# Profiling and optimizing a Spark job with Babar

BENOIT HANOTTE, BERLIN BUZZWORDS 2018

criteo

“Connecting shoppers to the things they need and love.”

**500TB**
INGESTED DAILY

**15PB**
READ DAILY

**4000+**
HADOOP NODES

criteo.

...

# Context

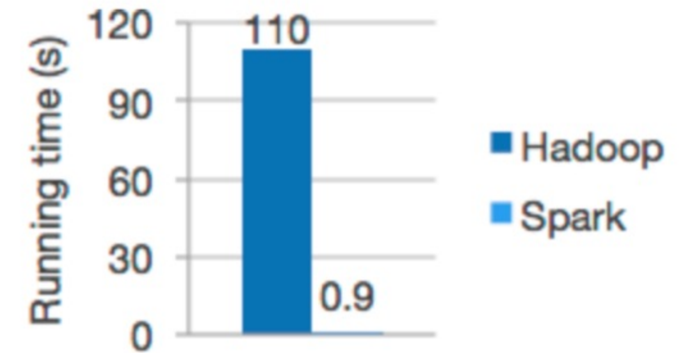
Teams moving from MapReduce-based frameworks to newer alternatives

**Spark** very popular, advertises faster & more efficient processing

## Speed

Run workloads 100x faster.

Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.

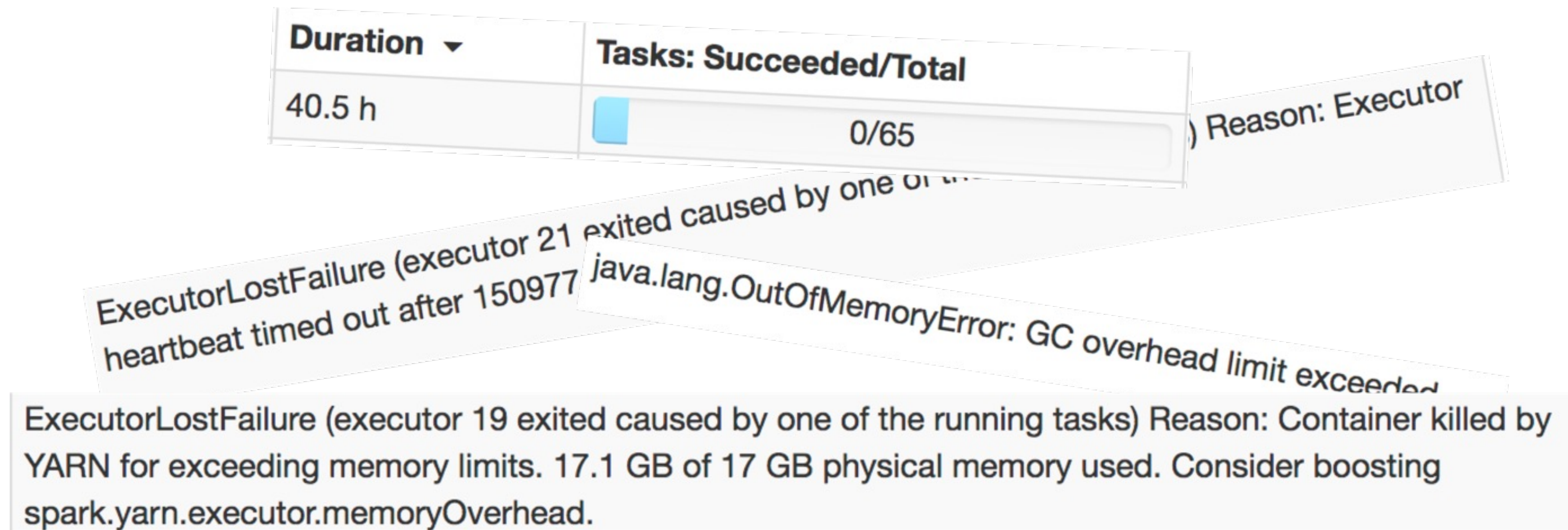



Logistic regression in Hadoop and Spark

Source: http://spark.apache.org (June 2018)

# Context

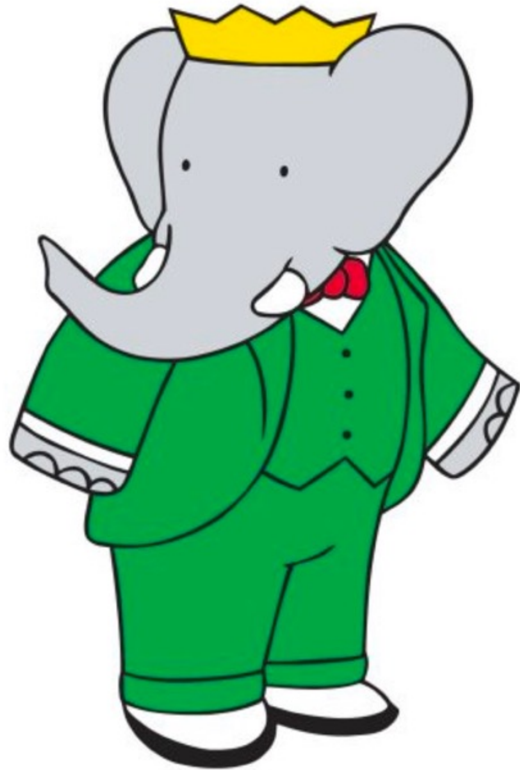
But this is not what we experienced…




Jobs were unstable, teams were solving it by allocating too much resources to their jobs.

