

Tuning Apache Spark for large-scale workloads

Sital Kedia

Gaoxiang Liu

Facebook

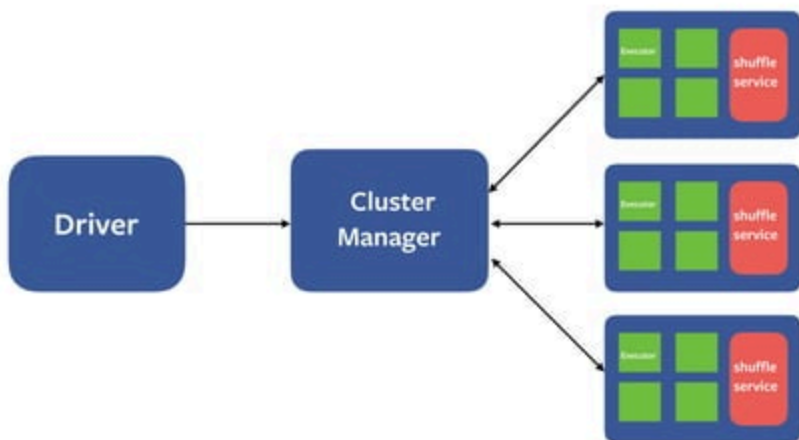
Agenda

- Apache Spark at Facebook
- Scaling Spark Driver
- Scaling Spark Executor
- Scaling External Shuffle
- Application tuning
- Tools

Apache Spark at Facebook

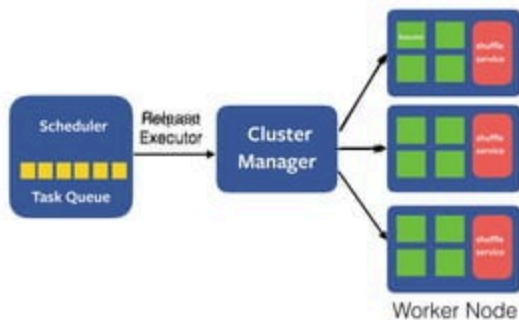
- Used for large scale batch workload
- Tens of thousands of jobs/day and growing
- Running on in the order of thousands of nodes
- Job scalability -
 - Processes hundreds of TBs of compressed input data and shuffle data
 - Runs hundreds of thousands of tasks

Spark Architecture



Scaling Spark Driver

Dynamic Executor Allocation

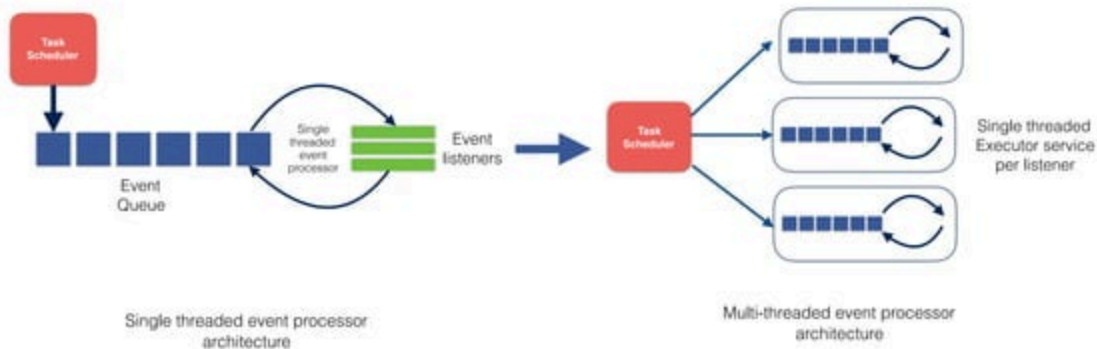


- Better Resource utilization
- Good for multi-tenant environment

```
spark.dynamicAllocation.enabled = true  
spark.dynamicAllocation.executorIdleTimeout = 2m  
spark.dynamicAllocation.minExecutors = 1  
spark.dynamicAllocation.maxExecutors = 2000
```

Multi-threaded Event Processor

[SPARK-18838]



Better Fetch Failure handling

[SPARK-19753] Avoid multiple retries of stages in case of Fetch Failure

Completed Stages (11)

