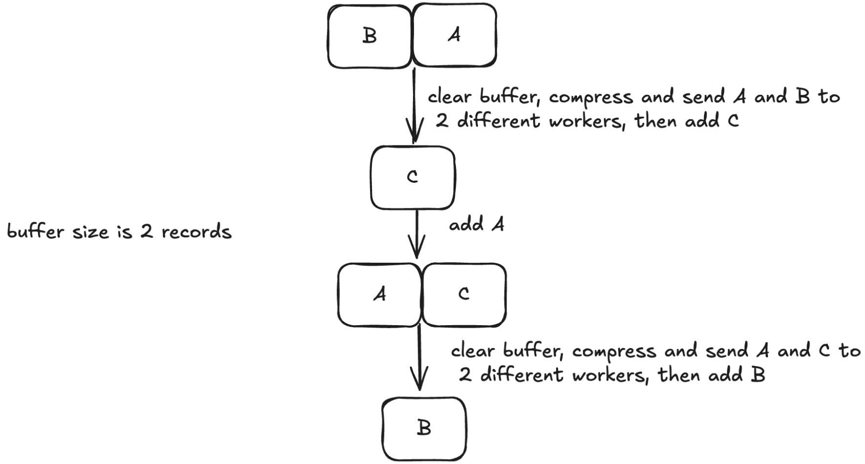
Input: “ABCAB”

Local



Input to compression stream:
AA,BB,C

Remote



Input to compression stream:
A,B,A,C,B

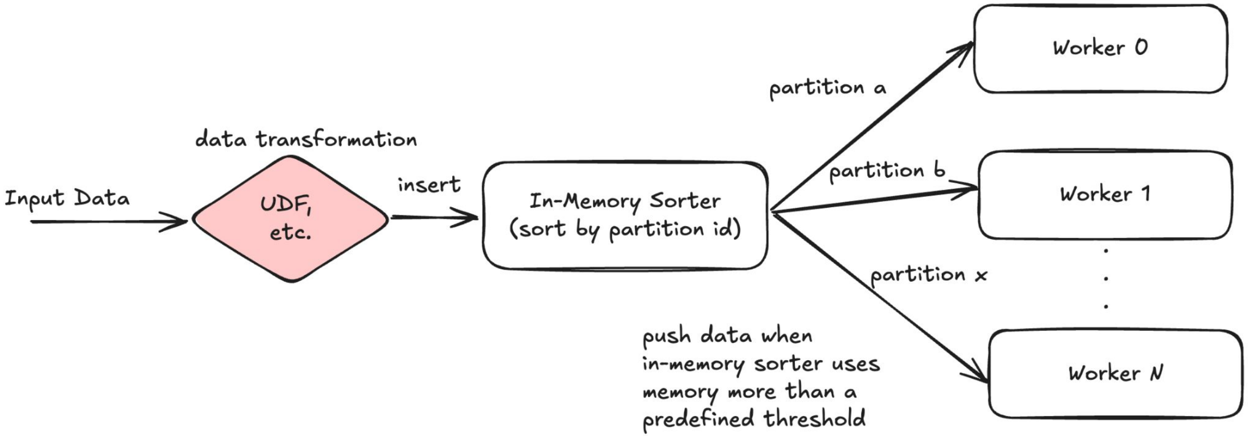
Conclusion of Result Validation Challenge

- Different buffer management strategy => different inputs to the compression

- Different inputs to the compression => different compressed shuffle size

- Different compressed shuffle size => different partitions counts/output file
    - Adaptive Query Execution optimizations consuming shuffle size
        - **Coalesce Partitions:** collecting ***shuffle sizes*** and combine contiguous and “too-small” partitions by coalescing
        - **Optimizing Skew Join:** checking ***shuffle size*** per partition and split too-big ones into smaller based on different map task Ids

Challenge 2: Performance Regression due to Small Partitions

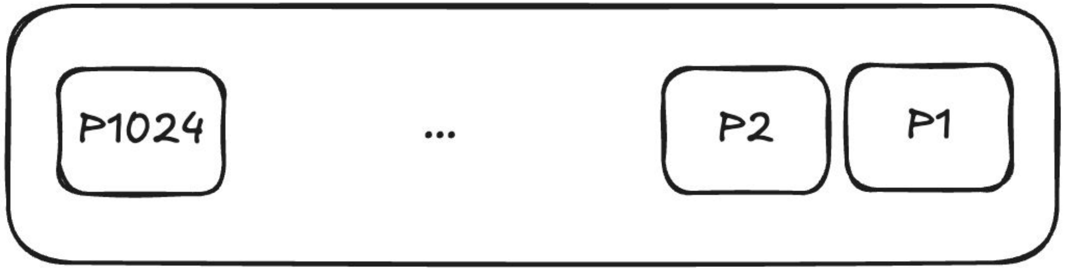


- Records for different partitions ***share*** the same buffer
- Records for different partitions are compressed and sent ***separately***

Example Case

1MB buffer

each partition's records take 1KB



compress and send 1KB data for 1024 times, which is inefficient

we observed zstd-related function calls takes a significant chunk in flamegraph

Mitigating the inefficiency by enlarging buffer adaptively

- Adaptive buffer management algorithm

  S *= 2, if ***B/C * (1 + T) < M (double the buffer size when the average pushed bytes is too small)***