criteo

# Babar



Profiler for **distributed applications** on **Hadoop**

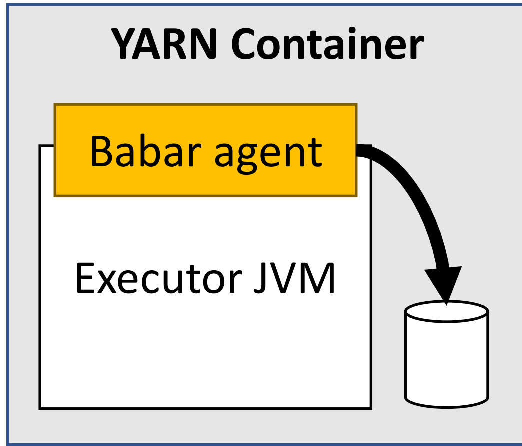
Works with any JVM framework

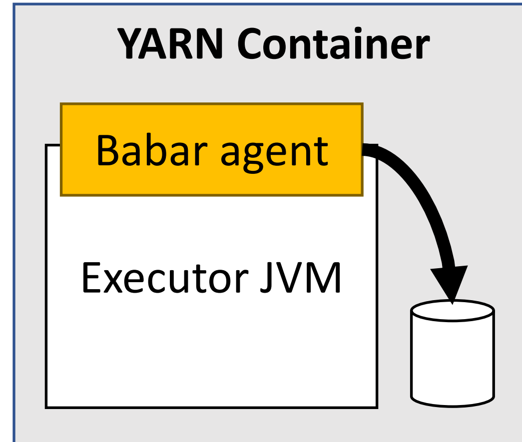
...

- Easy to get started, no infrastructure required
- Made for distributed applications
- Works on 10000+ containers apps
- Exports ready-to-use graphs as HTML file

https://github.com/criteo/babar

# Babar




YARN Container

Babar agent

Executor JVM

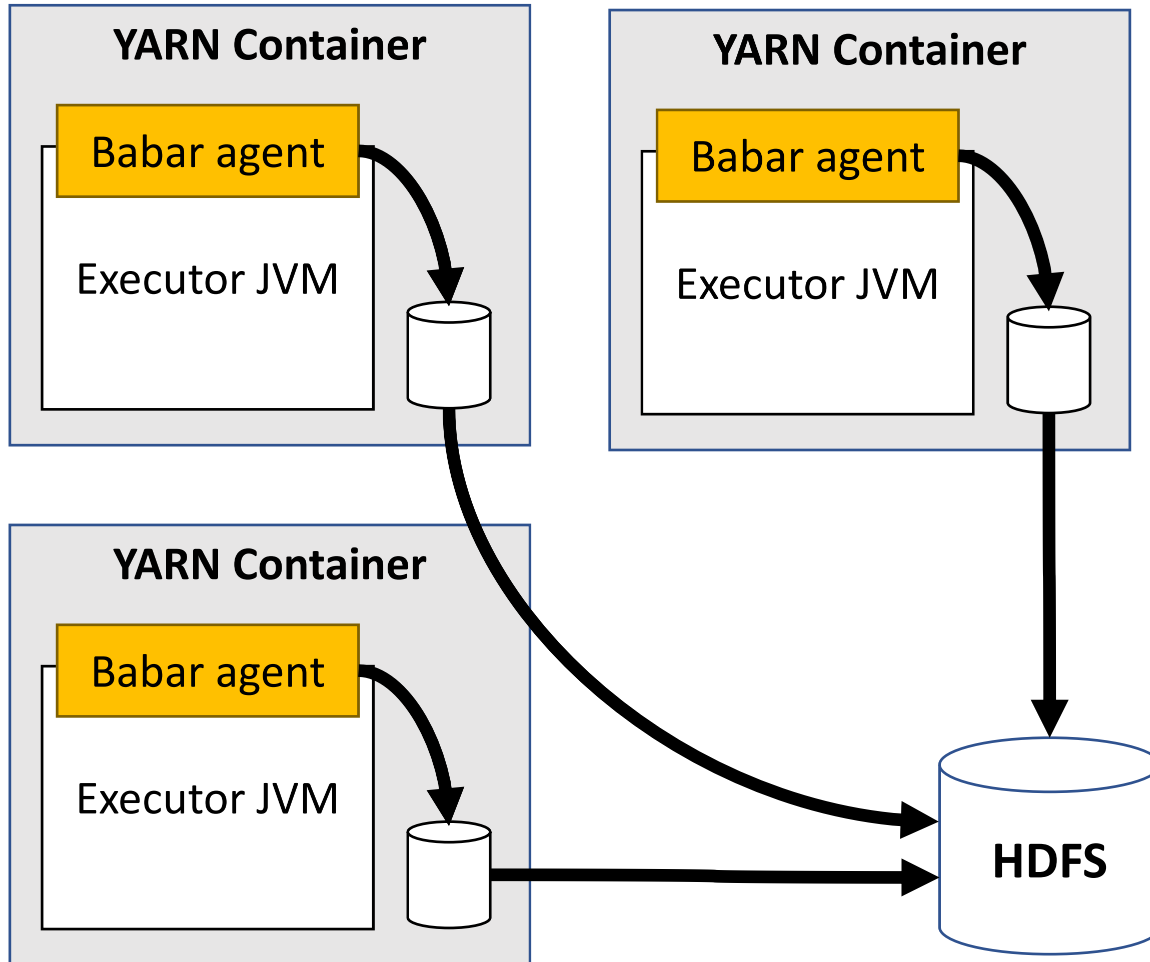
Babar agent instruments the executor JVM, logs metrics to local FS.