Stage ID	Description	Submitted	Duration	Tasks Succeeded/Total	Input	Out
2 (retry 1)	org.apache.spark.executor.TaskSet	20170519 21:47:11	2.2 s	3/3 [100%]	176.8 GB	
1 (retry 1)	org.apache.spark.executor.TaskSet	20170519 20:26:14	1.8 s	3/3 [100%]	34.9 GB	
3 (retry 1)	org.apache.spark.executor.TaskSet	20170519 20:28:08	27 ms	3/3 [100%]	107.7 GB	
4 (retry 1)	org.apache.spark.executor.TaskSet	20170519 20:28:08	18 ms	3/3 [100%]	49.3 GB	
1 (retry 2)	org.apache.spark.executor.TaskSet	20170519 18:21:03	1.1 s	3/3 [100%]	36.9 GB	
2 (retry 2)	org.apache.spark.executor.TaskSet	20170519 18:18:02	17 ms	3/3 [100%]	123.4 GB	
3 (retry 2)	org.apache.spark.executor.TaskSet	20170519 18:20:08	13 ms	3/3 [100%]	12.4 GB	
4 (retry 2)	org.apache.spark.executor.TaskSet	20170519 18:20:08	8.3 ms	3/3 [100%]	40.3 GB	
5 (retry 2)	org.apache.spark.executor.TaskSet	20170519 17:11:02	1.2 s	3/3 [100%]	52.9 GB	
6 (retry 2)	org.apache.spark.executor.TaskSet	20170519 17:11:03	1.8 s	3/3 [100%]	75.2 GB	

Failed Stages (1)

Stage ID	Description	Submitted	Duration	Tasks Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write	Failure Reason
2	org.apache.spark.executor.TaskSet	20170519 18:08:12	19 ms	0/3 [0%]	1.9 GB		6.8 TB		org.apache.spark.shuffle.ShuffleWriteException

Single fetch failure causing multiple retries

Completed Stages (8)

Stage ID	Description	Submitted	Duration	Tasks Succeeded/Total	Input	Output
3	org.apache.spark.executor.TaskSet	20170519 18:03:03	27 ms	3/3 [100%]		
2 (retry 1)	org.apache.spark.executor.TaskSet	20170519 16:41:02	1.4 s	3/3 [100%]	115.0 GB	
1 (retry 1)	org.apache.spark.executor.TaskSet	20170519 16:38:07	3.3 ms	3/3 [100%]	427.5 GB	
0 (retry 1)	org.apache.spark.executor.TaskSet	20170519 16:38:07	2.1 ms	3/3 [100%]	32.3 GB	
1	org.apache.spark.executor.TaskSet	20170519 14:57:25	23 ms	3/3 [100%]	84.5 GB	
0	org.apache.spark.executor.TaskSet	20170519 14:57:11	51 ms	3/3 [100%]		

Failed Stages (1)

Stage ID	Description	Submitted	Duration	Tasks Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write	Failure Reason
2	org.apache.spark.executor.TaskSet	20170519 15:48:11	45 ms	1/3 [33%]			14.7 TB	2.8 TB	org.apache.spark.shuffle.ShuffleWriteException

Single fetch failure causing single retries

Better Fetch Failure handling

- Avoid duplicate task run in case of Fetch Failure (SPARK-20163)
- Configurable max number of Fetch Failures (SPARK-13369)
`spark.max.fetch.failures.per.stage = 10`
- Ongoing effort (SPARK-20178)

Scaling Spark Driver

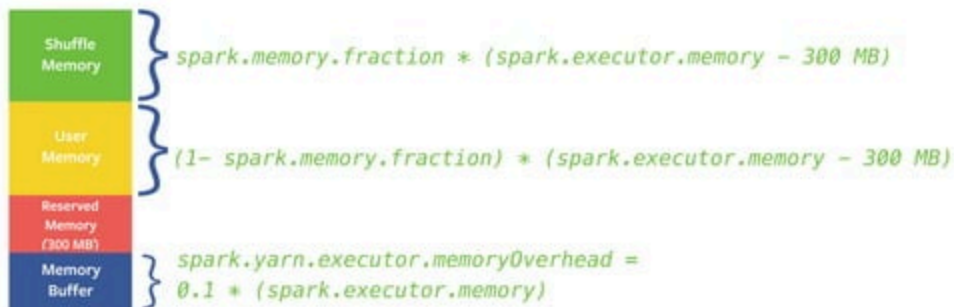
Tune RPC Server threads

- Frequent driver OOM when running many tasks in parallel
- Huge backlog of RPC requests built on Netty server of the driver
- Increase RPC server thread to fix OOM

```
spark.rpc.io.serverThreads = 64
```

