# Tuning Apache Spark for large-scale workloads

**Sital Kedia**

**Gaoxiang Liu**

Facebook

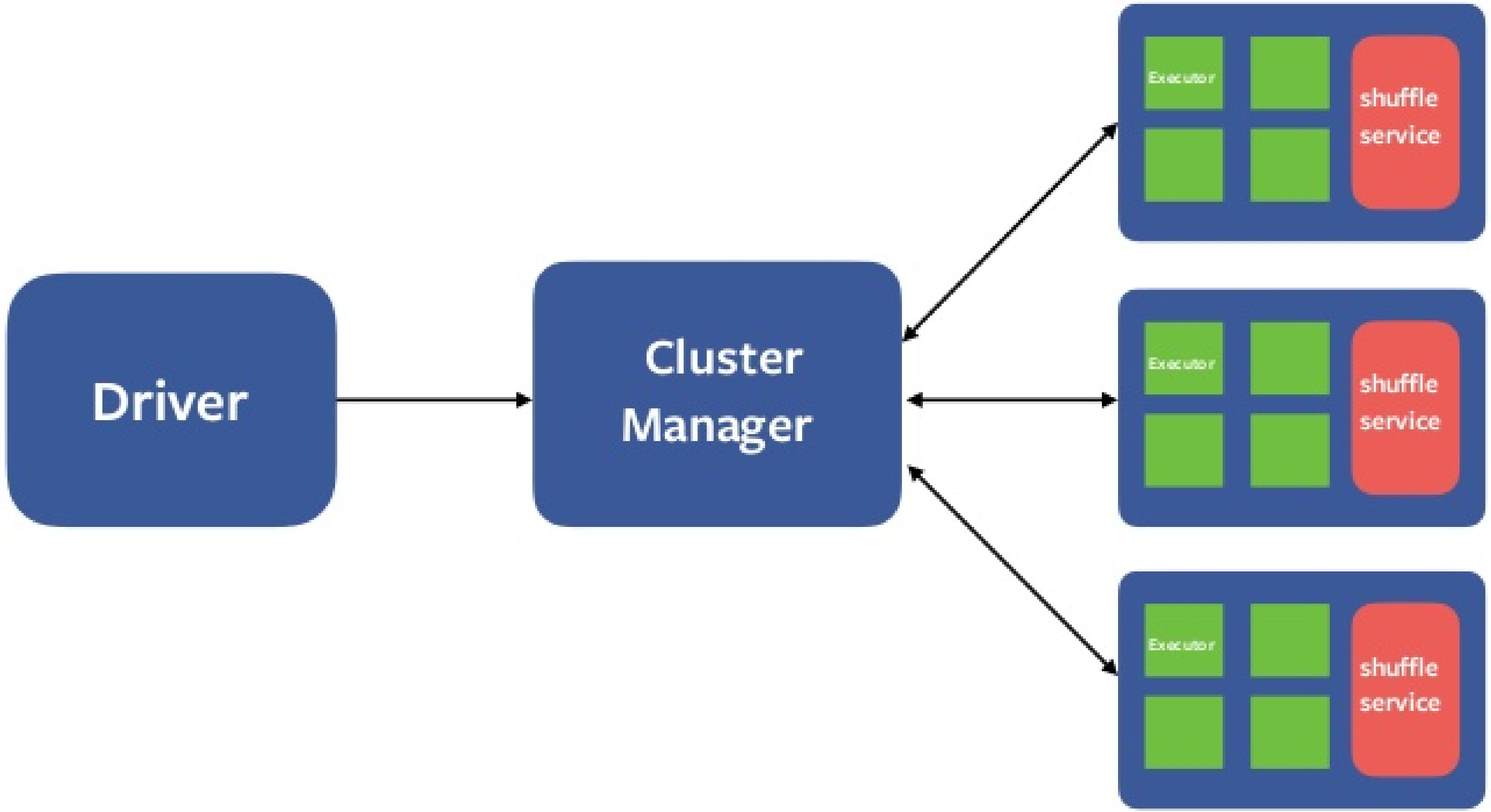
# Agenda

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

# Apache Spark at Facebook

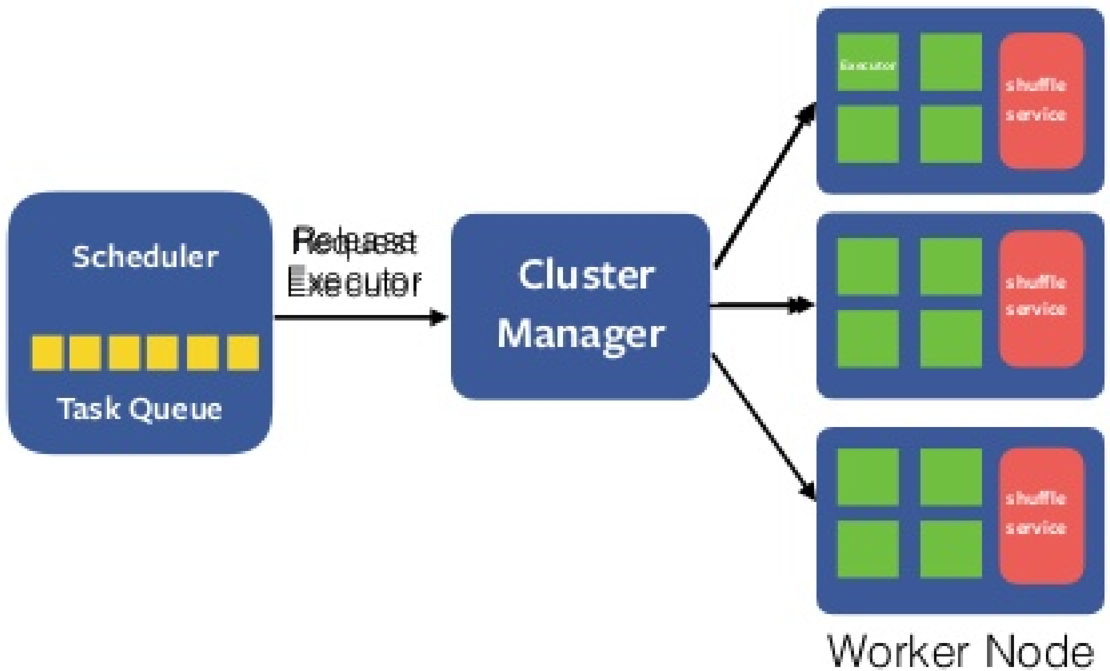
- Used for large scale batch workload
- Tens of thousands of jobs/day and growing
- Running on in the order of thousands of nodes
- Job scalability -