Agent can be distributed with spark-submit (no installation).

**Metrics from:**

- JVM
- Stack traces
- /proc/

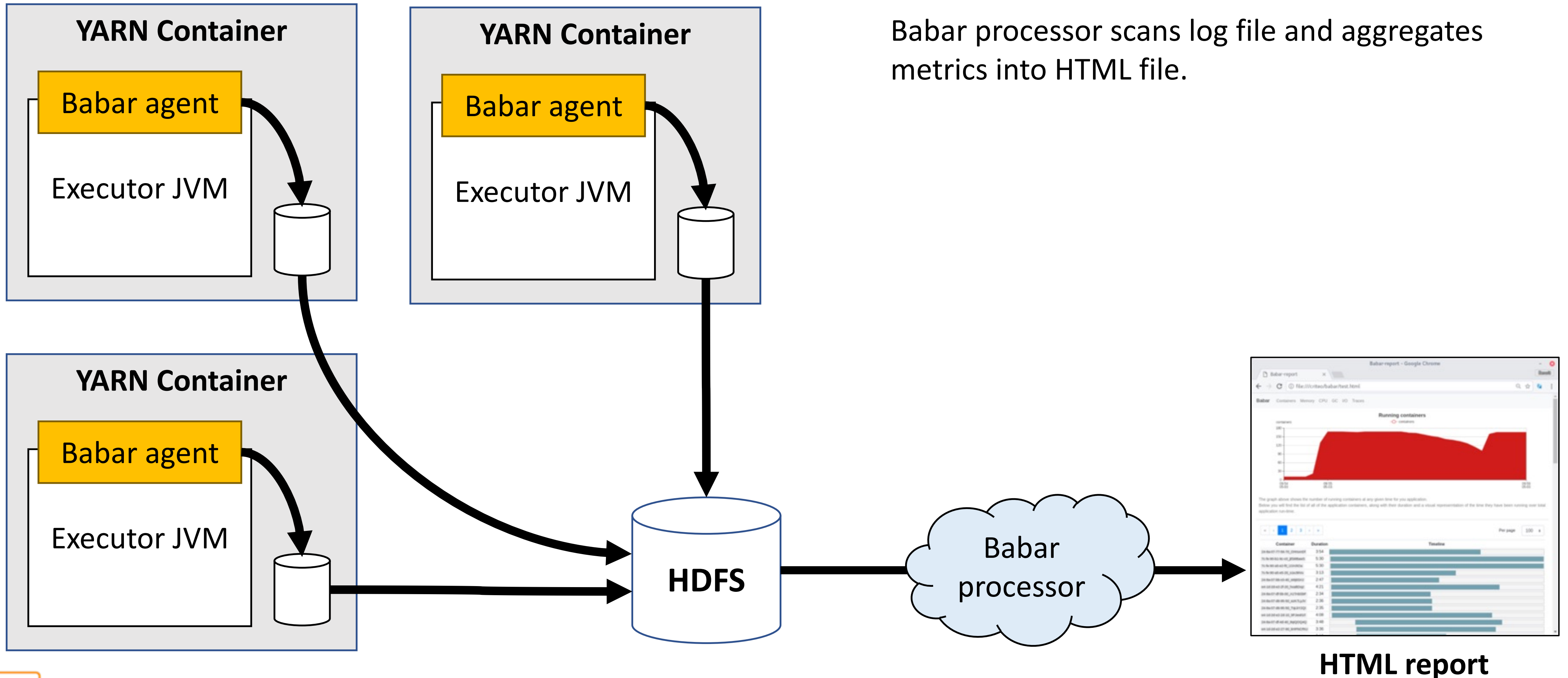
# Babar




On job complete, YARN aggregates logs from all executors into single file on HDFS.

