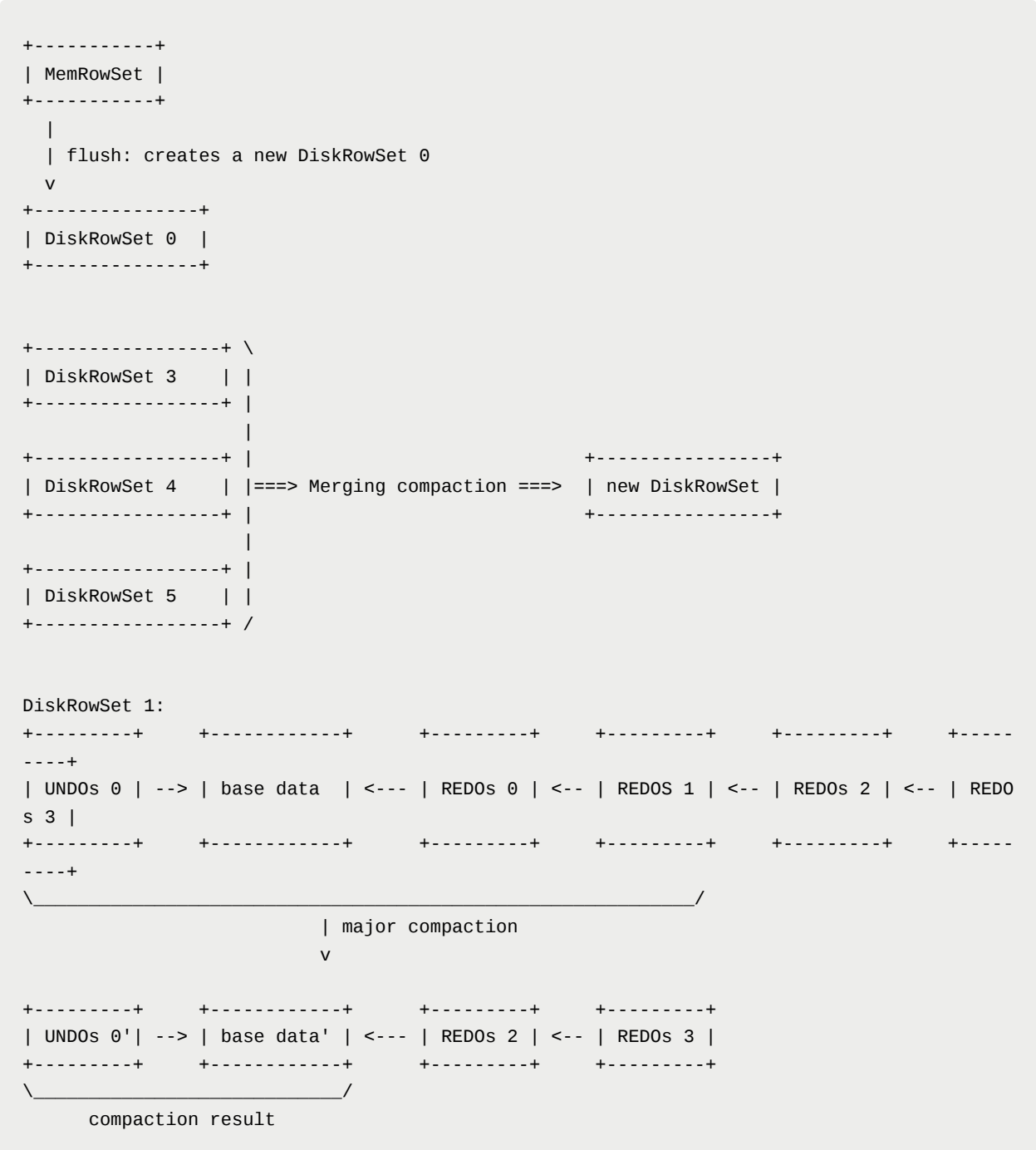
Tune Kudu MemRowSet Flush

What's in a Kudu tablet

https://github.com/apache/kudu/blob/master/docs/design-docs/tablet.md



```
DiskRowSet 2:
+---------+     +------------+      +---------+     +---------+     +---------+     +-----
----+
| UNDOs 0 | --> | base data  | <--- | REDOs 0 | <-- | REDOS 1 | <-- | REDOs 2 | <-- | REDO
s 3 |
+---------+     +------------+      +---------+     +---------+     +---------+     +-----
----+
                                    \_____________________________/
                                         | minor compaction
                                         v
+---------+     +------------+      +---------+      +---------+     +---------+
| UNDOs 0 | --> | base data  | <--- | REDOS 0'|  <-- | REDOs 2 | <-- | REDOs 3 |
+---------+     +------------+      +---------+      +---------+     +---------+
                                    \_________/
                                compaction result
```

