Apache Spark at Scale: A 60 TB+ production use case

Sital Kedia

Facebook

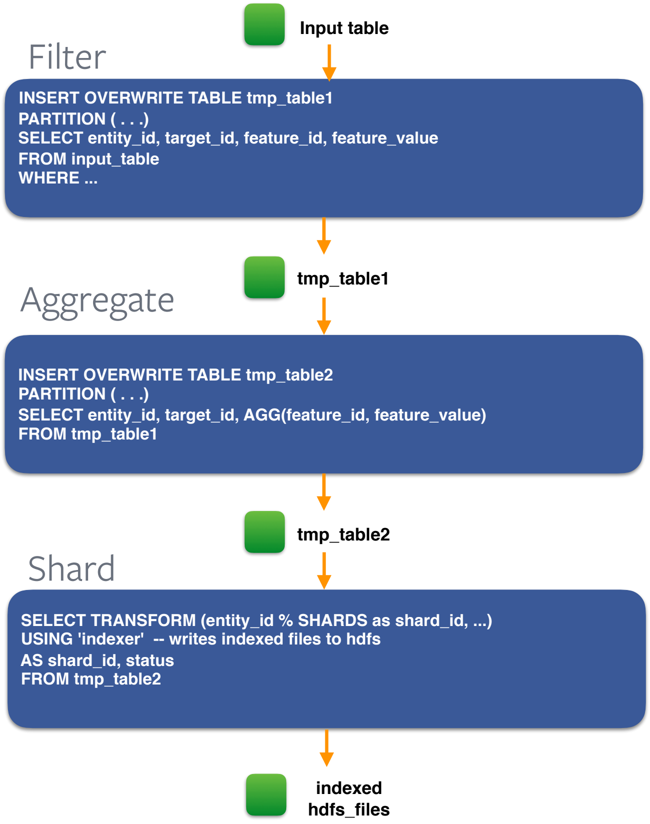
Agenda

- Use case: Entity ranking
- Previous Hive implementation
- Spark implementation
- Performance comparison
- Reliability improvements
- Performance improvements
- Configuration tuning

Use case: Entity ranking

- Used to serve realtime queries to rank entities
- Entity can be users, places, pages etc
- Raw features generated offline using Hive and loaded onto the system for real-time query.

Previous Hive implementation



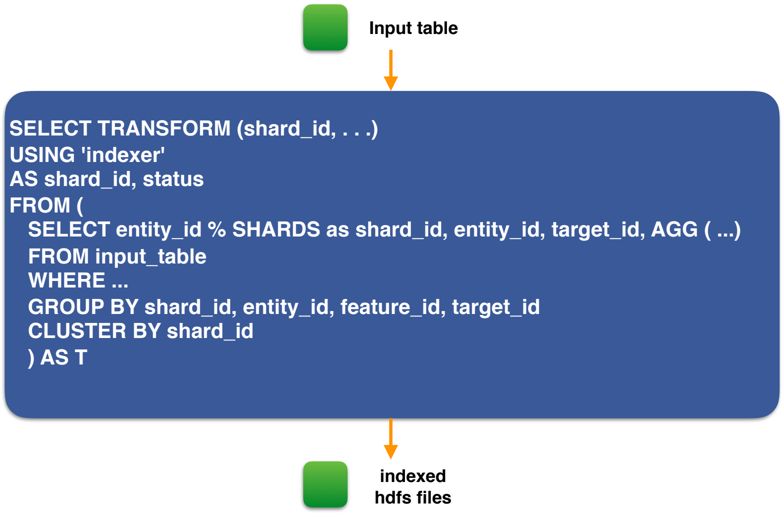
```
INSERT OVERWRITE TABLE tmp_table1
PARTITION ( . . .)
SELECT entity_id, target_id, feature_id, feature_value
FROM input_table
WHERE ...
```

```
INSERT OVERWRITE TABLE tmp_table2
PARTITION ( . . .)
SELECT entity_id, target_id, AGG(feature_id, feature_value)
FROM tmp_table1
```

```
SELECT TRANSFORM (entity_id % SHARDS as shard_id, ...)
USING 'indexer'  -- writes indexed files to hdfs
AS shard_id, status
FROM tmp_table2
```

- 60 TB + **compressed** input data size
- Split into hundreds of smaller hive jobs sharded by entity id
- Unmanageable and slow