# Babar

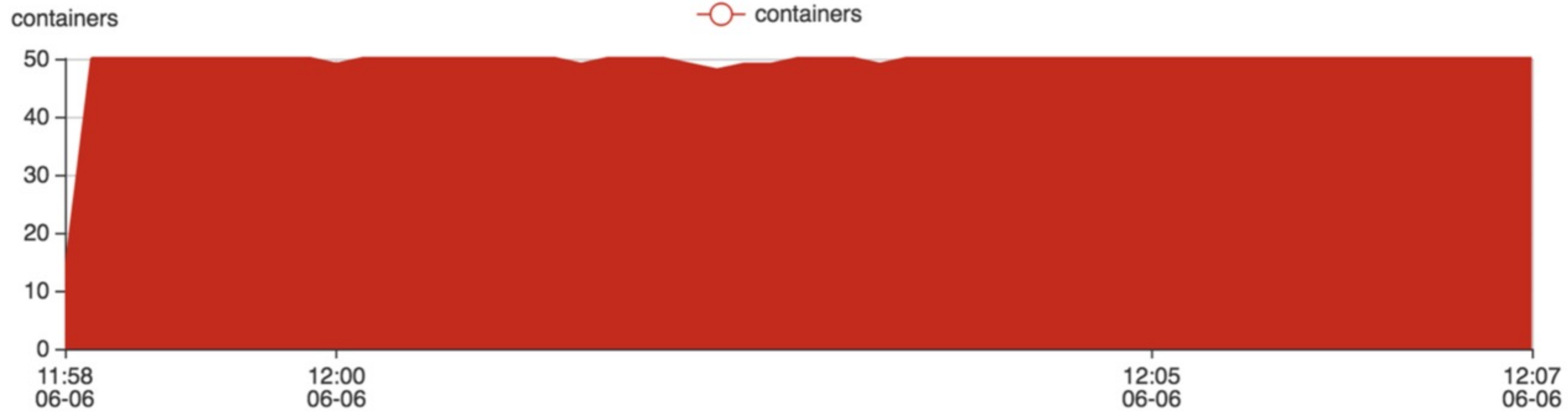
Babar processor scans log file and aggregates metrics into HTML file.



Babar-report

Benoit

Babar Containers Memory CPU GC I/O Traces




The graph above shows the number of running containers at any given time for your application.
Below you will find the list of all of the application containers, along with their duration and a visual represerntation of the time they have been running over total application run-time.

« ‹ 1 › »

Per page 100

| Container | Duration | Timeline |
|---|---|---|
| 24-8a-07-77-67-e0_IN2FxCvP | 8:53 | |
| 24-8a-07-bb-b6-90_mRGM3QrD | 8:52 | |
| 24-8a-07-bb-e2-90_D9jcq9xa | 8:52 | |
| 24-8a-07-bb-e2-00_DoyBHk3q | 8:51 | |