  - **C**: number of data pushes
  - **B**: number of bytes pushed to remote workers
  - **S**: size of push buffer
  - **M**: max number of bytes for each partition hold in memory
  - **T:** user-defined threshold

More details at PR

Conclusion of Inefficiency compression challenge

- Shared buffer among partitions => inefficient compression when having too many small partitions

- Adaptively increasing buffer => proactively resolve the issue instead of post-user-failure and manual tuning

Learning 2: Heavy Control Flow of Celeborn™ in Driver

IO Buffer Management
in ESS v.s. Apache Celeborn

Heavy control flow of Celeborn
in Driver side

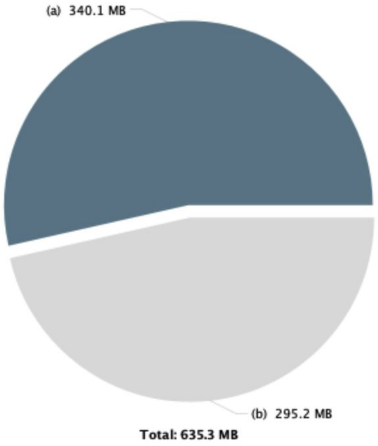
Symptom and investigation

- Spark applications with big shuffle size are vulnerable to Driver OOM
  - Counter-intuitive: executors should be more vulnerable

- Investigation with heap dump
  - Driver has a high volume of “***PartitionSplit***” RPC messages to process

the following screenshot shows the main memory consumer

Overview

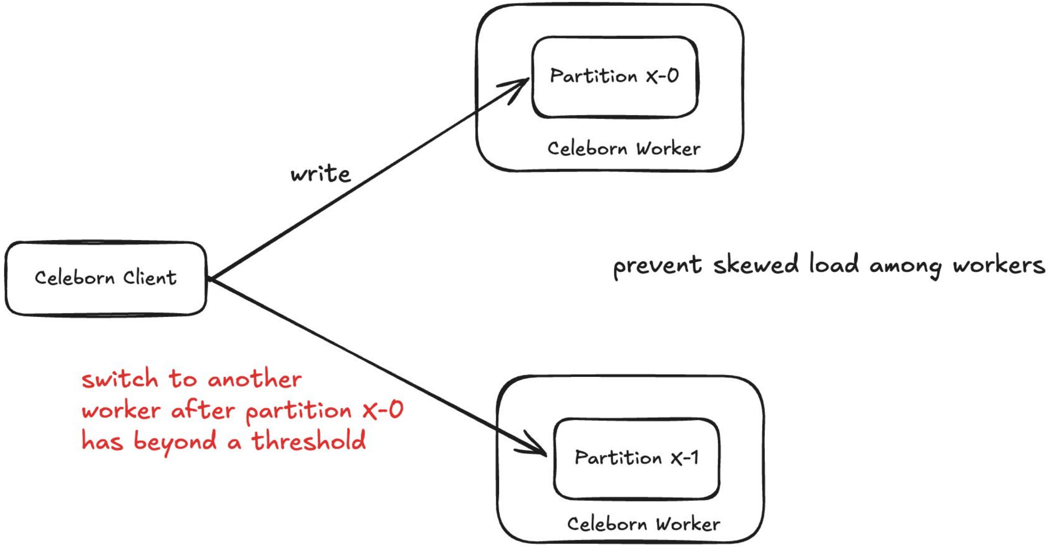


Problem Suspect 1

One instance of **"org.apache.celeborn.common.rpc.netty.Inbox"** loaded by **"sun.misc.Launcher$AppClassLoader @ 0x800043b0"** occupies **356,640,944 (53.54%)** bytes. The memory is accumulated in one instance of **"java.util.LinkedList"**, loaded by **"<system class loader>"**, which occupies **356,640,904 (53.54%)** bytes.

What is PartitionSplit?