Insert Cost

https://github.com/apache/kudu/blob/master/docs/design-docs/compaction-policy.md

In order to Insert, each of the rowsets must be checked for a duplicate key. By storing the rowset ranges in an interval tree, we can efficiently determine the set of rowsets whose intervals may contain the key to be inserted, and thus the cost is linear in that number of rowsets:

```
Let n = the Height of the tablet at the given key
Let B = the bloom filter false positive rate
Let C_bf = cost of bloom filter check
Let C_pk = cost of a primary key lookup
Cost = n*C_bf + n*B*C_pk
Cost = n(C_bf + B*C_pk)
```

```
       A (50MB)
|-------------------|
       B (50MB)
|-------------------|
       C (50MB)
|-------------------|
```

(worst) vs. (good)

```
|-A-||-B-||-C-||-D-|
|--------E---------|



   A       B     C
|------||------||--|
```

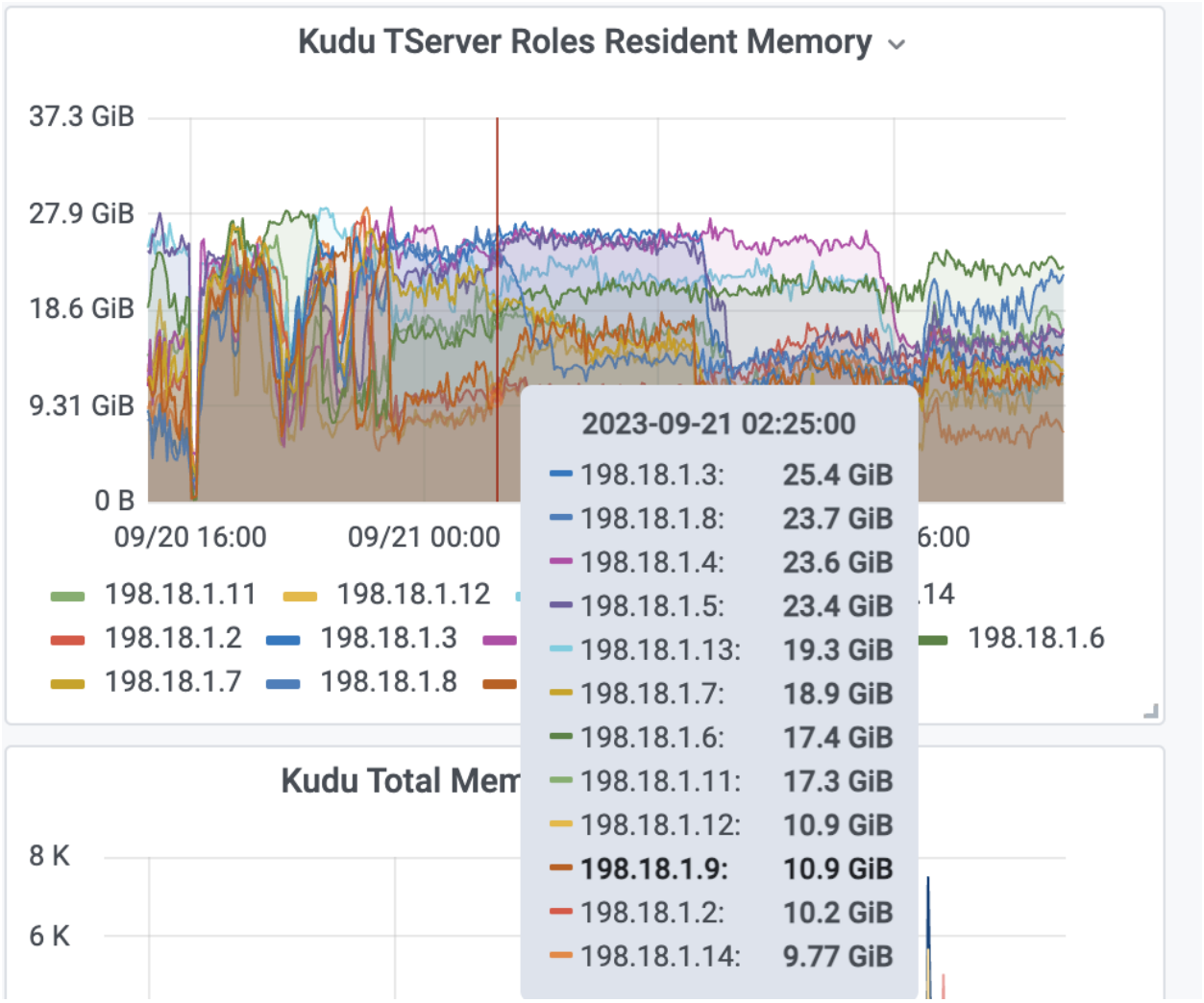
What we did so far

1. Reduce columns → reduce footprint in MemRowSet
2. Change PK ordering to be monotonically increasing → reduce the insert cost within a single RowSet
3. Reduce flush_threshold_mb (512) and increase maintenance threads (16) → trade off some CPU for faster MemRowSet flushing to disk to relieve mem pressure