criteo

Babar-report

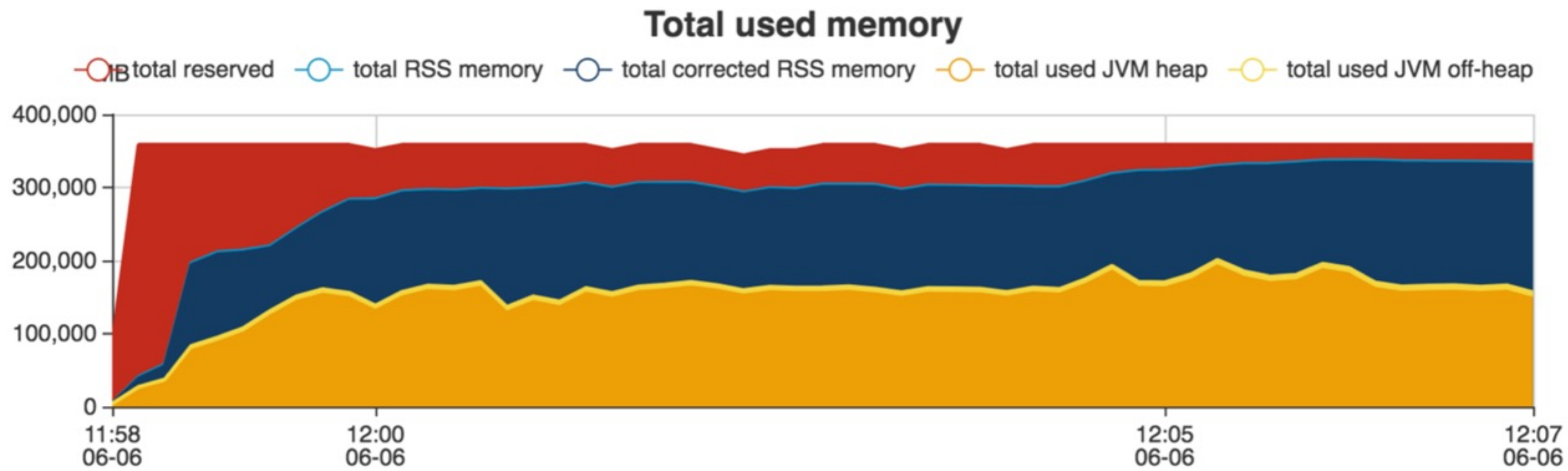
Benoit

Babar Containers Memory CPU GC I/O Traces

**Total memory** shows the sum of the memory usage over all containers
**Max memory** shows the maximum memory usage for any container

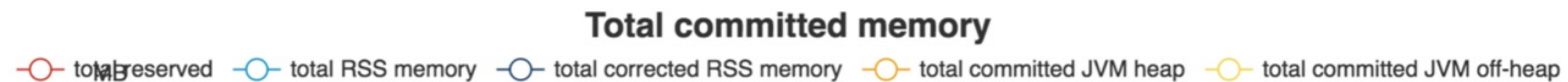
Total memory | Max memory

## Total used memory




This graph shows the total memory used by all of your application's running containers at any given time.
**Reserved memory** is the amount of memory reserved on the infrastructure.
**RSS memory** is the size of the memory pages that are loaded in the physical memory (RAM) for your containers process-tree (inclusing non JVM programs).

## Total committed memory




criteo

Babar-report

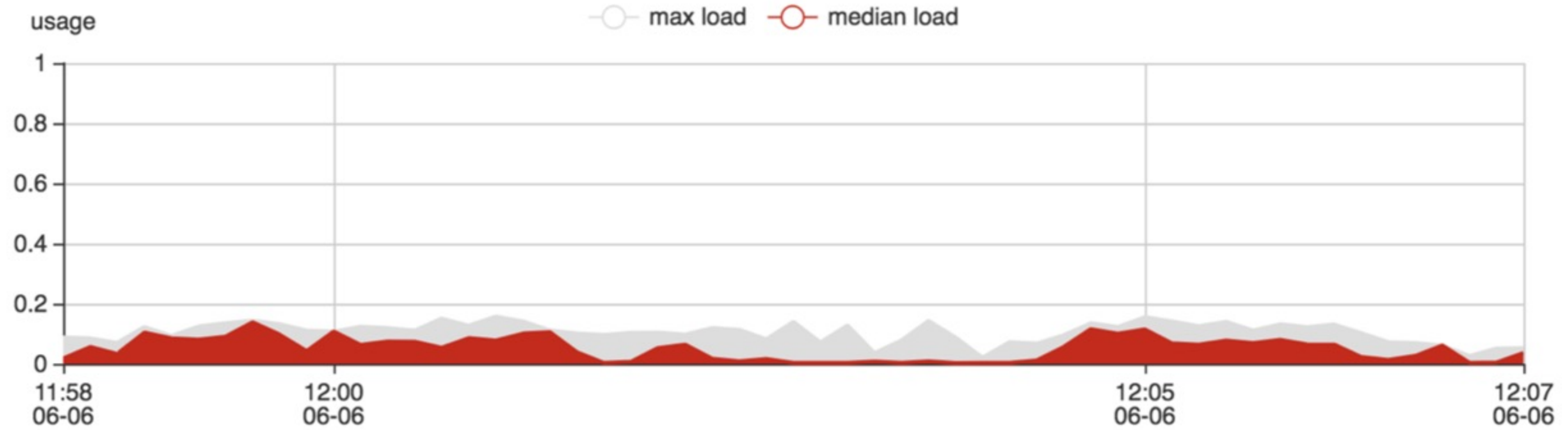
Benoit

Babar Containers Memory CPU GC I/O Traces

**JVM** shows the memory usage as reported by the JVM instrumentation using the *JVMProfiler*.
**ProcFS** shows the memory usage as reported by the `/proc` filesystem using the *ProcFSProfiler*.

JVM ProcFS

## JVM CPU usage




This graph shows the median and max **CPU usage as reported by the JVM instrumentation** over all the containers.
It only reports the JVM CPU usage and will ignore children processes spawned by the java application.