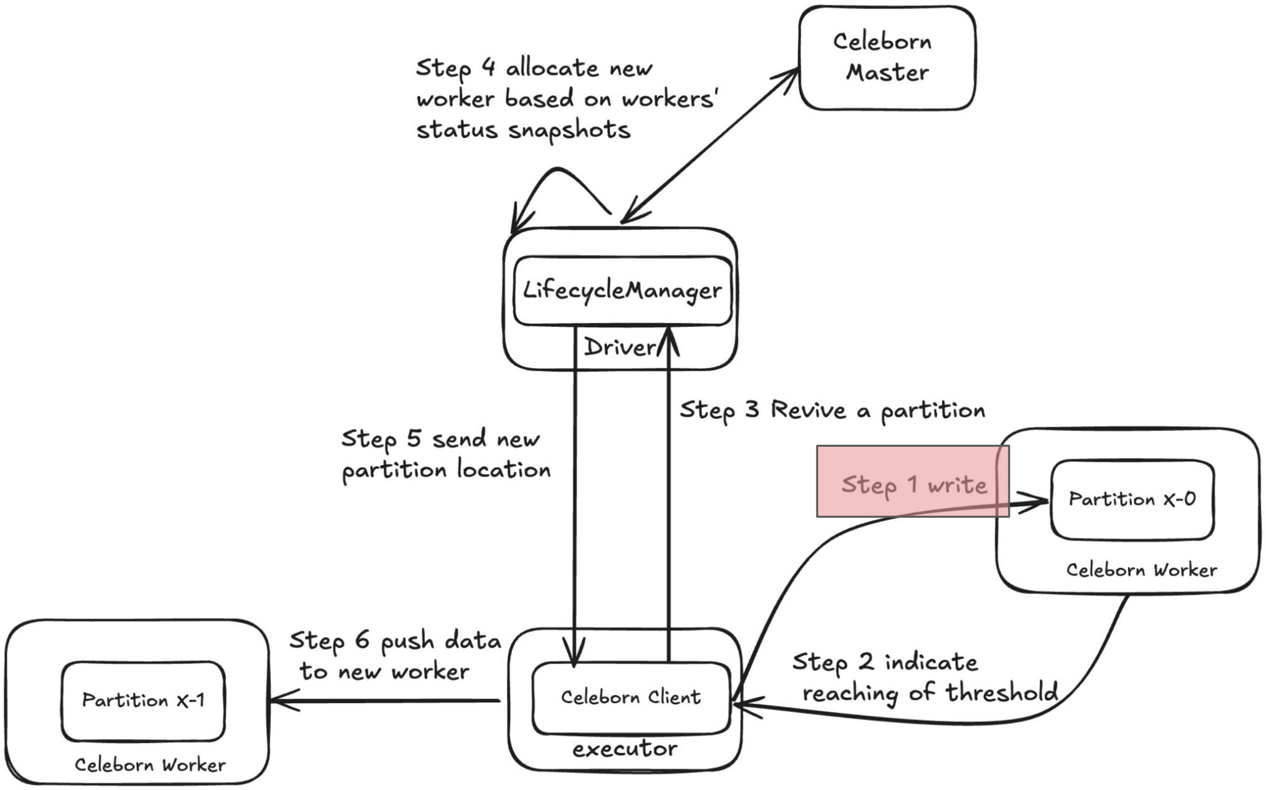
- Load balancing strategy for Apache Celeborn™ Workers



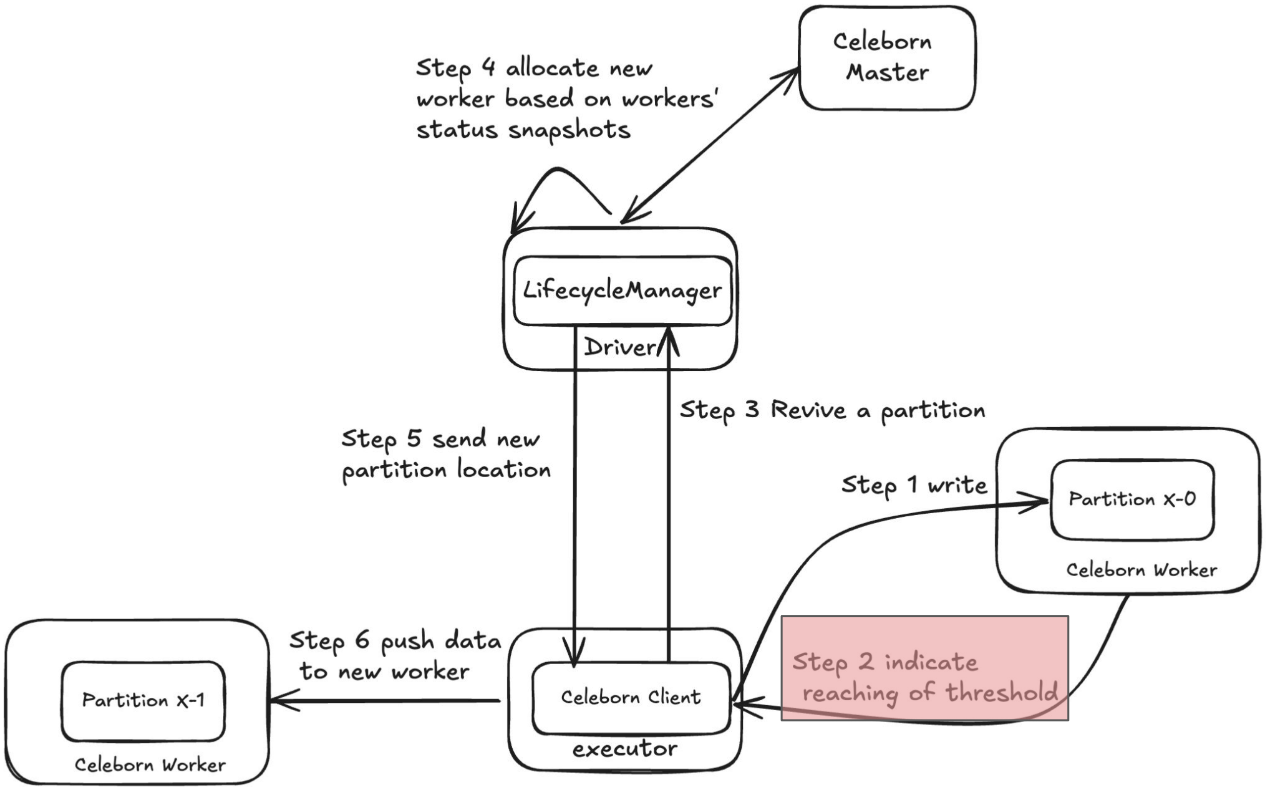
- Partition 0 - 1G, Partition 1 - 99G (2 workers)
  - Load distribution without partition Split: 1% v.s. 99%
  - Load distribution with PartitionSplit 50% v.s. 50%

Why PartitionSplit overloaded Driver?