  - Processes hundreds of TBs of compressed input data and shuffle data
  - Runs hundreds of thousands of tasks

# Spark Architecture



# Scaling Spark Driver

# Dynamic Executor Allocation

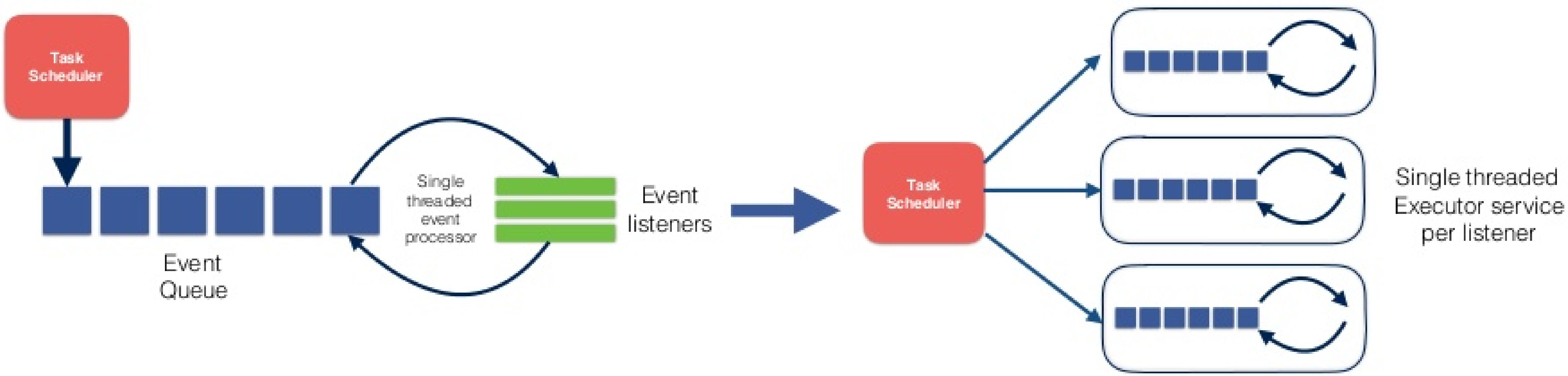



- Better Resource utilization
- Good for multi-tenant environment

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

# Multi-threaded Event Processor

[SPARK-18838]