## Host CPU usage

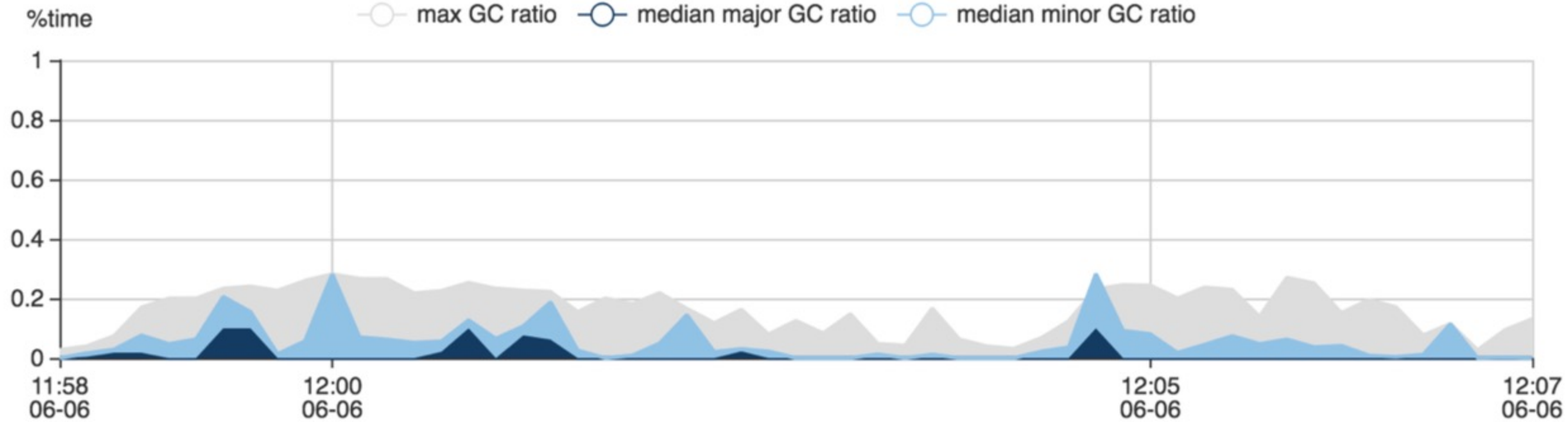
criteo

Babar-report

Benoit

Babar Containers Memory CPU GC I/O Traces

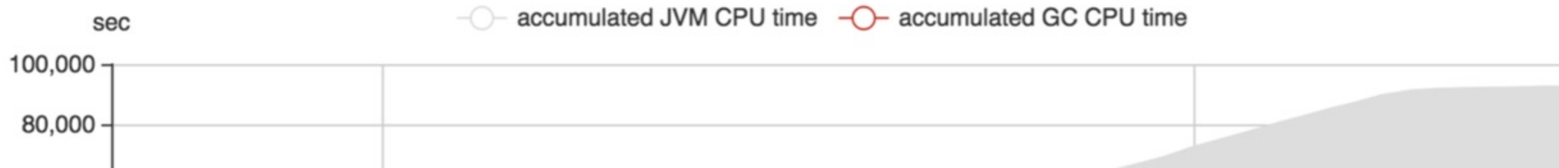
## Minor & Major median GC ratio




This graph shows the median ratio of wall-clock time spent doing minor and major garbage collections in the JVMs over all containers. **Minor GC** only clean the young generation, while **Major GC** cleans both the young and old ones. The major GC should be much less frequent that the minor one, otherwise it could indicate that too many short lived-objects are promototed to the old generation. If this is the case, you may want to resize the generations and make sure that no humongous object uses most of the old generation (which could trigger frequent major GC).

You can tune the size of the generations either by specifying the `-XX:NewRatio` (integer value only) parameter or with the `-XX:NewSize` and `-XX:MaxNewSize` parameters for finer-grained tuning.