Spark implementation

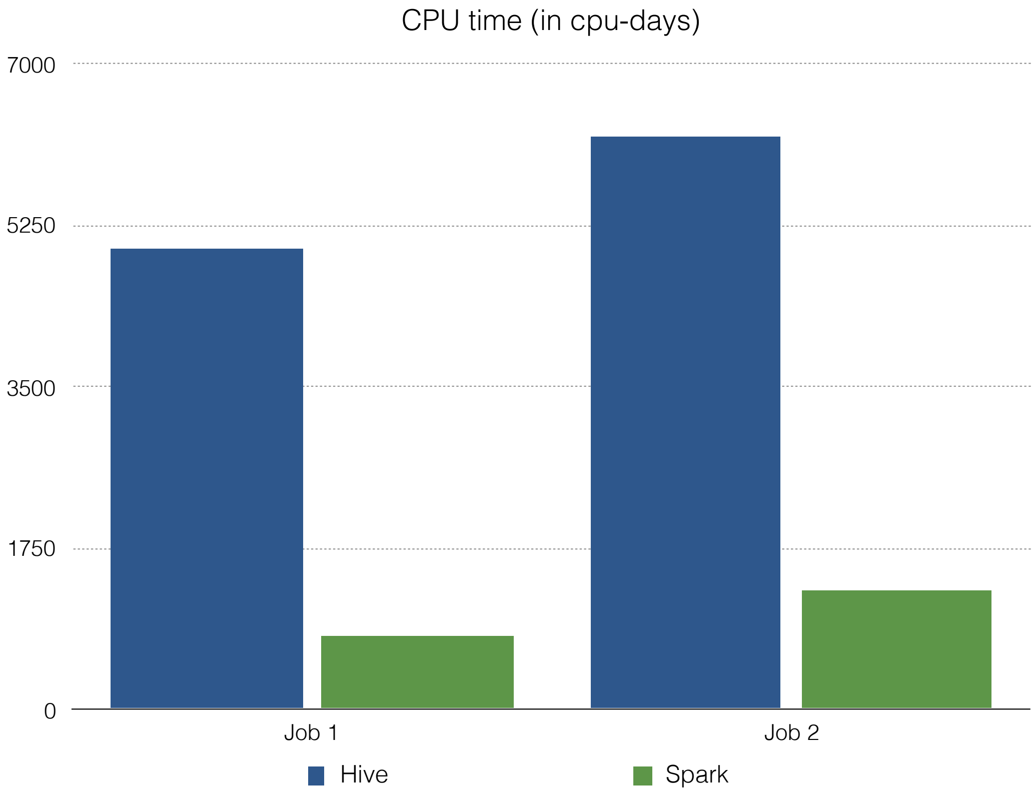


```
SELECT TRANSFORM (shard_id, . . .)
USING 'indexer'
AS shard_id, status
FROM (
   SELECT entity_id % SHARDS as shard_id, entity_id, target_id, AGG ( ...)
   FROM input_table
   WHERE ...
   GROUP BY shard_id, entity_id, feature_id, target_id
   CLUSTER BY shard_id
   ) AS T
```