Single threaded event processor architecture

Multi-threaded event processor architecture

# 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) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 21:47:11 | 2.2 h | 20011/20011 (8 failed) | | |
| 1 (retry 3) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 20:26:14 | 1.3 h | 55/55 (3 failed) | 179.8 GB | |
| 0 (retry 5) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 20:22:08 | 27 min | 31/31 | 94.9 GB | |
| 0 (retry 4) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 20:09:28 | 13 min | 35/35 | 107.7 GB | |
| 0 (retry 3) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 19:35:13 | 34 min | 16/16 (1 failed) | 49.3 GB | |
| 1 (retry 2) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 19:21:20 | 1.1 h | 12/12 (1 failed) | 36.9 GB | |
| 0 (retry 2) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 19:18:03 | 17 min | 40/40 | 123.6 GB | |
| 1 (retry 1) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 19:08:28 | 13 min | 4/4 | 12.4 GB | |
| 0 (retry 1) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 19:08:28 | 9.5 min | 13/13 | 40.3 GB | |
| 1 | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 17:11:22 | 1.2 h | 17560/17560 (27 failed) | 52.9 TB | |
| 0 | sql at GenericHQLEvaluator.scala:28 | +details | 2017/02/15 17:11:20 | 1.3 h | 25025/25025 (1 failed) | 75.2 TB | |

Failed Stages (1)

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write | Failure Reason |
|---|---|---|---|---|---|---|---|---|---|
| 2 | sql at GenericHQLEvaluator.scala:28 +details | 2017/02/15 18:58:12 | 10 min | 0/20011 (7197 | | | 5.6 TB | | org.apache.spark.shuffle.Fe [redacted] |

Single fetch failure causing multiple retries

Completed Stages (6)

| Stage Id | Description | | Submitted | Duration | Tasks: Succeeded/Total | Input | Outp |
|---|---|---|---|---|---|---|---|
| 3 | sql at GenericHQLEvaluator.scala:28 | +details | 2017/05/23 19:03:33 | 27 min | 4996/4996 | | |
| 2 (retry 1) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/05/23 18:41:52 | 1.4 h | 3712/3712 | | |
| 1 (retry 1) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/05/23 15:38:37 | 3.3 min | 114/114 | 115.0 GB | |
| 0 (retry 1) | sql at GenericHQLEvaluator.scala:28 | +details | 2017/05/23 18:38:37 | 2.1 min | 410/410 | 427.5 GB | |
| 1 | sql at GenericHQLEvaluator.scala:28 | +details | 2017/05/23 14:07:26 | 83 min | 32809/32809 (20 failed) | 38.3 TB | |
| 0 | sql at GenericHQLEvaluator.scala:28 | +details | 2017/05/23 14:57:11 | 51 min | 83082/83082 (22 failed) | 84.5 TB | |

Failed Stages (1)

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write | Failure Reason |
|---|---|---|---|---|---|---|---|---|---|
| 2 | sql at GenericHQLEvaluator.scala:28 +details | 2017/05/23 18:48:11 | 45 min | 1257/4988 (3712 | | | 14.7 TB | 2.6 TB | org.apache.spark.sh reset by peer |