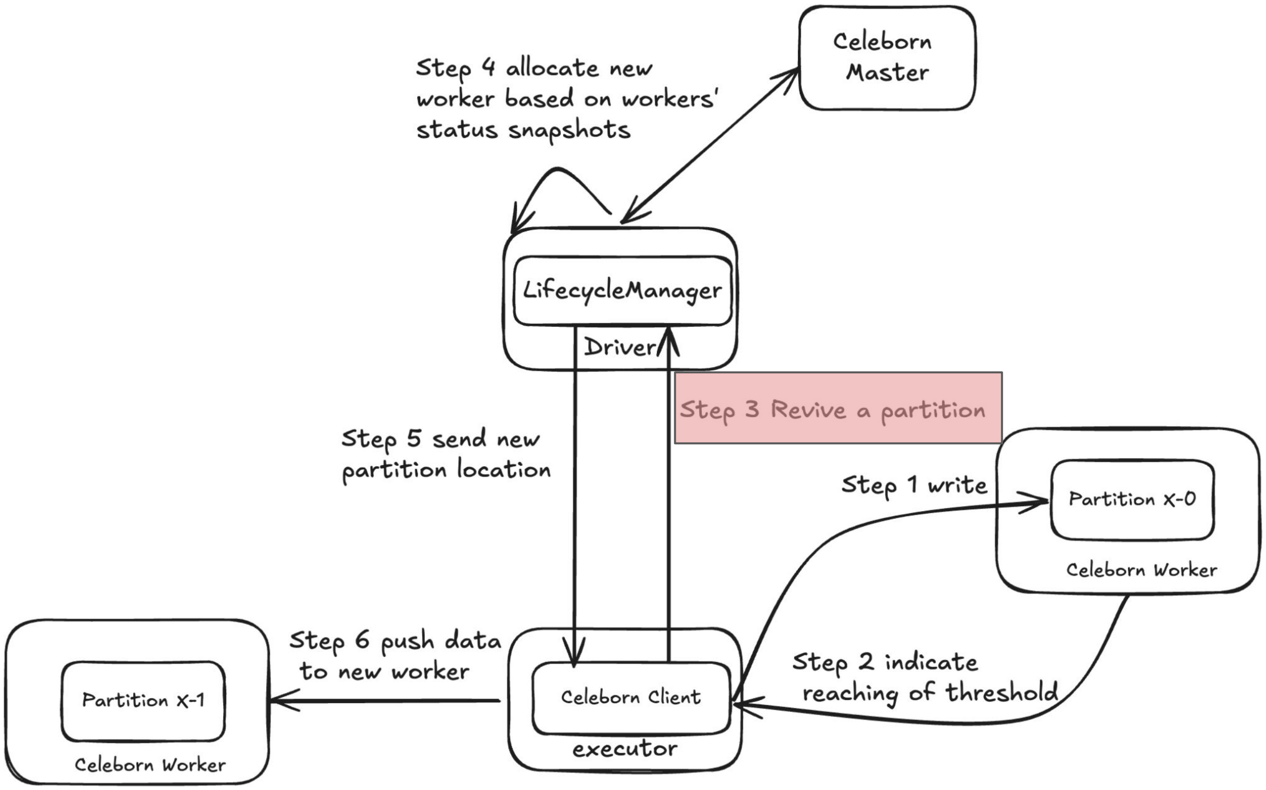
Driver OOM led by PartitionSplit (1)