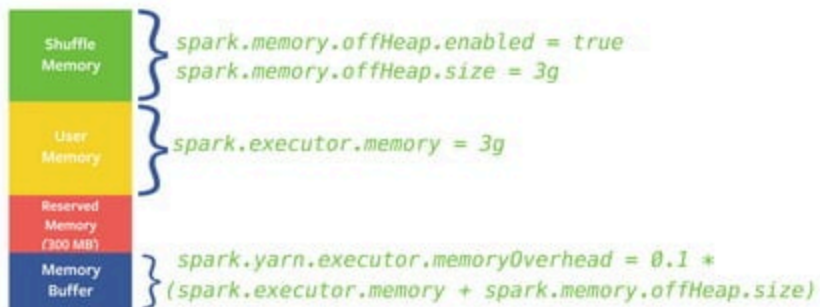
Scaling Spark Executor

Executor memory layout



Tuning memory configurations

Enable off-heap memory



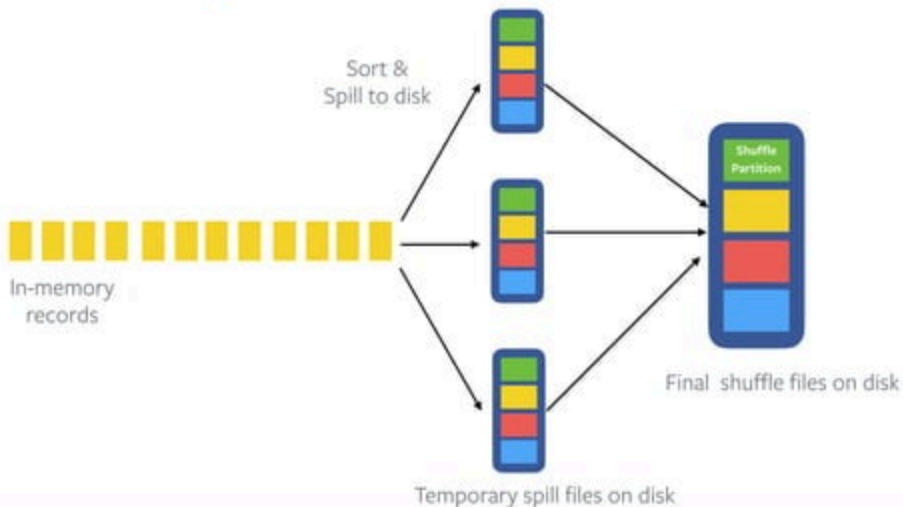
Tuning memory configurations

Garbage collection tuning

- Large contiguous in-memory buffers allocated by Spark's shuffle internals.
- G1GC suffers from fragmentation due to *Humongous Allocations*, if object size is more than 32 MB (Maximum region size of G1GC)
- Use parallel GC instead of G1GC

```
spark.executor.extraJavaOptions = -XX:ParallelGCThreads=4 -XX:+UseParallelGC
```

Eliminating disk I/O bottleneck



Eliminating disk I/O bottleneck

Tune Shuffle file buffer

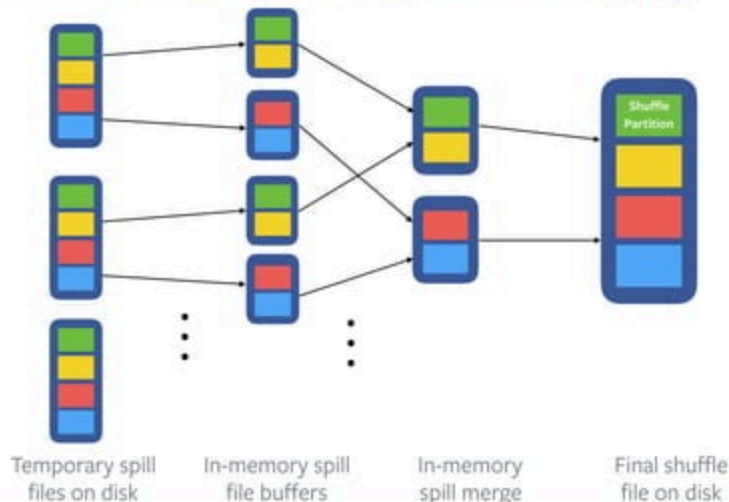
- Disk access is 10 - 100K times slower than memory access
- Make write buffer sizes for disk I/O configurable ([SPARK-20074](#))
- Amortize disk I/O cost by doing buffered read/write

```
spark.shuffle.file.buffer = 1 MB
```

```
spark.unsafe.sorter.spill.reader.buffer.size = 1MB
```