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

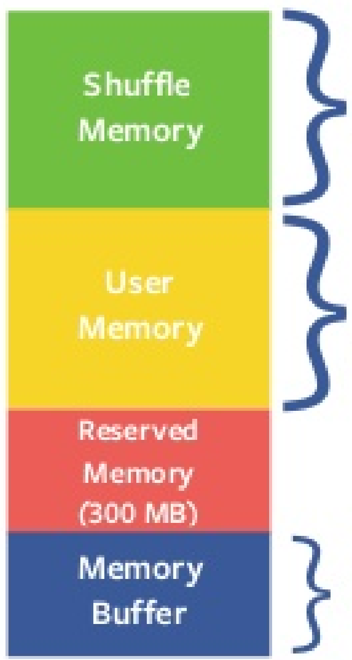
## Tune RPC Server threads

- 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



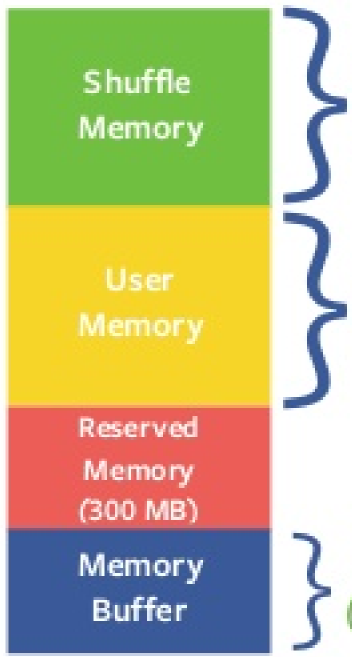

`spark.memory.fraction * (spark.executor.memory - 300 MB)`

`(1- spark.memory.fraction) * (spark.executor.memory - 300 MB)`

`spark.yarn.executor.memoryOverhead =`
`0.1 * (spark.executor.memory)`

# Tuning memory configurations

## Enable off-heap memory




```
spark.memory.offHeap.enabled = true
spark.memory.offHeap.size = 3g

spark.executor.memory = 3g

spark.yarn.executor.memoryOverhead = 0.1 *
(spark.executor.memory + spark.memory.offHeap.size)
```