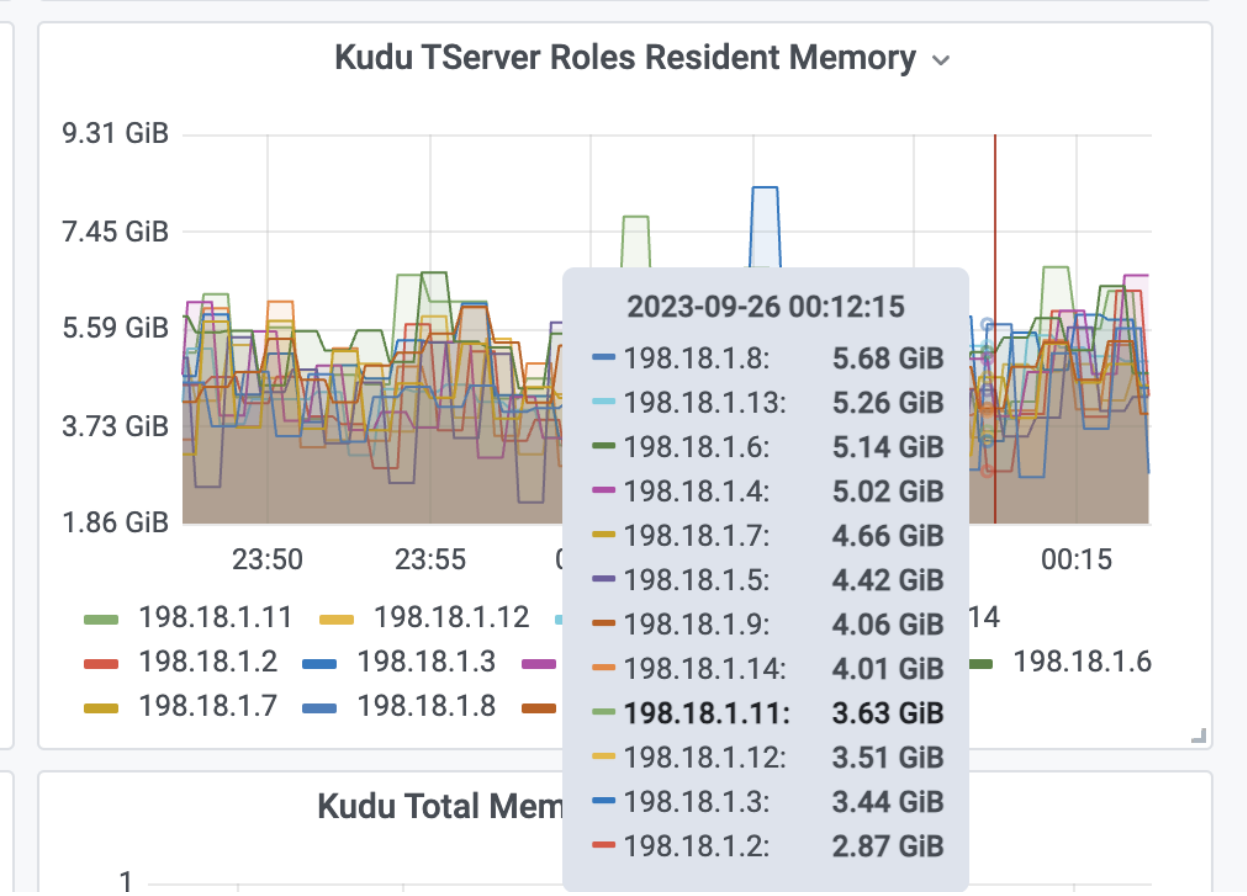
Results

- **MemRowSet usage**

Before

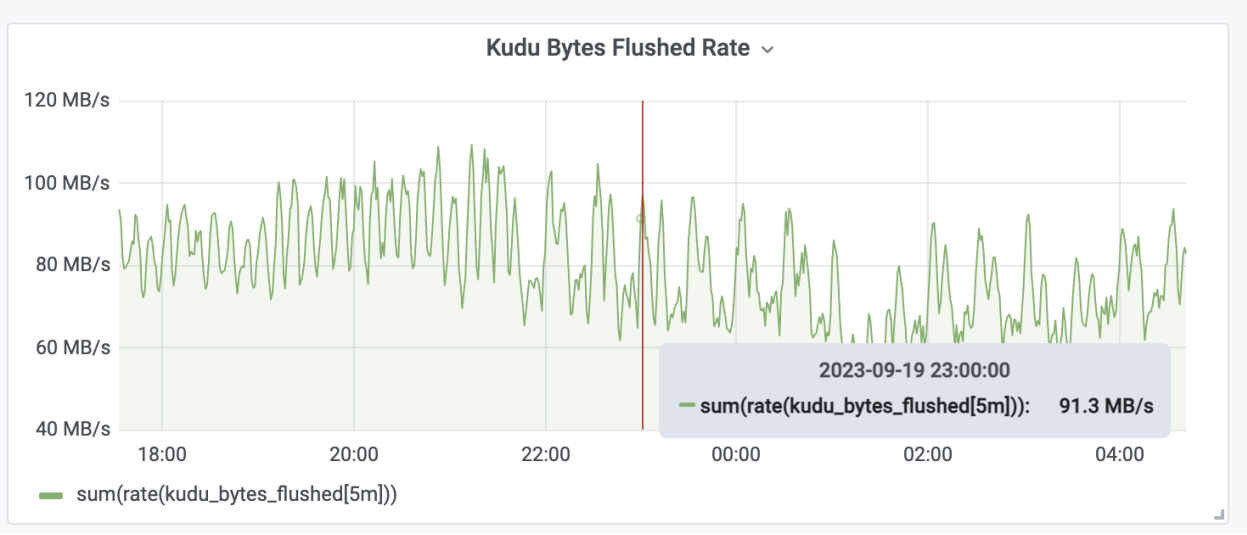

Kudu TServer Roles Resident Memory
37.3 GiB
27.9 GiB
18.6 GiB
9.31 GiB
0 B
09/20 16:00
09/21 00:00
198.18.1.11
198.18.1.12
198.18.1.2
198.18.1.3
198.18.1.7
198.18.1.8
198.18.1.6
2023-09-21 02:25:00
198.18.1.3: 25.4 GiB
198.18.1.8: 23.7 GiB
198.18.1.4: 23.6 GiB
198.18.1.5: 23.4 GiB
198.18.1.13: 19.3 GiB
198.18.1.7: 18.9 GiB
198.18.1.6: 17.4 GiB
198.18.1.11: 17.3 GiB
198.18.1.12: 10.9 GiB
198.18.1.9: 10.9 GiB
198.18.1.2: 10.2 GiB
198.18.1.14: 9.77 GiB
8 K
6 K