- Single job with 2 stages
- Shuffles 90 TB+ **compressed** intermediate data

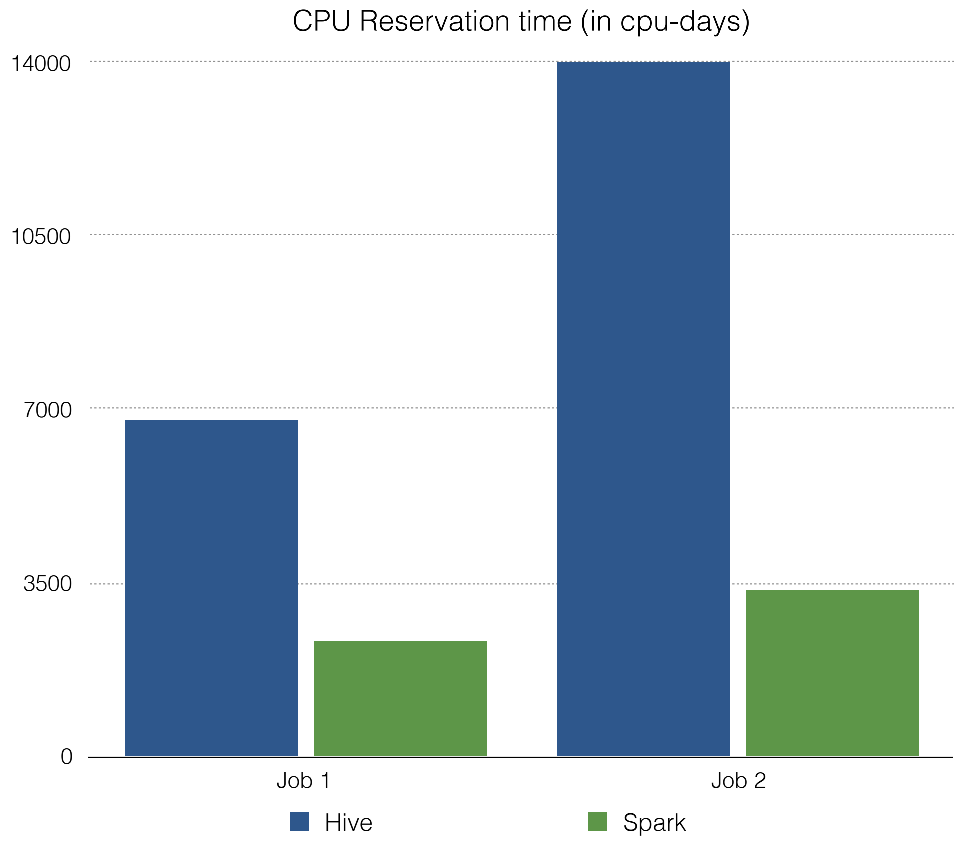
Perfomance comparison

CPU time



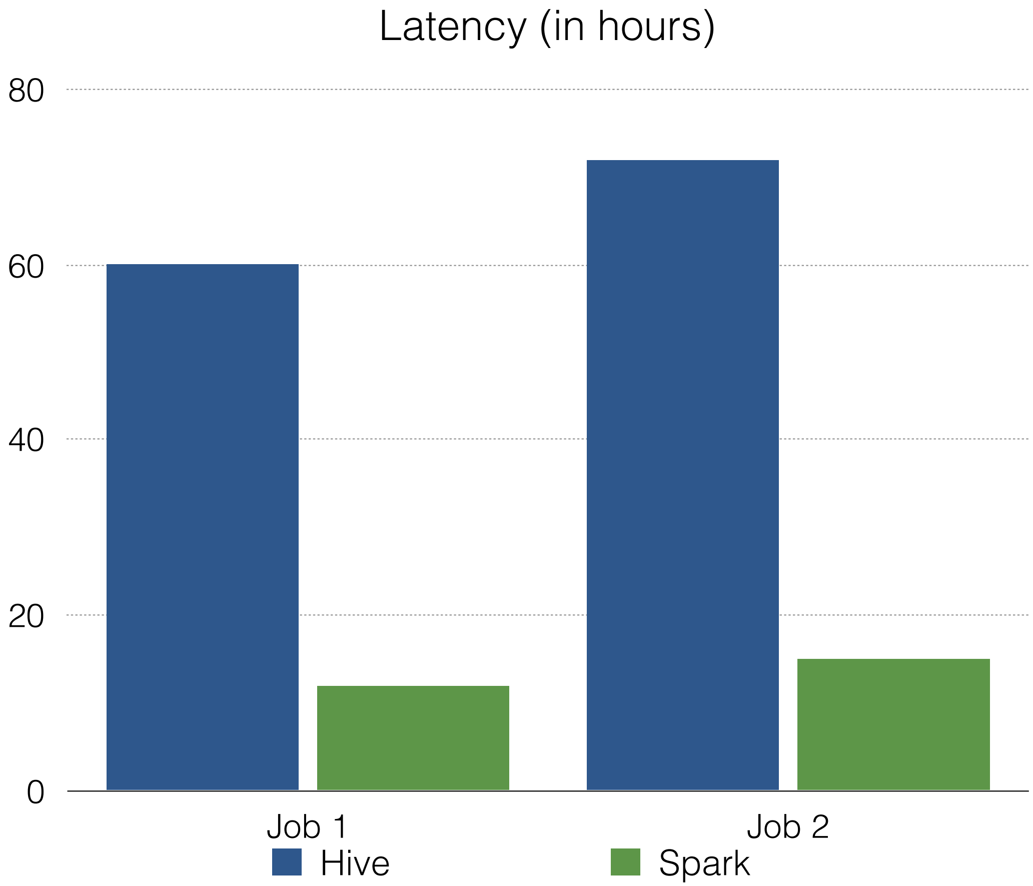
- Collected from OS proc file-system.
- Aggregated across all executors