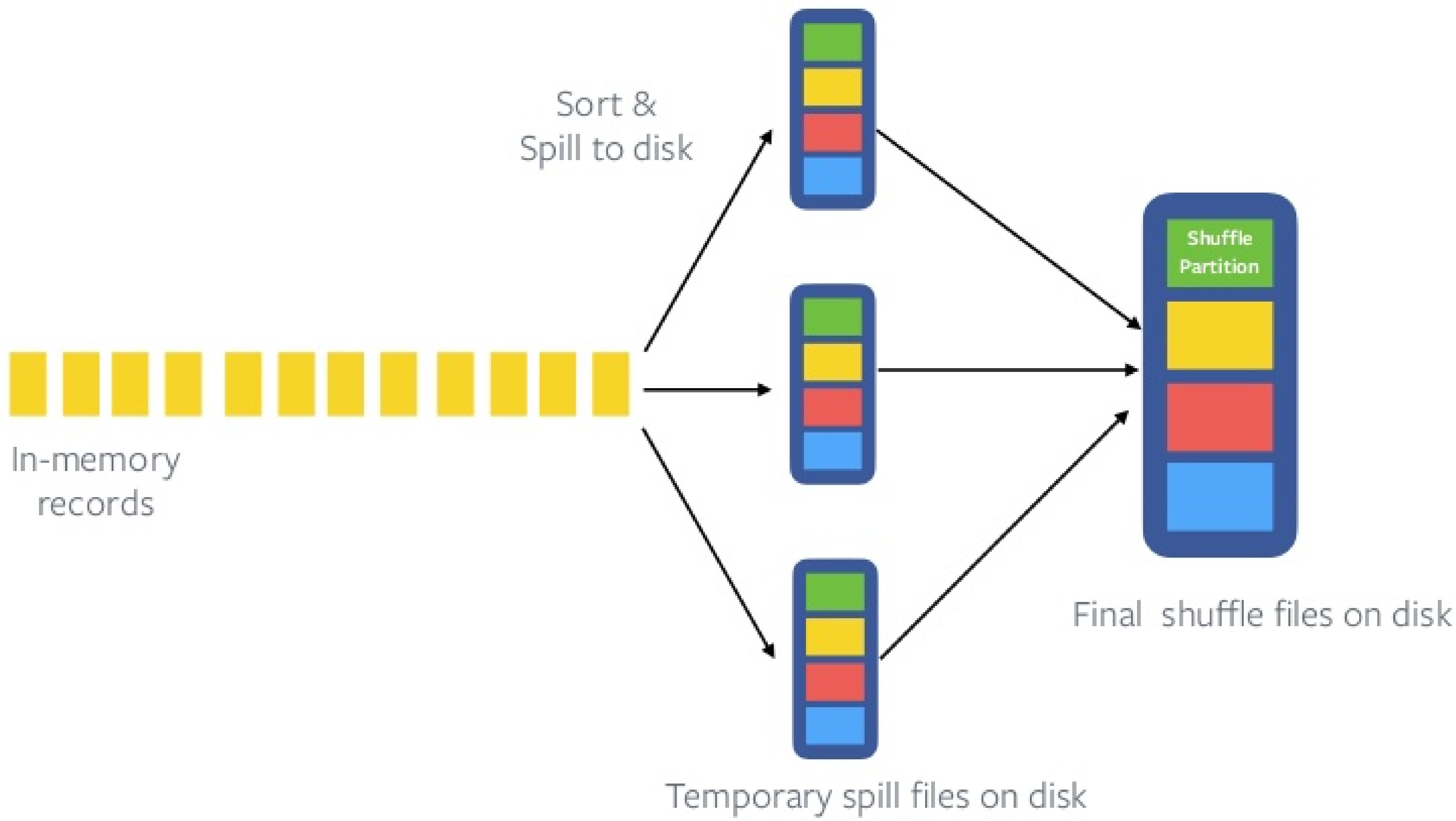
# Tuning memory configurations

## 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

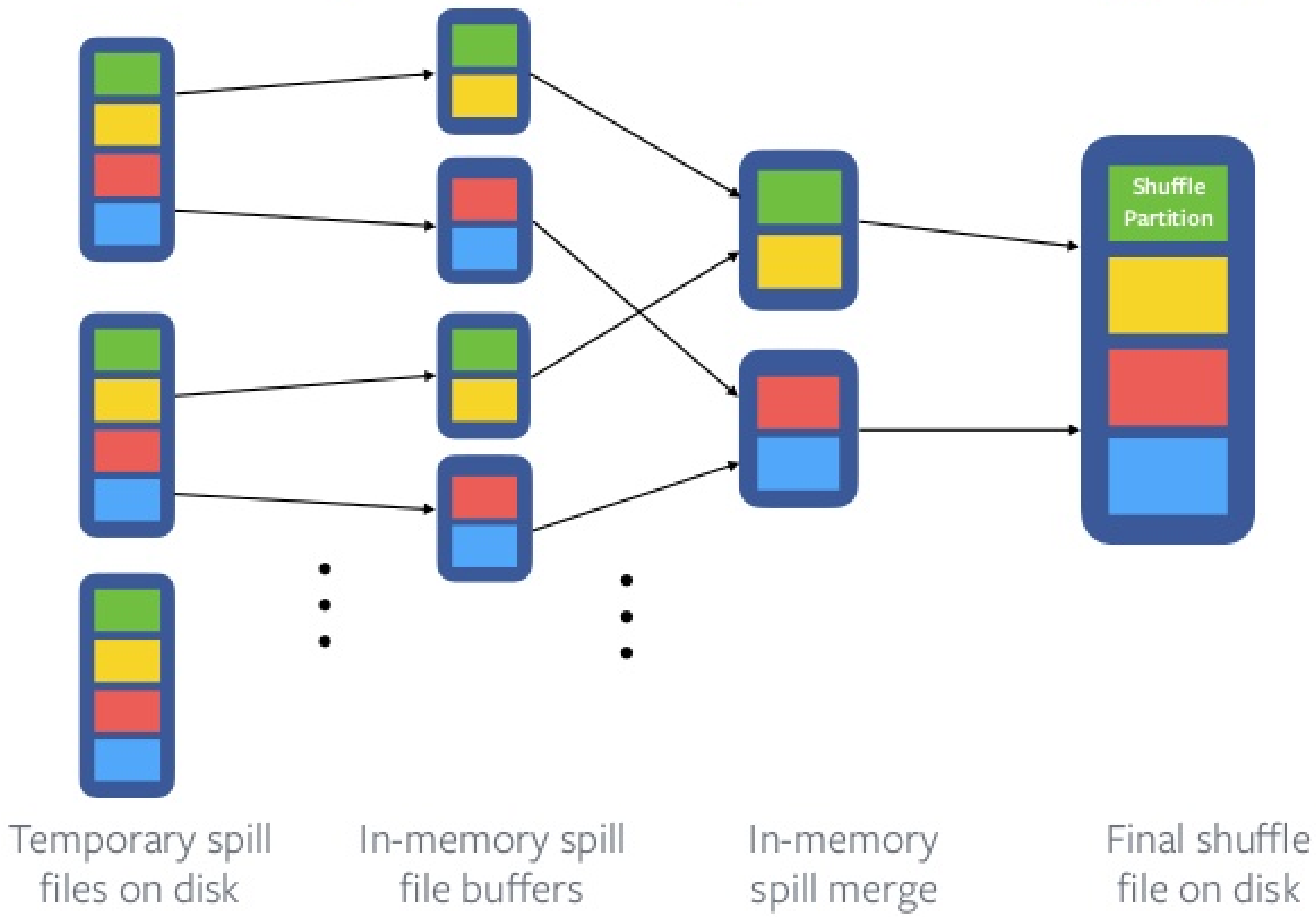
## Tune Shuffle file buffer

- 