## Accumulated JVM CPU time and GC CPU time



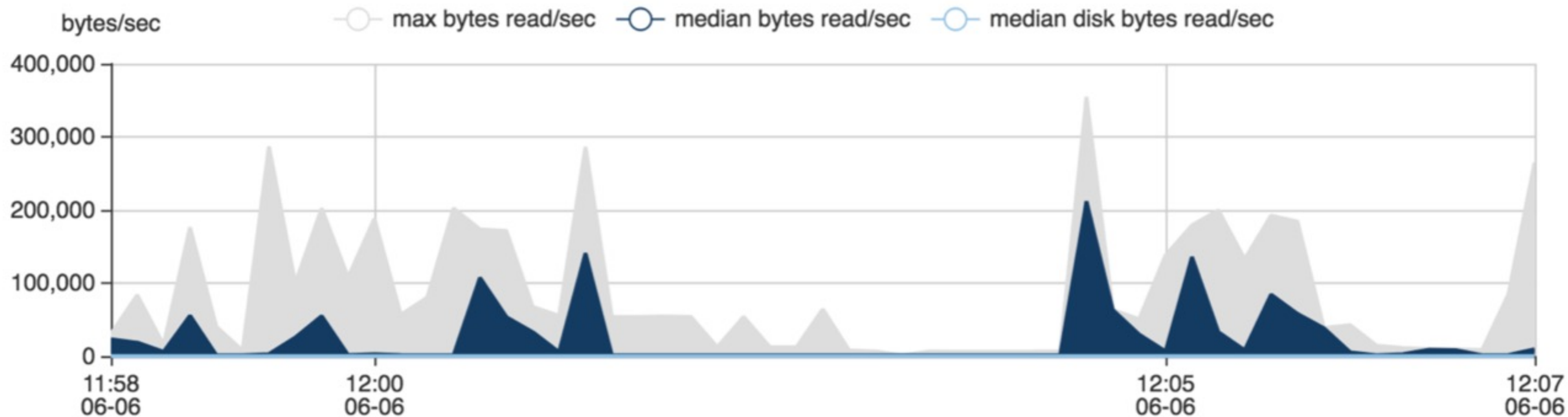

criteo

Babar-report

Benoit

**Babar** Containers Memory CPU GC I/O Traces

## Process tree peak bytes read / sec

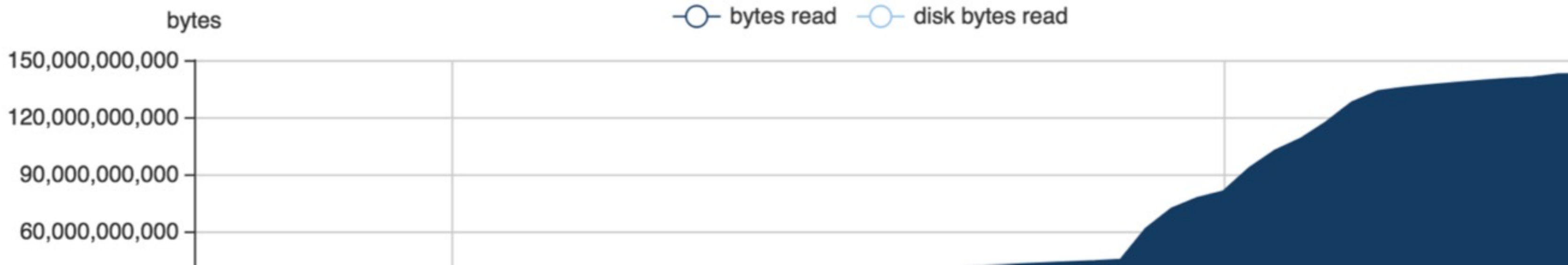


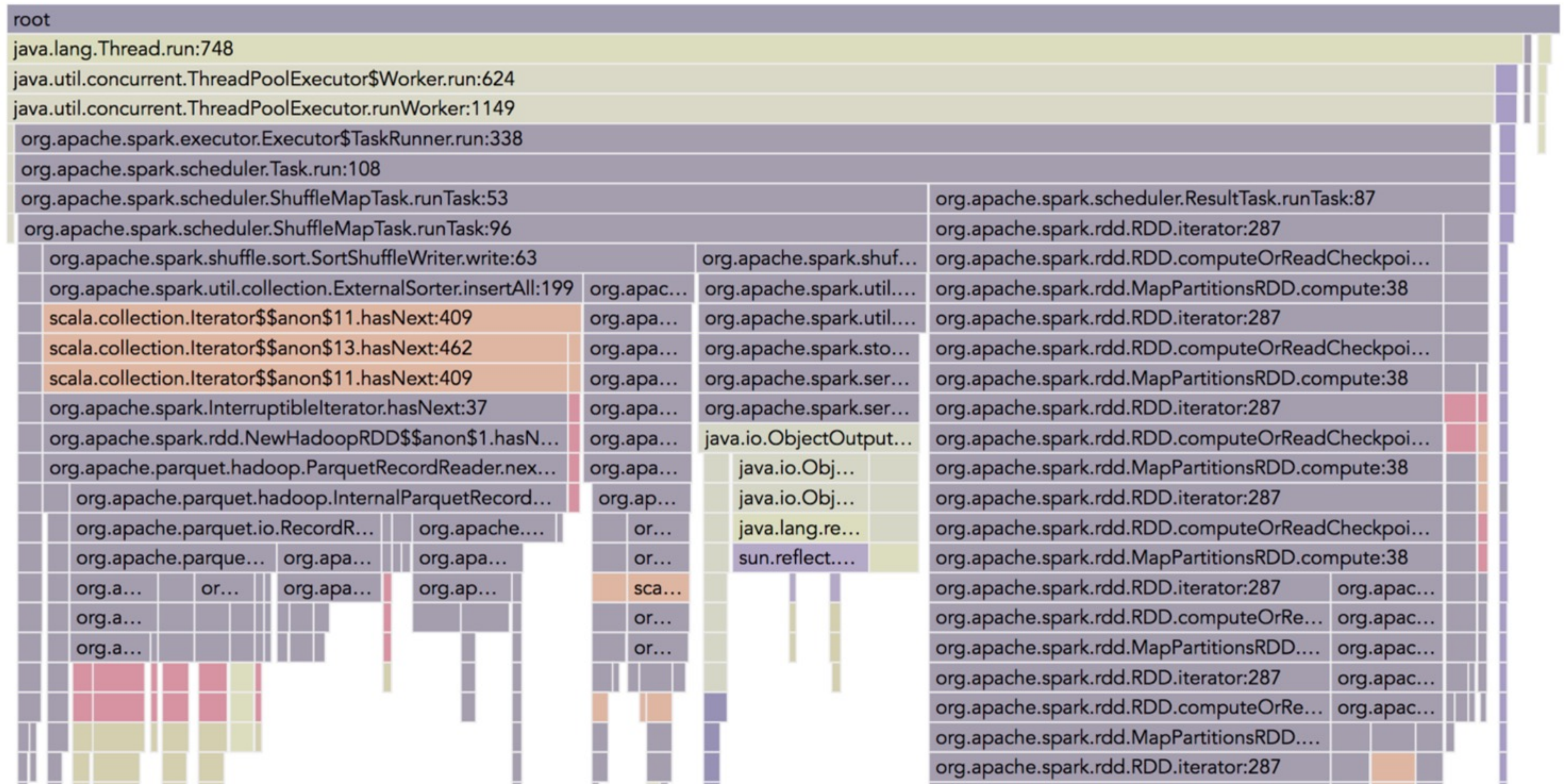

This graph shows the peak bandwidth used to read data from disk I/O by the