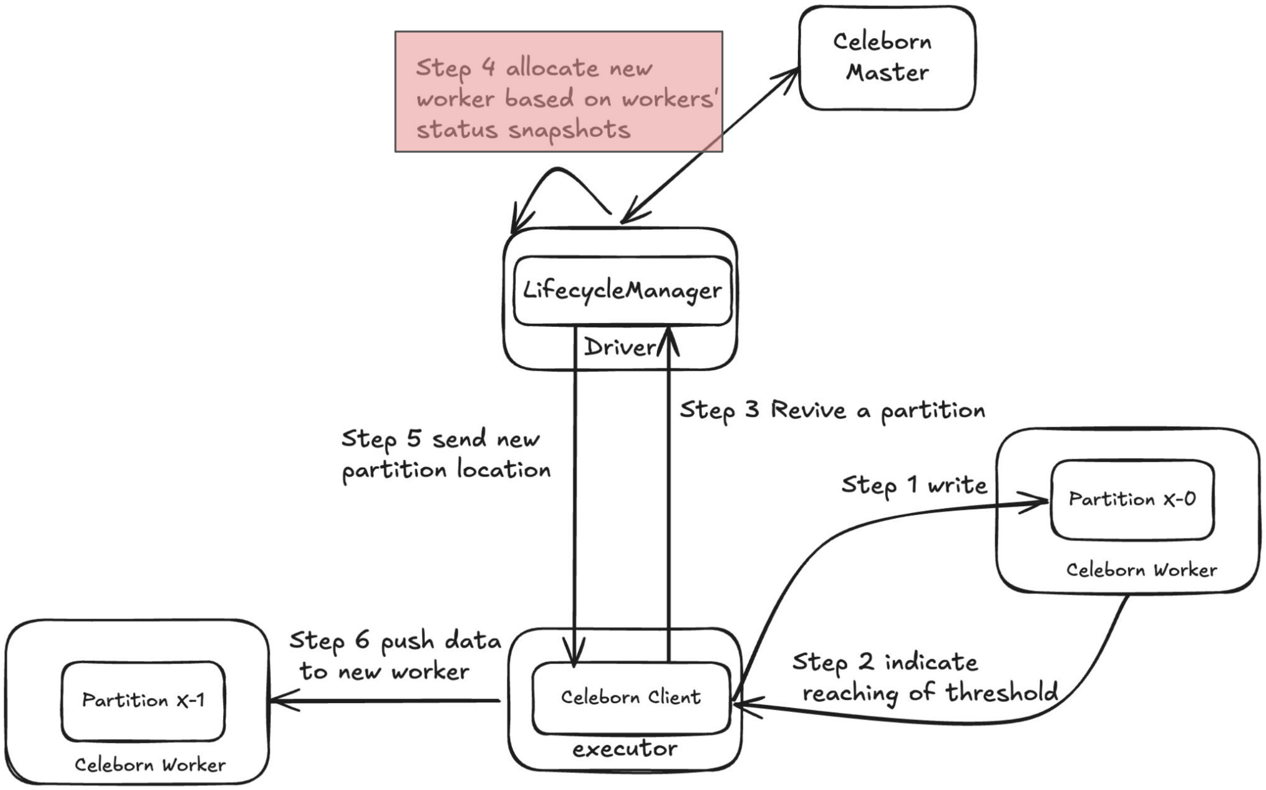
Driver OOM led by PartitionSplit (2)



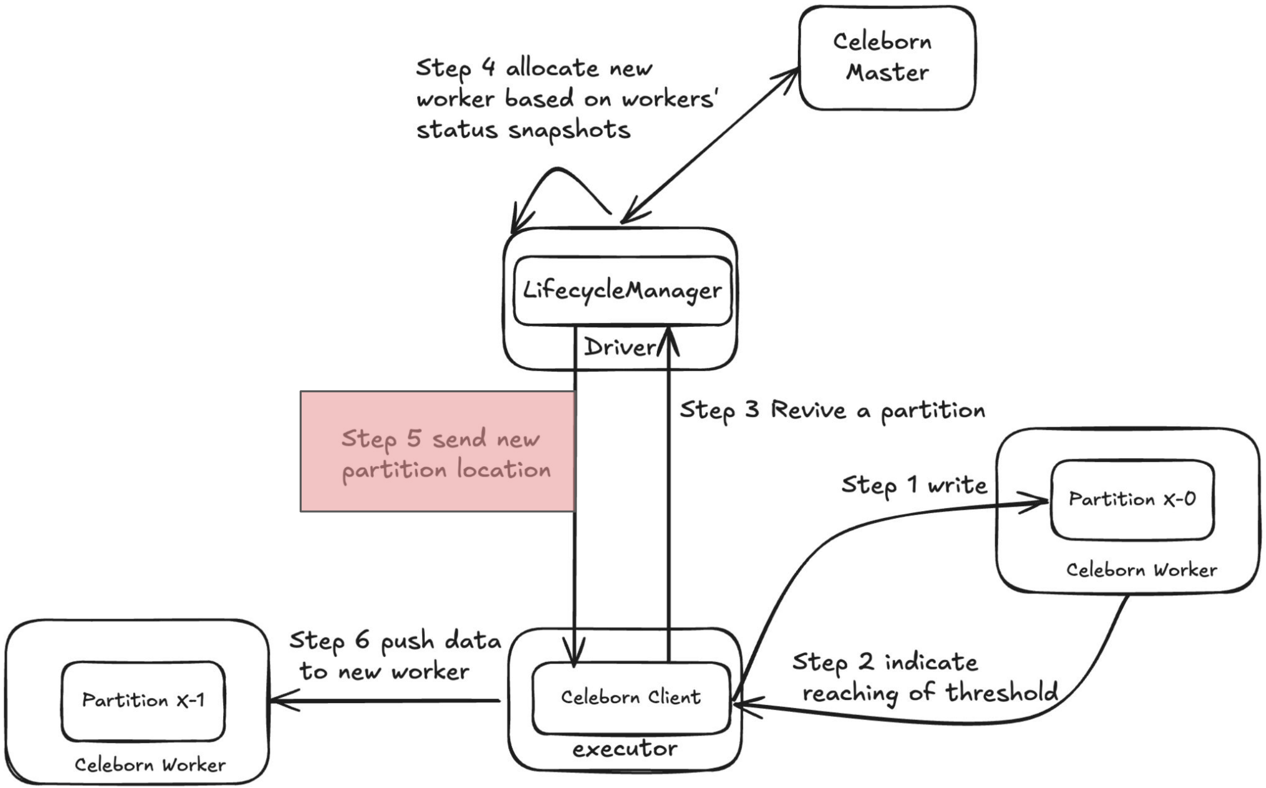
Driver OOM led by PartitionSplit (3)