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

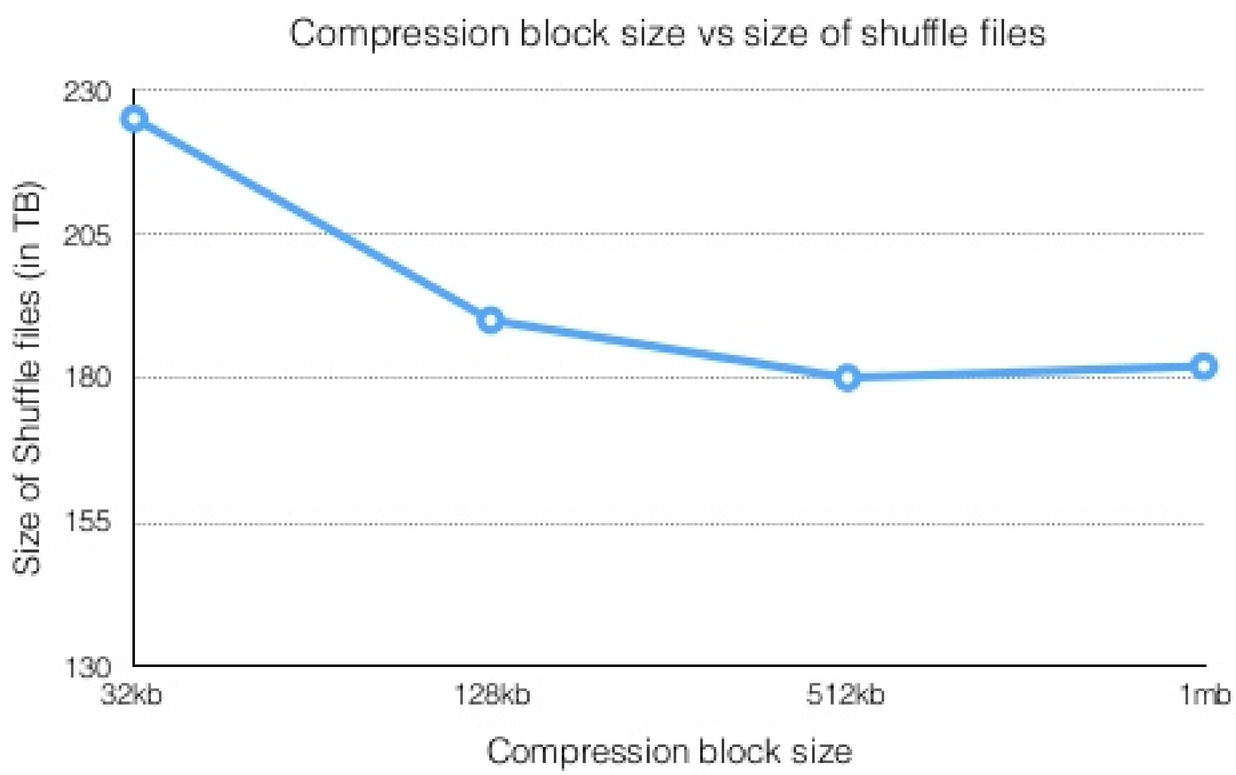
## [SPARK-20014] Optimize spill files merging




```
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
```