CPU Reservation time



- `Executor run time * spark.executor.cores`
- Aggregated across all executors

Latency

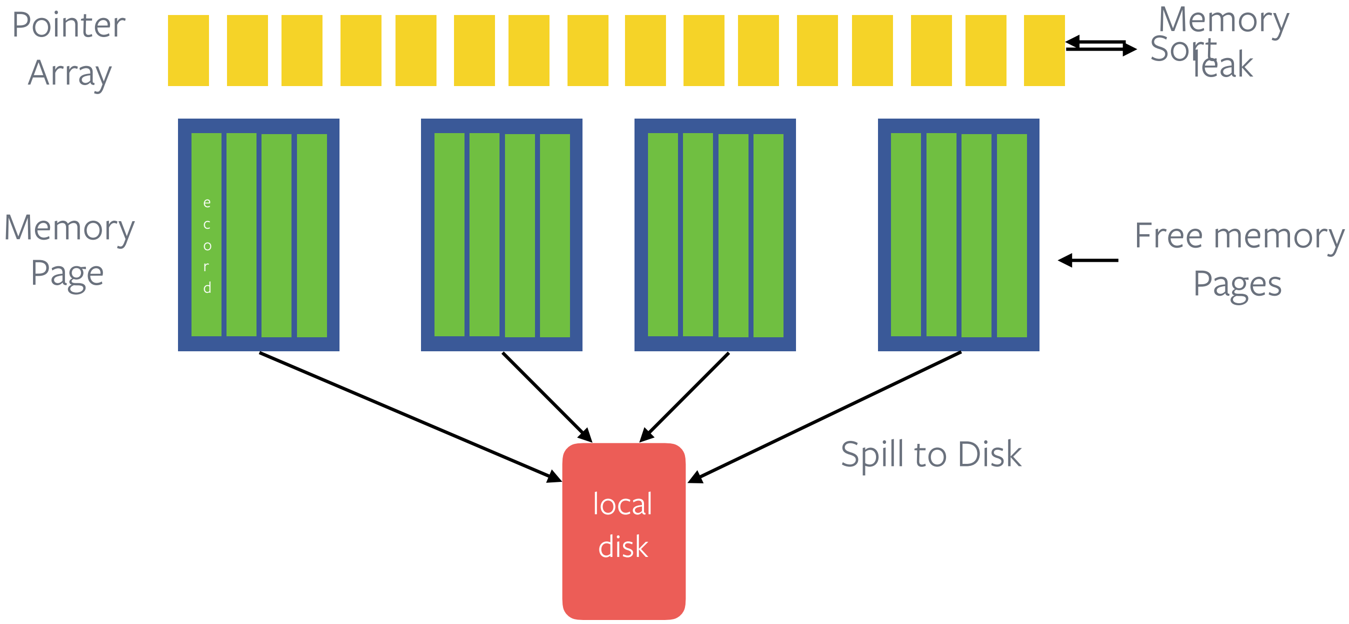


- End to end latency of the job

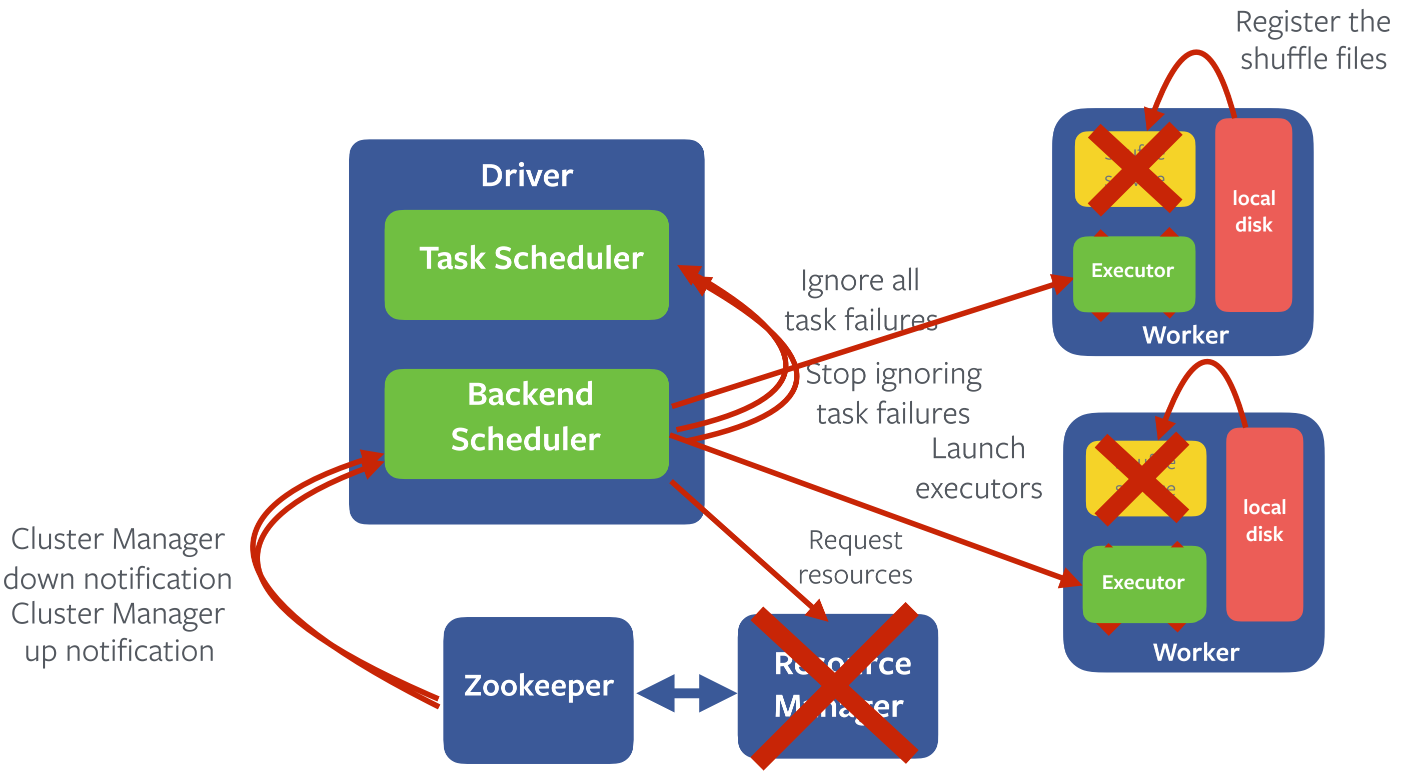
Reliability improvements

Fix memory leak in the sorter

SPARK-14363



Seamless cluster restart



Register the shuffle files

Ignore all task failures

Stop ignoring task failures

Launch executors