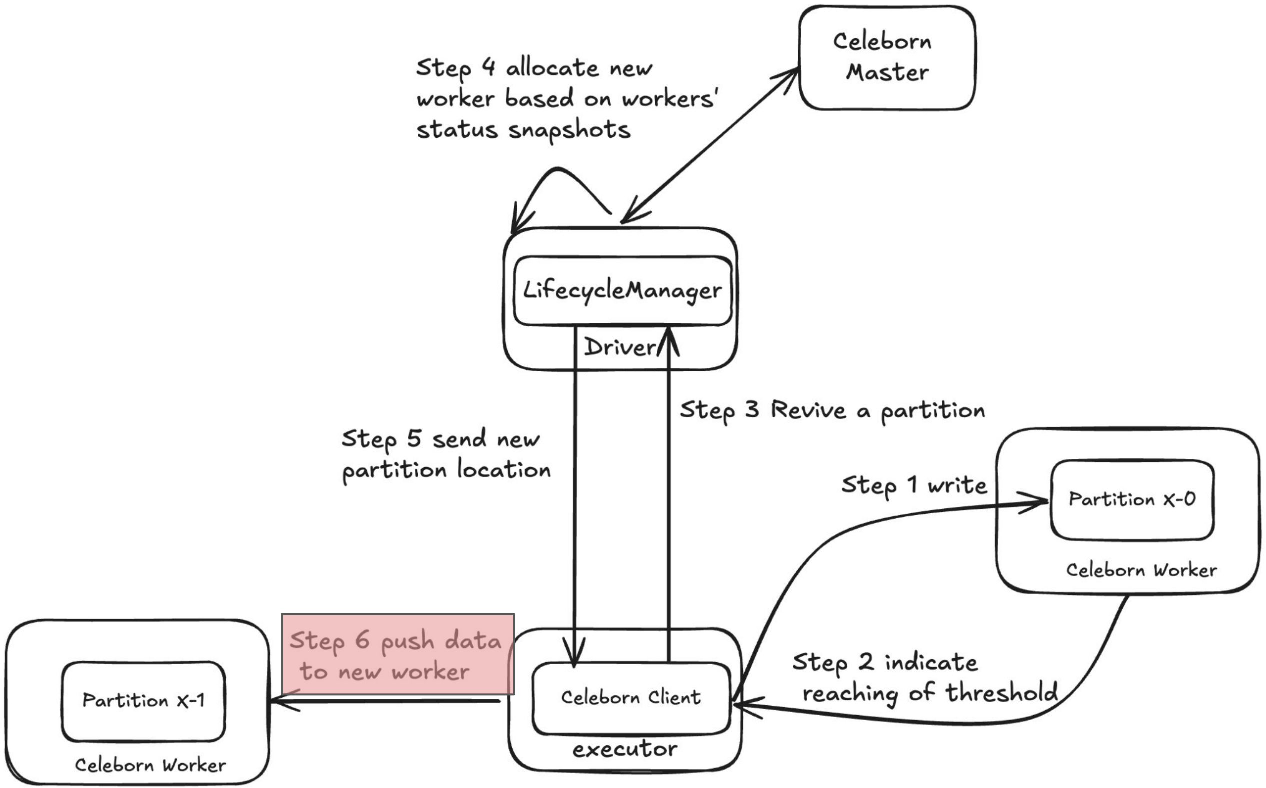
Driver OOM led by PartitionSplit (4)