Eliminating disk I/O bottleneck

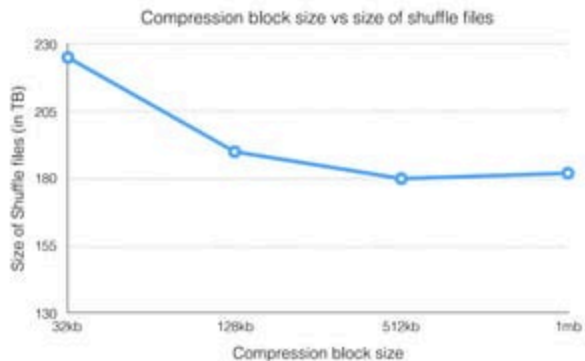
[SPARK-20014] Optimize spill files merging



```
spark.file.transferTo = false  
spark.shuffle.file.buffer = 1 MB  
spark.shuffle.unsafe.file  
.output.buffer = 5 MB
```

Eliminating disk I/O bottleneck

Tune compression block size



- Default compression block size of 32 kb is sub-optimal
- Upto 20% reduction in shuffle/spill file size by increasing the block size

`spark.io.compression.lz4.blockSize = 512KB`

Various Memory leak fixes and improvements

- Memory leak fixes ([SPARK-14363](#), [SPARK-17113](#), [SPARK-18208](#))
- Snappy optimization ([SPARK-14277](#))
- Reduce update frequency of shuffle bytes written metrics ([SPARK-15569](#))
- Configurable initial buffer size for Sorter([SPARK-15958](#))

Scaling External Shuffle Service

Cache Index files on Shuffle Server

SPARK-15074



`spark.shuffle.service.index.cache.entries = 2048`

Scaling External Shuffle Service

- Tune shuffle service worker thread and backlog

```
spark.shuffle.io.serverThreads = 128  
spark.shuffle.io.backLog = 8192
```

- Configurable shuffle registration timeout and retry
([SPARK-20640](#))

```
spark.shuffle.registration.timeout = 2m  
spark.shuffle.registration.maxAttempts = 5
```

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



Sital Kedia



Shuzhe Wang



Avery Ching

Facebook often uses analytics for data-driven decision making. Over the past few years, user and product growth has pushed our analytics engines to operate on data sets in the tens of terabytes for a single query. Some of our batch analytics is executed through the venerable **Hive** platform (contributed to Apache Hive by Facebook in 2009) and **Corona**, our custom MapReduce implementation. Facebook has also continued to grow its Presto footprint for ANSI-SQL queries against several internal data stores, including Hive. We support other types of analytics such as graph processing and machine learning (**Apache Giraph**) and streaming (e.g., **Puma**, **Swift**, and **Stylus**).

While the sum of Facebook's offerings covers a broad spectrum of the analytics space, we continually interact with the open source community in order to share our experiences and also learn from others. **Apache Spark** was started by Matei Zaharia at UC-Berkeley's AMPLab in 2009 and was later contributed to Apache in 2013. It is currently one of the fastest-growing data processing platforms, due to its ability to support streaming, batch, imperative (RDD), declarative (SQL), graph, and machine learning use cases all within the same API and underlying compute engine. Spark can

Related



<https://code.facebook.com/posts/1671373793181703>

Application tuning

Motivation

- Improve performance of job latency (under same amount of resource)
- Improve usability - eliminate manual tuning as much as possible to achieve comparable job performance with manually tuned parameters

Auto tuning of mapper and reducer

- Heuristics-based approach based on table input size

```
1 // number of mappers
2 num of mappers =
3 max(256 MB, inputTableSize / 50000)
4
5 // number of reducers
6 num of reducers = max(
7   200,
8   min(10000, max(inputTableSize / 256 MB * 0.125, 200))
9 )
```

- Max cap due to the constrain of the scalability of shuffle service and drivers
- Min cap due to the minimum guarantee of resource to user's job

Tools

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

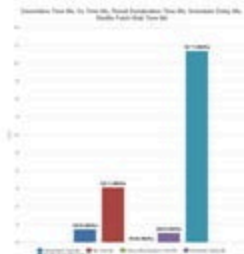
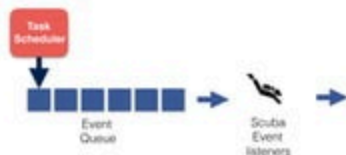
Tools

Flame Graph



Tools

Analysis of Task metrics using Facebook's Scuba



- Enables us to do complex queries like -
 - *How many job failed because of OOM in ExternalSorter in last 1 hour?*
 - *What percentage of total execution time is being spent in shuffle read?*
 - *Did fetch failure rate go up after the last Spark release?*
- Set up monitoring and alerting to catch regression

Resources

- [Scuba: Diving into Data at Facebook](#)
- [Apache Spark @Scale: A 60 TB+ production use case](#)

Questions?