After

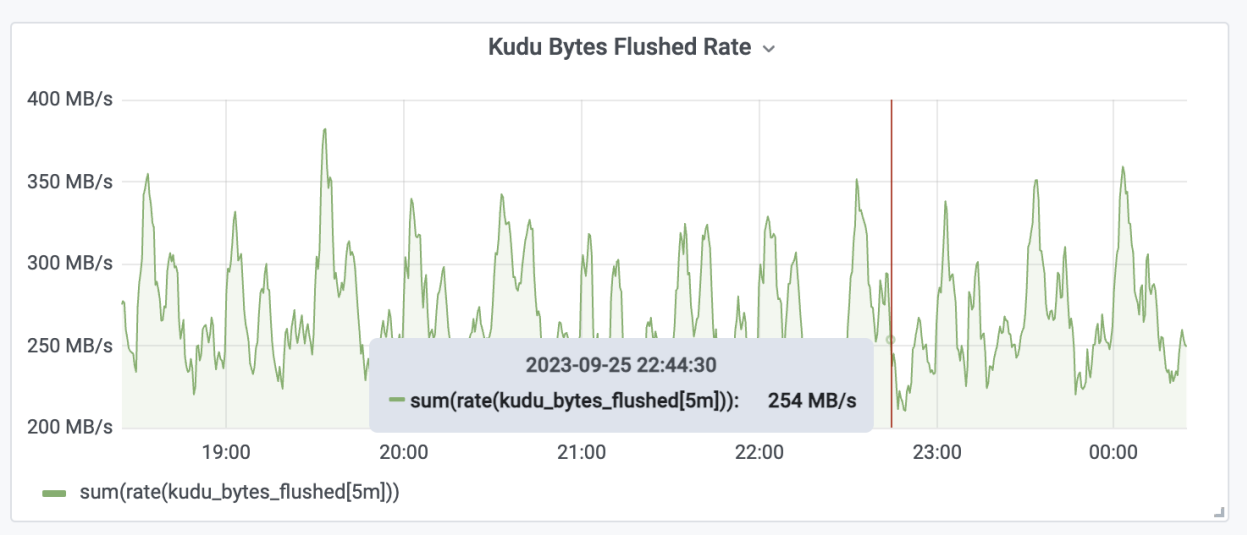


- **Flush Rate**

Before

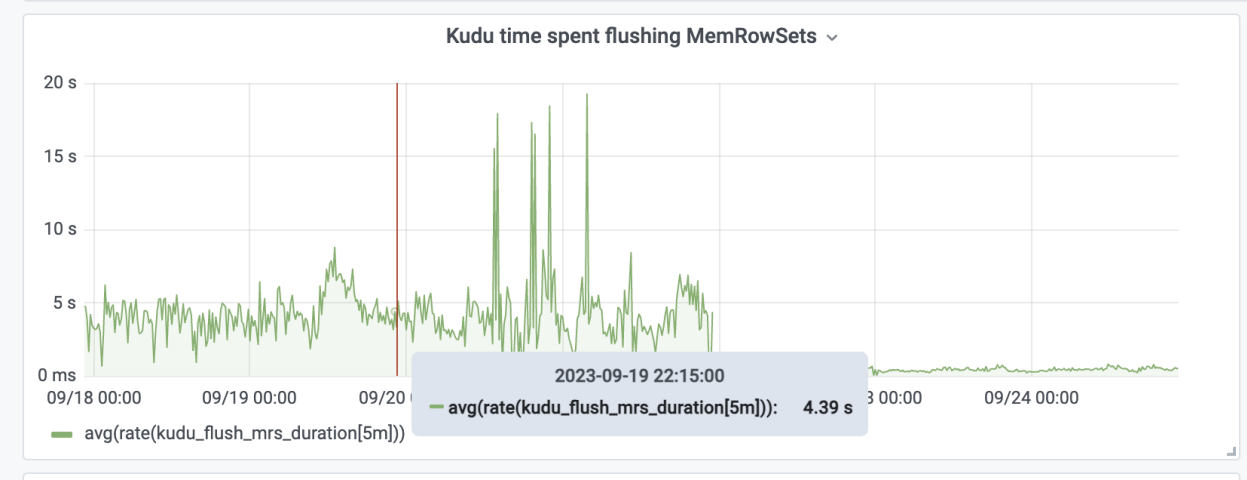


After

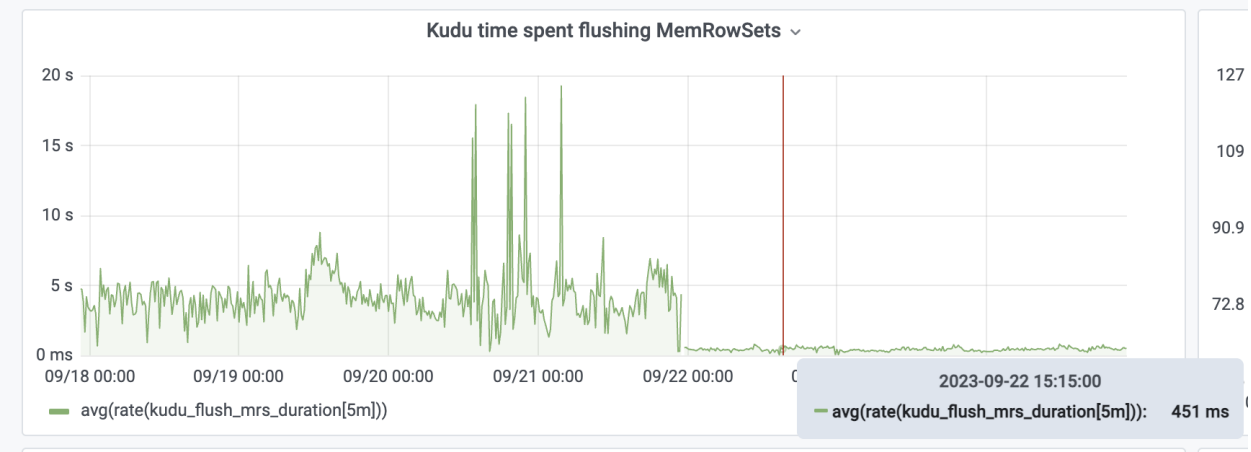


- **Flush Time Spent**

Before

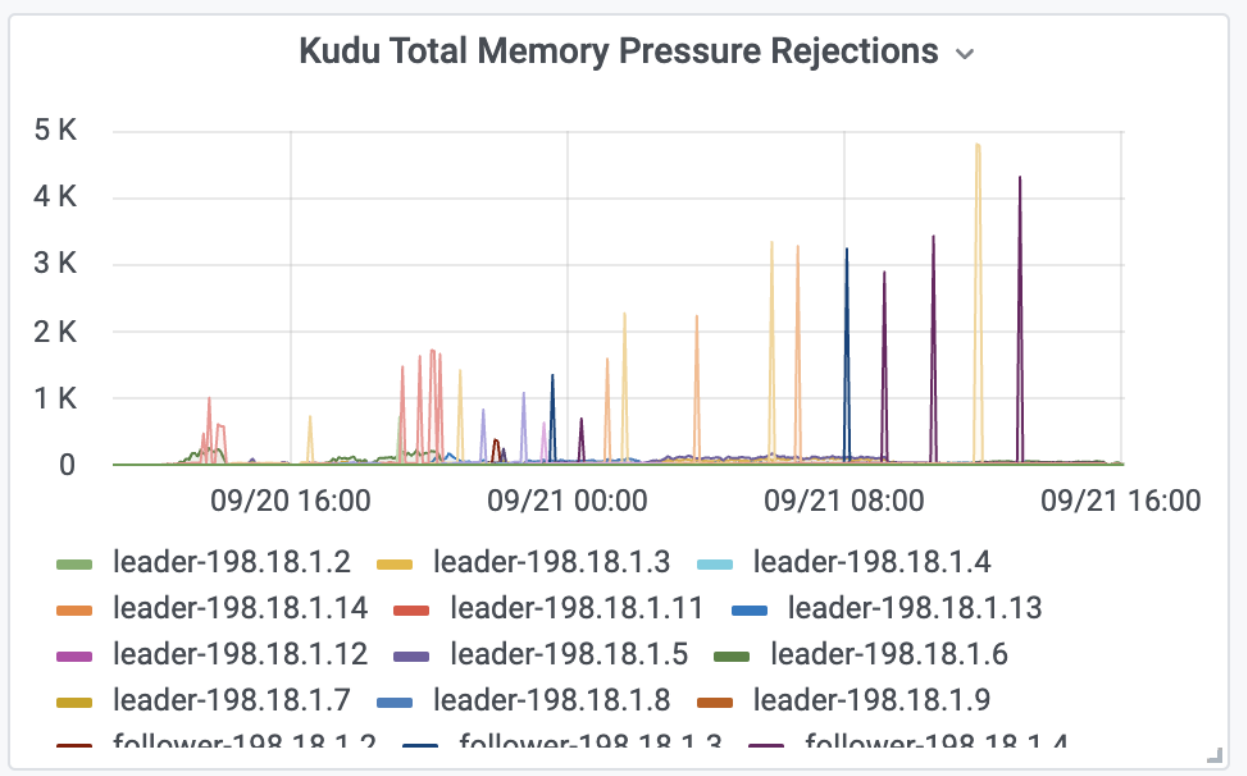


After

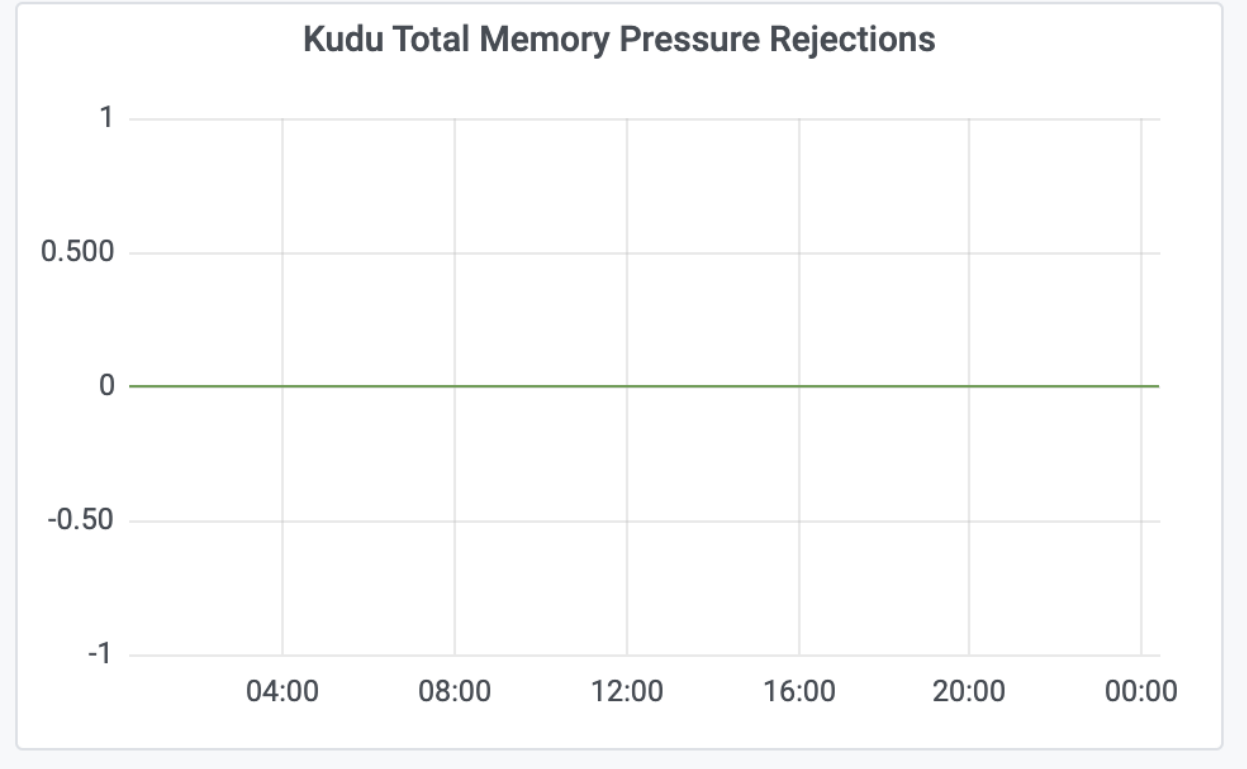


- **Mem Rejection**

Before



After



Further tuning is still possible

1. Add more hash bucket
2. Add more mem for query cache:
    a. `--log_cache_size_limit_mb`
    b. `--global_log_cache_size_limit_mb`
    c. `--block_cache_capacity_mb`
3. Column block size: `--cfile_default_block_size`
4. Log segment size: `--log_segment_size_mb`
5. Compression & encoding: `--cfile_default_compression_codec`

No good solution yet

1. Because of the mixed FV/report/on-demand workload and using start_time as the range partition, uniform insert is inevitable → it may still cause a lot of hot partitions because writes are not hitting a continuous range of keys.