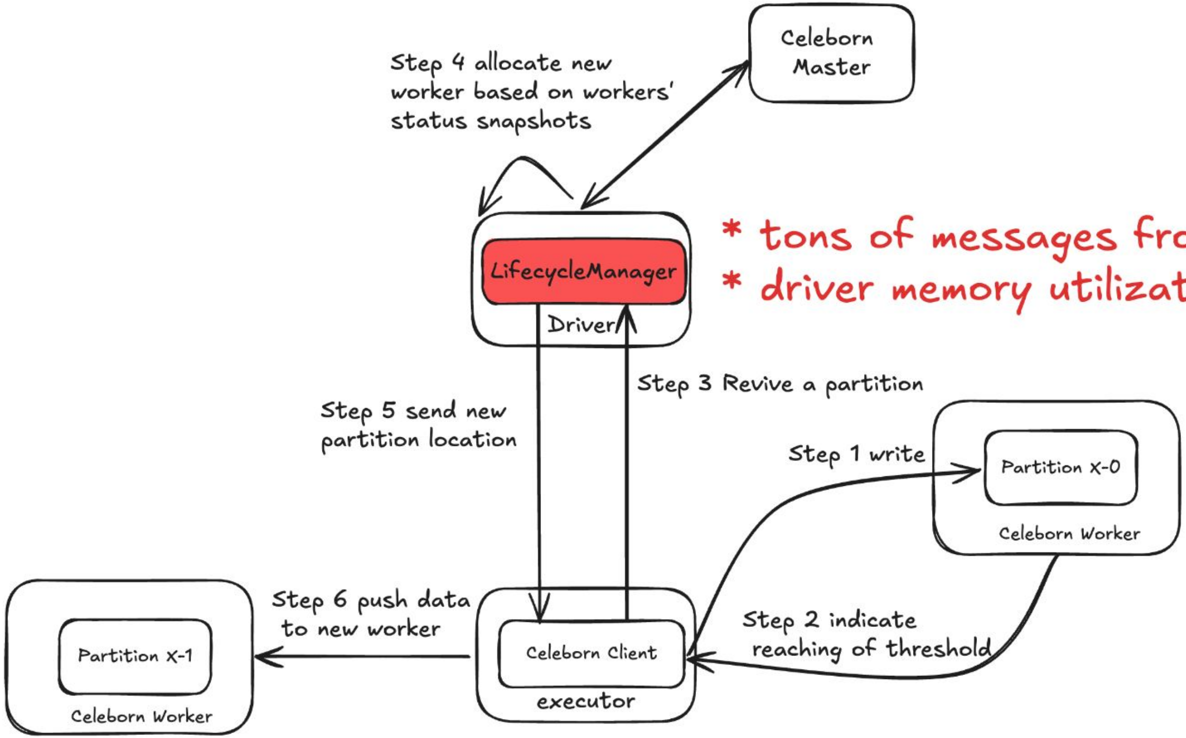
Driver OOM led by PartitionSplit (5)



Driver OOM led by PartitionSplit (6)



Driver OOM led by PartitionSplit (7)



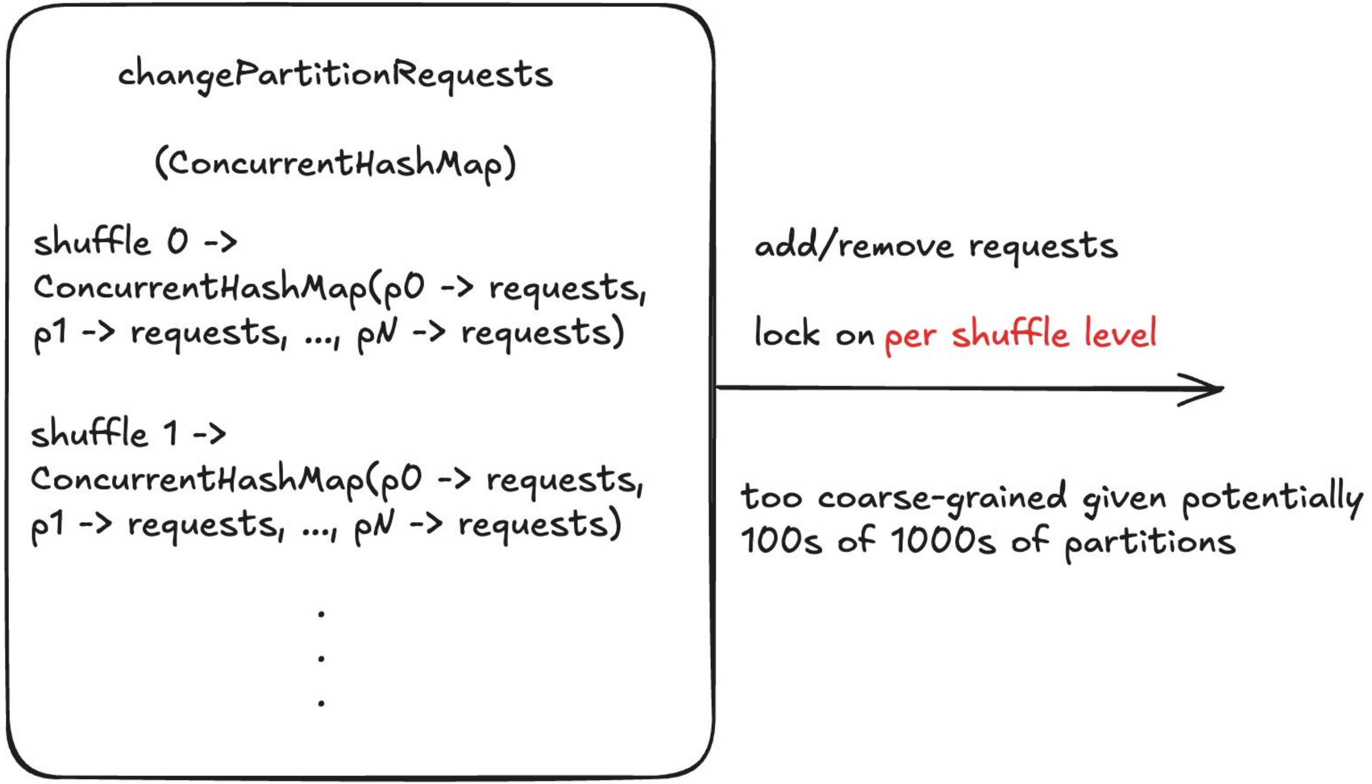
* tons of messages from clients accumulated in LM
* driver memory utilization goes up over time until OOM