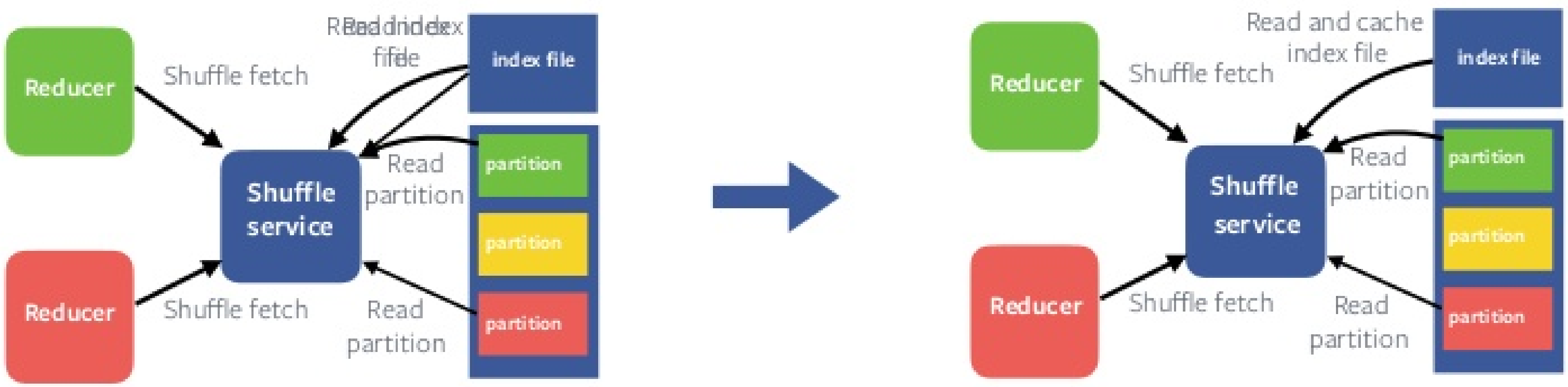
# Various Memory leak fixes and improvements

- 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 Shuojie 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

Even faster: Data at the speed of Presto ORC

Scaling the Facebook data warehouse to 300 PB

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

```
// number of mappers
num of mappers =
max(256 MB, inputTableSize / 50000)

// number of reducers
num of reducers = max(
  200,
  min(10000, max(inputTableSize / 256 MB * 0.125, 200))
)
```

- 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**

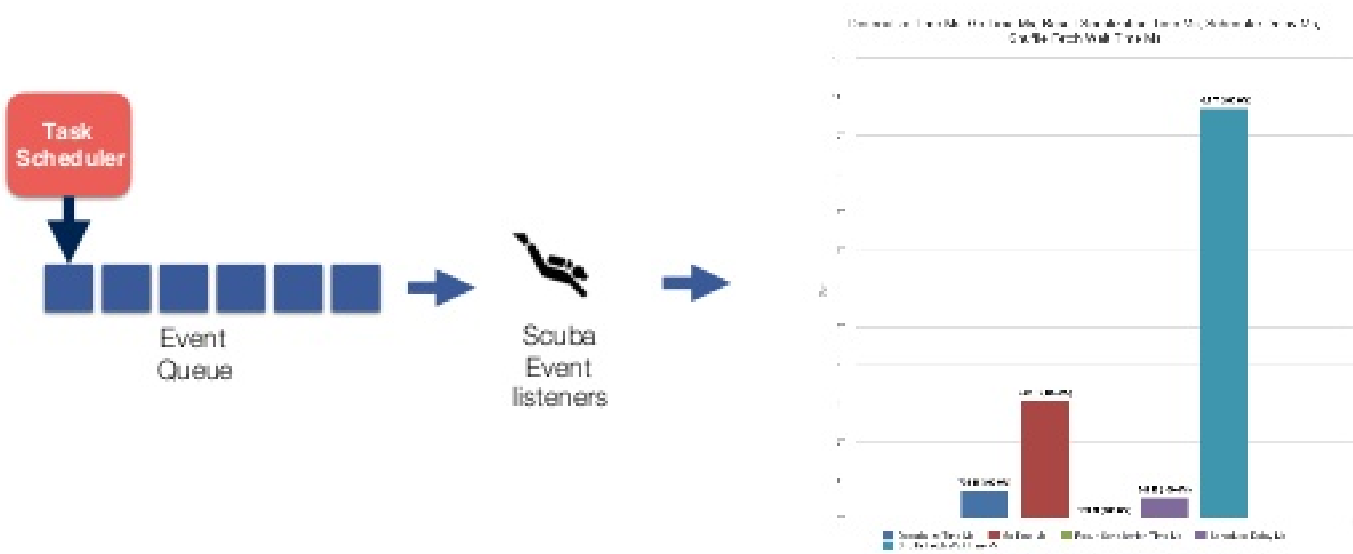
| Metric | Min | 25th percentile | Median | 75th percentile | Max |
|---|---|---|---|---|---|
| Duration | 1.4 min | 11 min | 15 min | 20 min | 1.1 h |
| Scheduler Delay | 0.1 s | 0.1 s | 0.1 s | 0.1 s | 1.6 min |
| Task Deserialization Time | 3 ms | 5 ms | 5 ms | 6 ms | 9 s |
| GC Time | 0.2 s | 1 s | 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?