Riffle: Optimized Shuffle Service for Large-Scale Data Analytics

Haoyu Zhang
Brian Cho
Ergin Seyfe
Avery Ching
Michael J. Freedman
Batch analytics systems are widely used

• Large-scale SQL queries
• Custom batch jobs
• Pre-/Post-processing for ML

At **Facebook**

10s of PB new data is generated every day for batch processing

100s of TB data is added to be processed by a single job
Batch analytics jobs: logical graph

- **Map**
- **Filter**
- **Join, groupBy**
- **Filter**

**Narrow dependency**

**Wide dependency**
Batch analytics jobs: DAG execution plan

- Shuffle: all-to-all communication between stages
- >10x larger than available memory, strong fault tolerance requirements
  → on-disk shuffle files
The case for tiny tasks

- Benefits of slicing jobs into small tasks
  - Improve parallelism [Tinytasks HotOS 13] [Subsampling IC2E 14] [Monotask SOSP 17]
  - Improve load balancing [Sparrow SOSP 13]
  - Reduce straggler effect [Dolly NSDI 13] [SparkPerf NSDI 15]
The case against tiny tasks

Although we were able to run the Spark job with such a high number of tasks, we found that there is significant performance degradation when the number of tasks is too high.

- Engineering experience often argues against running too many tasks
  - Medium scale → very large scale (10x larger than memory space)
  - Single-stage jobs → multi-stage jobs (> 50%)

[*] Apache Spark @Scale: A 60 TB+ Production Use Case. https://tinyurl.com/yadx29gl
Shuffle I/O grows quadratically with data

- Large amount of fragmented I/O requests
  - Adversarial workload for hard drives!
Strawman: tune number of tasks in a job

• Tasks spill intermediate data to disk if data splits exceed memory capacity
• Larger task execution reduces shuffle I/O, but increases spill I/O
Strawman: tune number of tasks in a job

- Need to retune when input data volume changes for each individual job
- Bulky tasks can be detrimental [Dolly NSDI 13] [SparkPerf NSDI 15] [Monotask SOSP 17]
  - Straggler problems, imbalanced workload, garbage collection overhead
Small Tasks

Large Amount of Fragmented Shuffle I/O

Bulky Tasks

Fewer, Sequential Shuffle I/O
Rifle: optimized shuffle service

• Rifle shuffle service: a long running instance on each physical node
• Rifle scheduler: keeps track of shuffle files and issues merge requests
Riffle: optimized shuffle service

- When receiving a merge request
  1. Combines small shuffle files into larger ones
  2. Keeps original file layout
- Reducers fetch fewer, large blocks instead of many, small blocks
Results with merge operations on synthetic workload

- Riffle reduces number of fetch requests by 10x
- Reduce stage -393s, map stage +169s → job completes 35% faster
Best-effort merge: mixing merged and unmerged files

- Reduce stage -393s, map stage +52s → job completes 53% faster
  - Riffle finishes job with only ~50% of cluster resources!
Additional enhancements

• Handling merge operation failures
• Efficient memory management
• Balance merge requests in clusters
Experiment setup

- Testbed: Spark on a 100-node cluster
  - 56 CPU cores, 256GB RAM, 10Gbps Ethernet links
  - Each node runs 14 executors, each with 4 cores, 14GB RAM

- Workload: 4 representative production jobs at Facebook

<table>
<thead>
<tr>
<th>Data</th>
<th>Map</th>
<th>Reduce</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  167.6 GB</td>
<td>915</td>
<td>200</td>
<td>983 K</td>
</tr>
<tr>
<td>2  1.15 TB</td>
<td>7,040</td>
<td>1,438</td>
<td>120 K</td>
</tr>
<tr>
<td>3  2.7 TB</td>
<td>8,064</td>
<td>2,500</td>
<td>147 K</td>
</tr>
<tr>
<td>4  267 TB</td>
<td>36,145</td>
<td>20,011</td>
<td>360 K</td>
</tr>
</tbody>
</table>
Reduction in shuffle I/O requests

- Riffle reduces # of I/O requests by 5--10x for medium / large scale jobs
Savings in end-to-end job completion time

- Map stage time is almost not affected (with best-effort merge)
- Reduces job completion time by 20--40% for medium / large jobs
Conclusion

• Shuffle I/O becomes scaling bottleneck for multi-stage jobs

• Efficiently schedule merge operations, mitigate merge stragglers

• Riffle is deployed for Facebook’s production jobs processing PBs of data
Thanks!

Haoyu Zhang
haoyuz@cs.princeton.edu
http://www.haoyuzhang.org
Riffle merge policies

Block 1
Block 2
...
Block R

Block 1
Block 2
...
Block R

N files

Block 1
Block 2
...
Block R

Block 1
Block 2
...
Block R

Block 1
Block 2
...
Block R

Block 1
Block 2
...
Block R

Block 1
Block 2
...
Block R

Block 1
Block 2
...
Block R

total average block size > merge threshold
Best-effort merge

• Observation: slowdown in map stage is mostly due to stragglers

• Best-effort merge: mixing merged and unmerged shuffle files
  • When number of finished merge requests is larger than a user specified percentage threshold, stop waiting for more merge results