Solutions to Driver Memory Pressure (1)

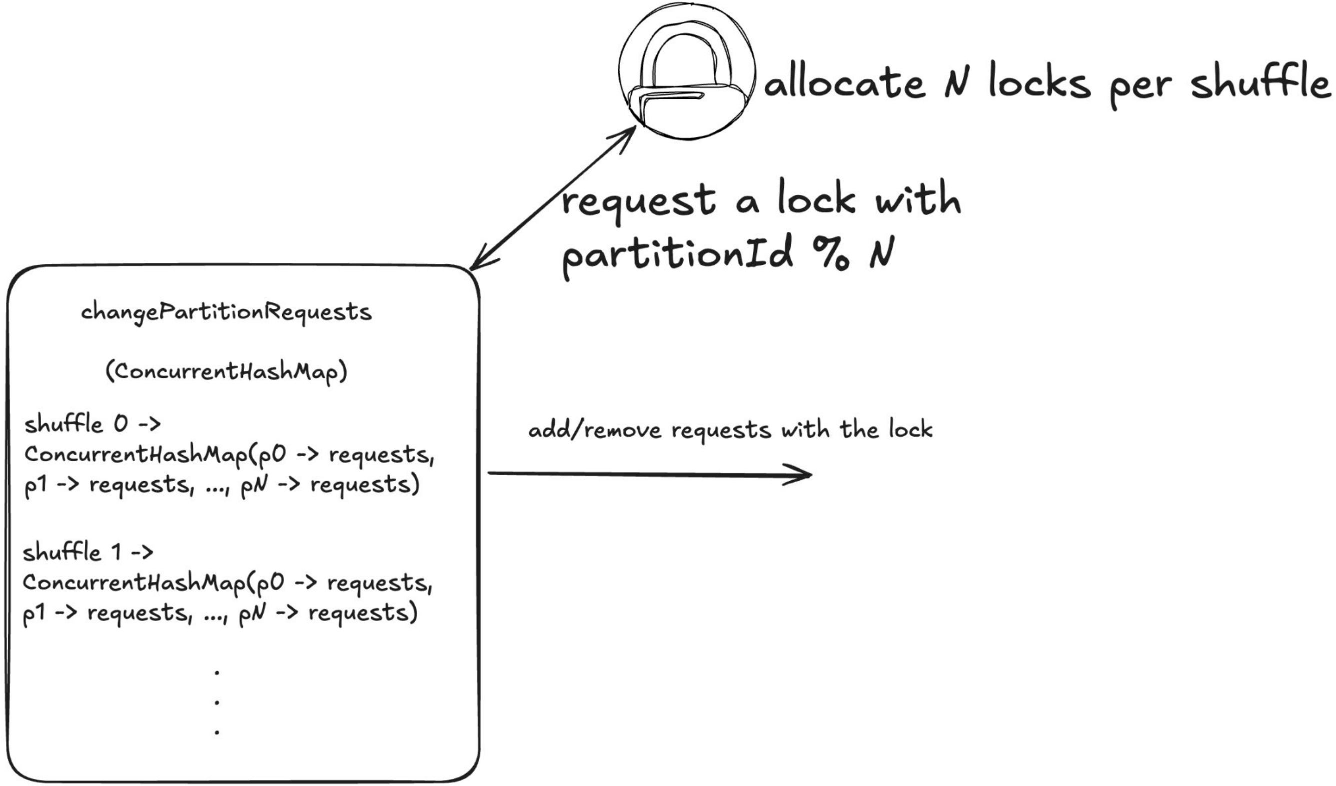
- Reduce the PartitionSplit frequency
  - Increase PartitionSplit threshold to 10G from default value (1G) globally

- Increase the throughput of PartitionSplit message processing
  - Introducing finer-grained locks when changing location of partitions

Coarse-grained lock in change partitions locations



Lock striping in changing partitions locations



Conclusion on Driver OOM issue

- Spark applications with big shuffle size requires more PartitionSplit to balance load among workers
- PartitionSplit is a heavy operation on Driver side, leading to backlogs in RPC queue of Driver => memory pressure
- Solutions
    - Reduce PartitonSplit frequency by increasing PartitionSplit threshold
    - Improve PartitionSplit processing throughput by introducing finer-grained locks

Pinterest - Actively working with Apache Celeborn Community

- Authoring or co-working on Apache Celeborn™ features
  - Adaptive Sort-based Buffer Management
  - Capacity-bounded inbox for RPC endpoint
  - Metrics Enhancement for ActiveSlots
  - Best Effort memory allocation in SortBasedBuffer
  - Fine-grained locks in PartitionSplit handling
  - Fixing OOM due to the concurrent connections in ShuffleReader
  - etc.

Summary & Future Work

- Apache Celeborn™ - Key solution to some most critical issues in running Spark at massive scale
    - Noisy neighbours - **resource reserving mechanism**
    - Slow shuffle read - **simplify M * N topology to N connections**
    - Dynamic allocation support in K8S - **manage shuffle data with a dedicated cluster**
- Future work
    - Optimizing shuffle efficiency further to reduce the cost of Spark™ applications
    - Handling jobs with PB level shuffle

Thank You!