Request resources

Cluster Manager down notification

Cluster Manager up notification

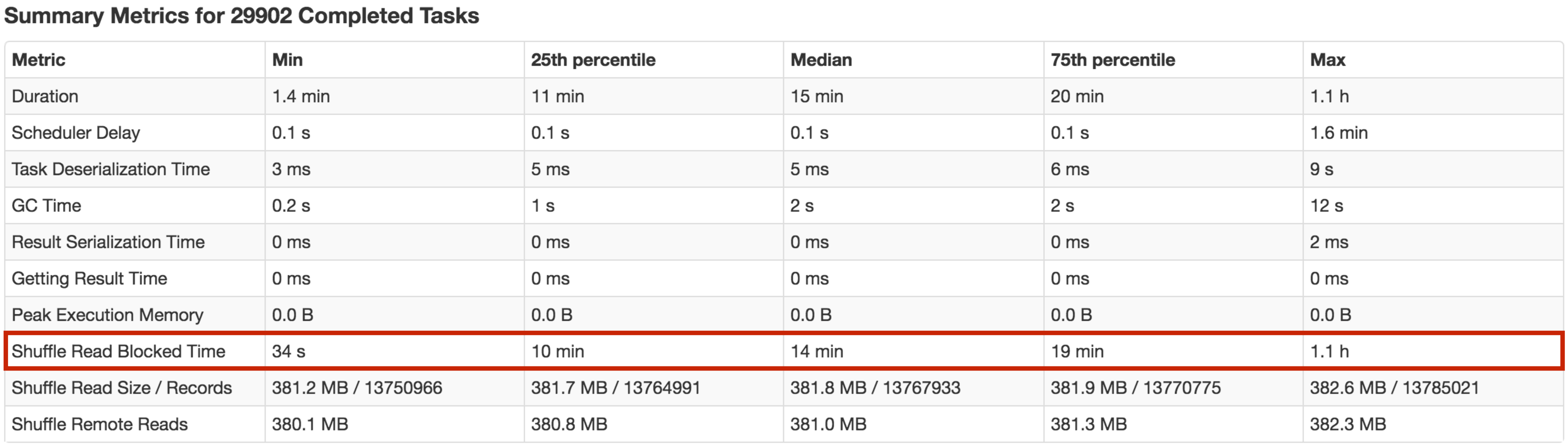
Other reliability improvements

- Various memory leak fixes (SPARK-13958 and SPARK-17113)
- Make PipedRDD robust to fetch failures (SPARK-13793)
- Configurable max number of fetch failures (SPARK-13369)
- Unresponsive driver (SPARK-13279)
- TimSort issue due to integer overflow for large buffer (SPARK-13850)

Performance improvements

Tools

Spark UI metrics

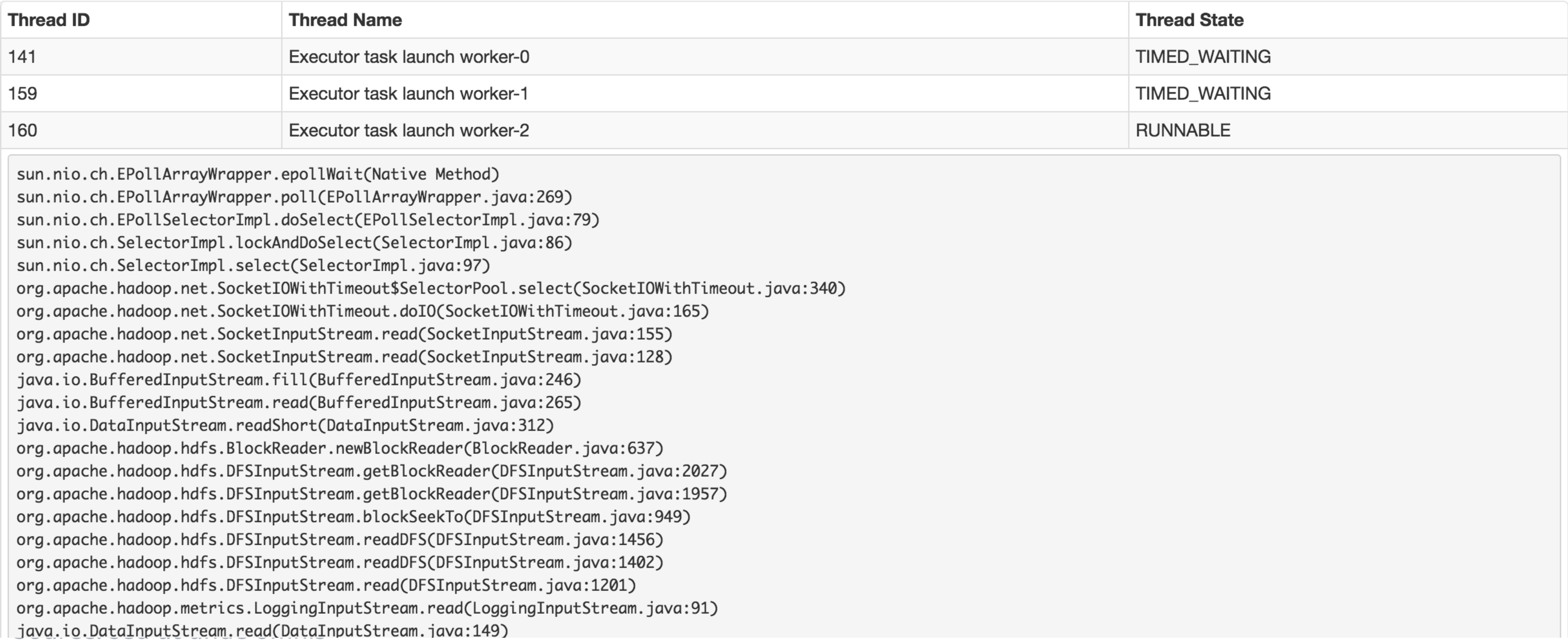


Summary Metrics for 29902 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1.4 min	11 min	15 min	20 min	1.1 h
Scheduler Delay	0.1 s	0.1 s	0.1 s	0.1 s	1.6 min
Task Deserialization Time	3 ms	5 ms	5 ms	6 ms	9 s
GC Time	0.2 s	1 s	2 s	2 s	12 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	2 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	34 s	10 min	14 min	19 min	1.1 h
Shuffle Read Size / Records	381.2 MB / 13750966	381.7 MB / 13764991	381.8 MB / 13767933	381.9 MB / 13770775	382.6 MB / 13785021
Shuffle Remote Reads	380.1 MB	380.8 MB	381.0 MB	381.3 MB	382.3 MB

Tools

Thread dump from Spark UI

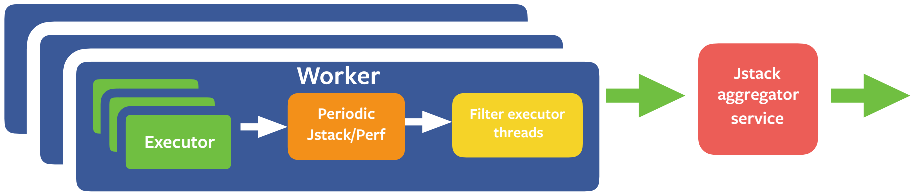


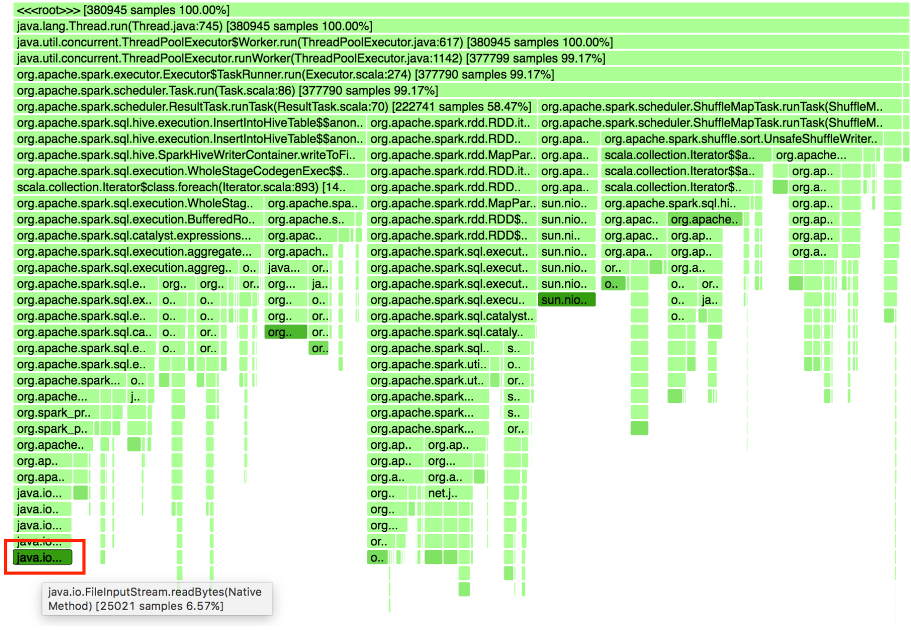
Thread ID	Thread Name	Thread State
141	Executor task launch worker-0	TIMED_WAITING
159	Executor task launch worker-1	TIMED_WAITING
160	Executor task launch worker-2	RUNNABLE

```
sun.nio.ch.EPollArrayWrapper.epollWait(Native Method)
sun.nio.ch.EPollArrayWrapper.poll(EPollArrayWrapper.java:269)
sun.nio.ch.EPollSelectorImpl.doSelect(EPollSelectorImpl.java:79)
sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:86)
sun.nio.ch.SelectorImpl.select(SelectorImpl.java:97)
org.apache.hadoop.net.SocketIOWithTimeout$SelectorPool.select(SocketIOWithTimeout.java:340)
org.apache.hadoop.net.SocketIOWithTimeout.doIO(SocketIOWithTimeout.java:165)
org.apache.hadoop.net.SocketInputStream.read(SocketInputStream.java:155)
org.apache.hadoop.net.SocketInputStream.read(SocketInputStream.java:128)
java.io.BufferedInputStream.fill(BufferedInputStream.java:246)
java.io.BufferedInputStream.read(BufferedInputStream.java:265)
java.io.DataInputStream.readShort(DataInputStream.java:312)
org.apache.hadoop.hdfs.BlockReader.newBlockReader(BlockReader.java:637)
org.apache.hadoop.hdfs.DFSInputStream.getBlockReader(DFSInputStream.java:2027)
org.apache.hadoop.hdfs.DFSInputStream.getBlockReader(DFSInputStream.java:1957)
org.apache.hadoop.hdfs.DFSInputStream.blockSeekTo(DFSInputStream.java:949)
org.apache.hadoop.hdfs.DFSInputStream.readDFS(DFSInputStream.java:1456)
org.apache.hadoop.hdfs.DFSInputStream.readDFS(DFSInputStream.java:1402)
org.apache.hadoop.hdfs.DFSInputStream.read(DFSInputStream.java:1201)
org.apache.hadoop.metrics.LoggingInputStream.read(LoggingInputStream.java:91)
java.io.DataInputStream.read(DataInputStream.java:149)
```

Tools

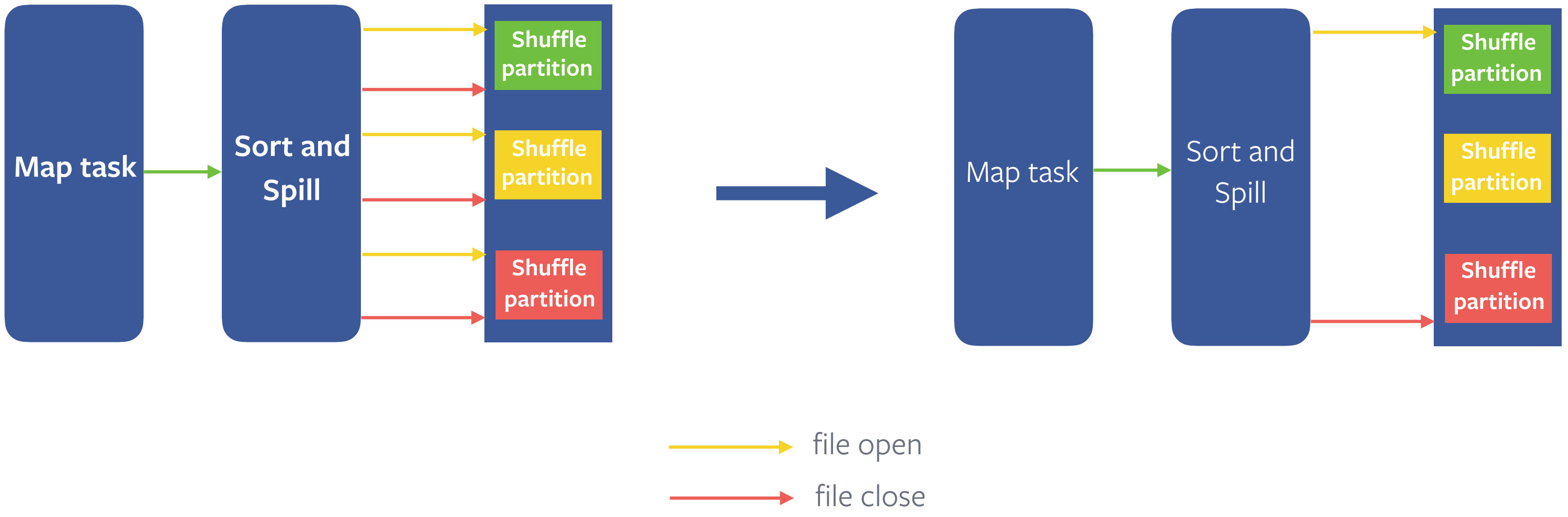
Flame Graph





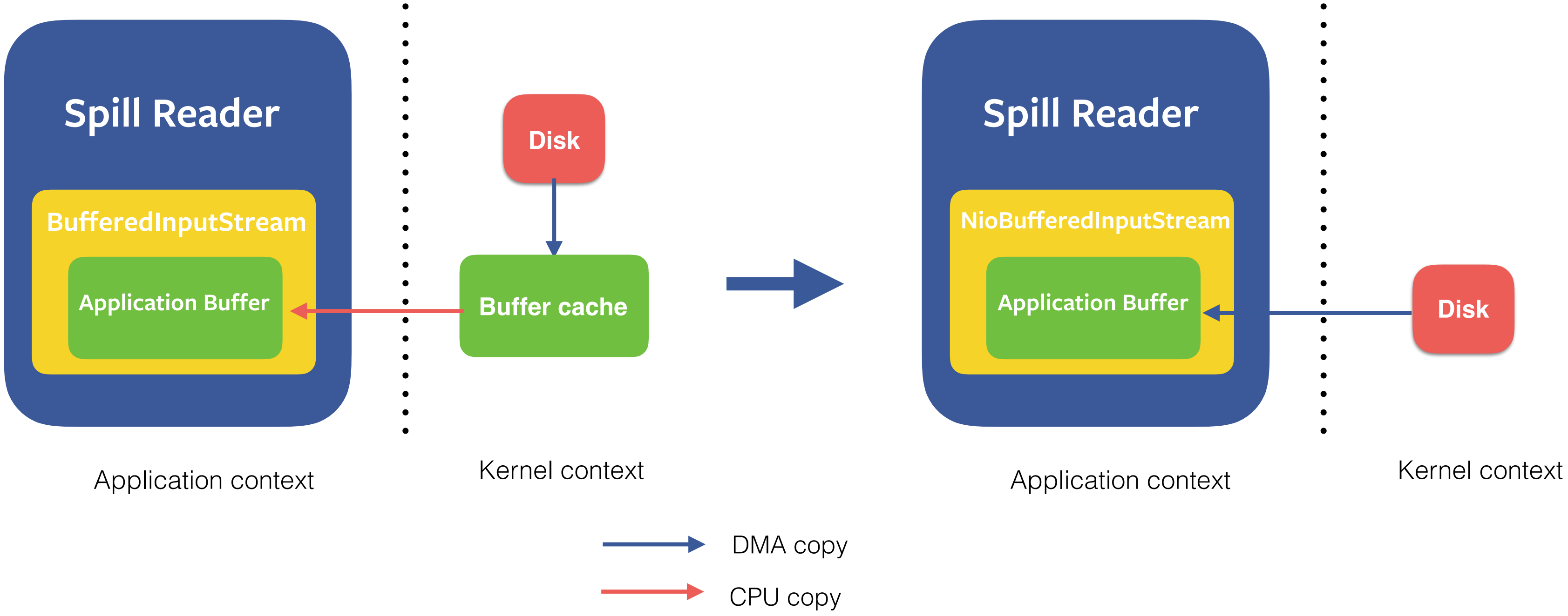
Reduce shuffle write latency

SPARK-5581 (Up to 50% speed-up)



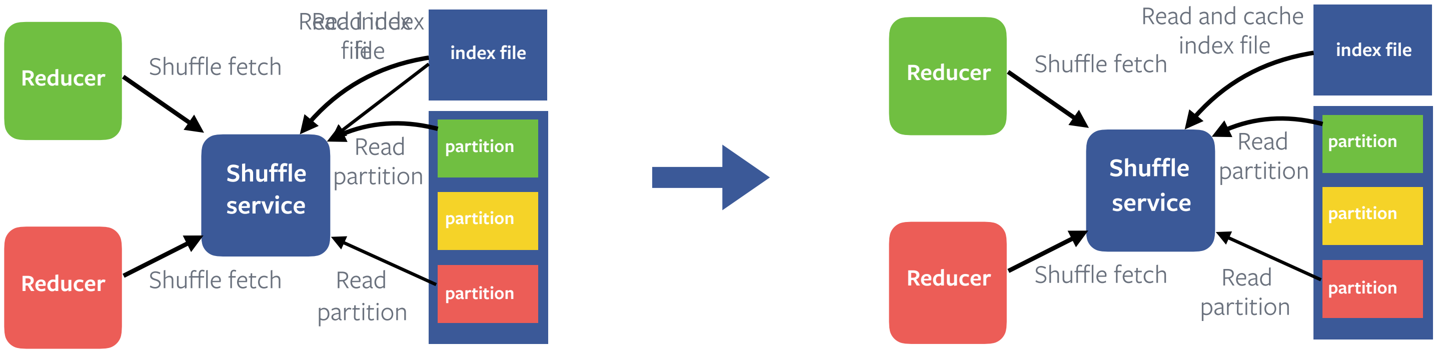
Zero copy based Spill file reader

SPARK-17839 (Up to 7% speed-up)



Cache index files on shuffle server

SPARK-15074



Other performance improvements

- Snappy optimization (SPARK-14277)
- Fix duplicate task run issue due to fetch failure (SPARK-14649)
- Configurable buffer size for PipedRDD (SPARK-14542)
- Reduce update frequency of shuffle bytes written metrics (SPARK-15569)
- Configurable initial buffer size for Sorter(SPARK-15958)

Configuration tuning

Configuration tuning

- Memory configurations
  - `spark.memory.offHeap.enabled = true`
  - `spark.executor.memory = 3g`
  - `spark.memory.offHeap.size = 3g`

- Use parallel GC instead of G1GC
  - `spark.executor.extraJavaOptions = -XX:UseParallelGC`

- Enable dynamic executor allocation
  - `spark.dynamicAllocation.enabled = true`

Configuration tuning

- Tune Shuffle service
  - `spark.shuffle.io.serverThreads = 128`
  - `spark.shuffle.io.backLog = 8192`

- Buffer size configurations -
  - `spark.unsafe.sorter.spill.reader.buffer.size = 2m`
  - `spark.shuffle.file.buffer = 1m`
  - `spark.shuffle.sort.initialBufferSize = 4194304`

Resource

- Apache Spark @Scale: A 60 TB+ production use case

Questions?