entire process-tree as reported by the `/prod/[pid]/io` file.

**Bytes read/sec** show the bandwidth for all read operations, while **disk bytes read/sec** only shows the bandwidth for I/O operations that acutally read data from the block storage layer (i.e. excluding pagecache).

---

## Process tree Accumulated bytes read



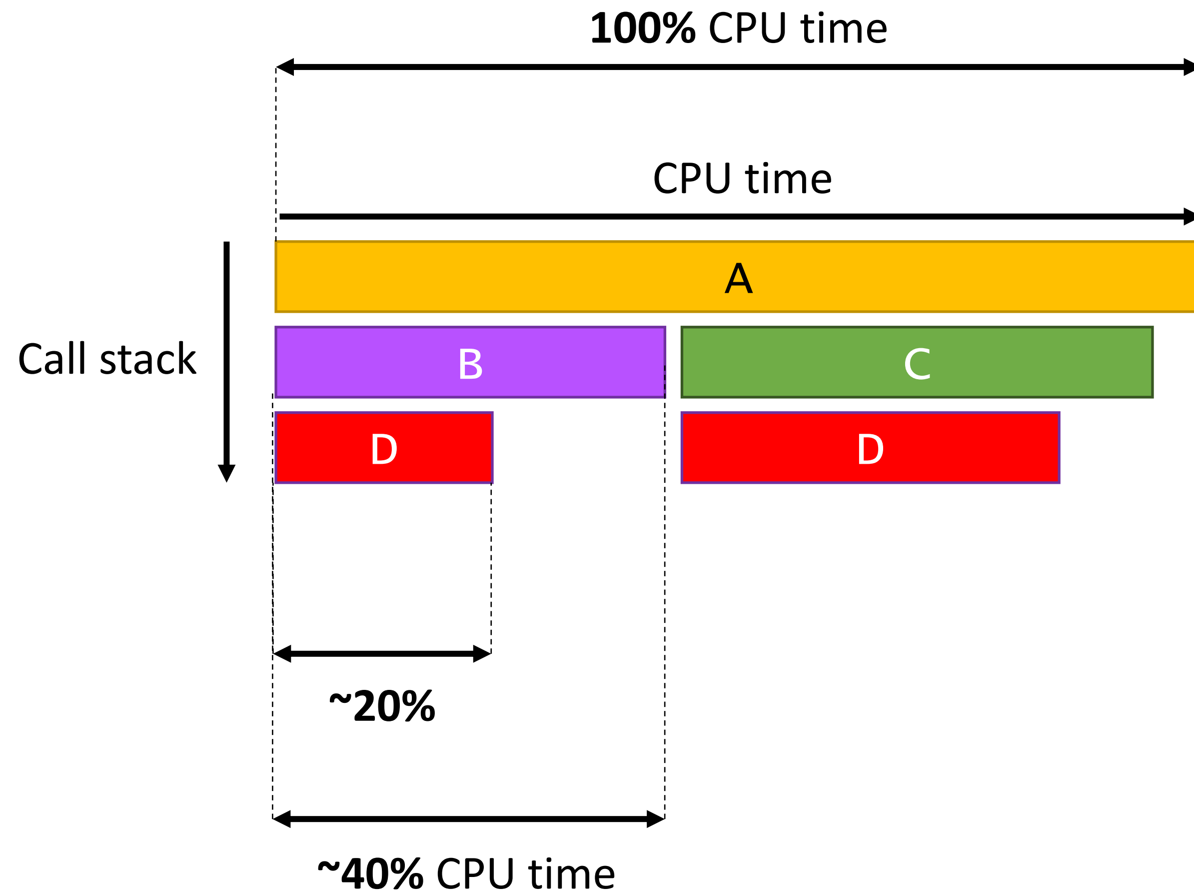


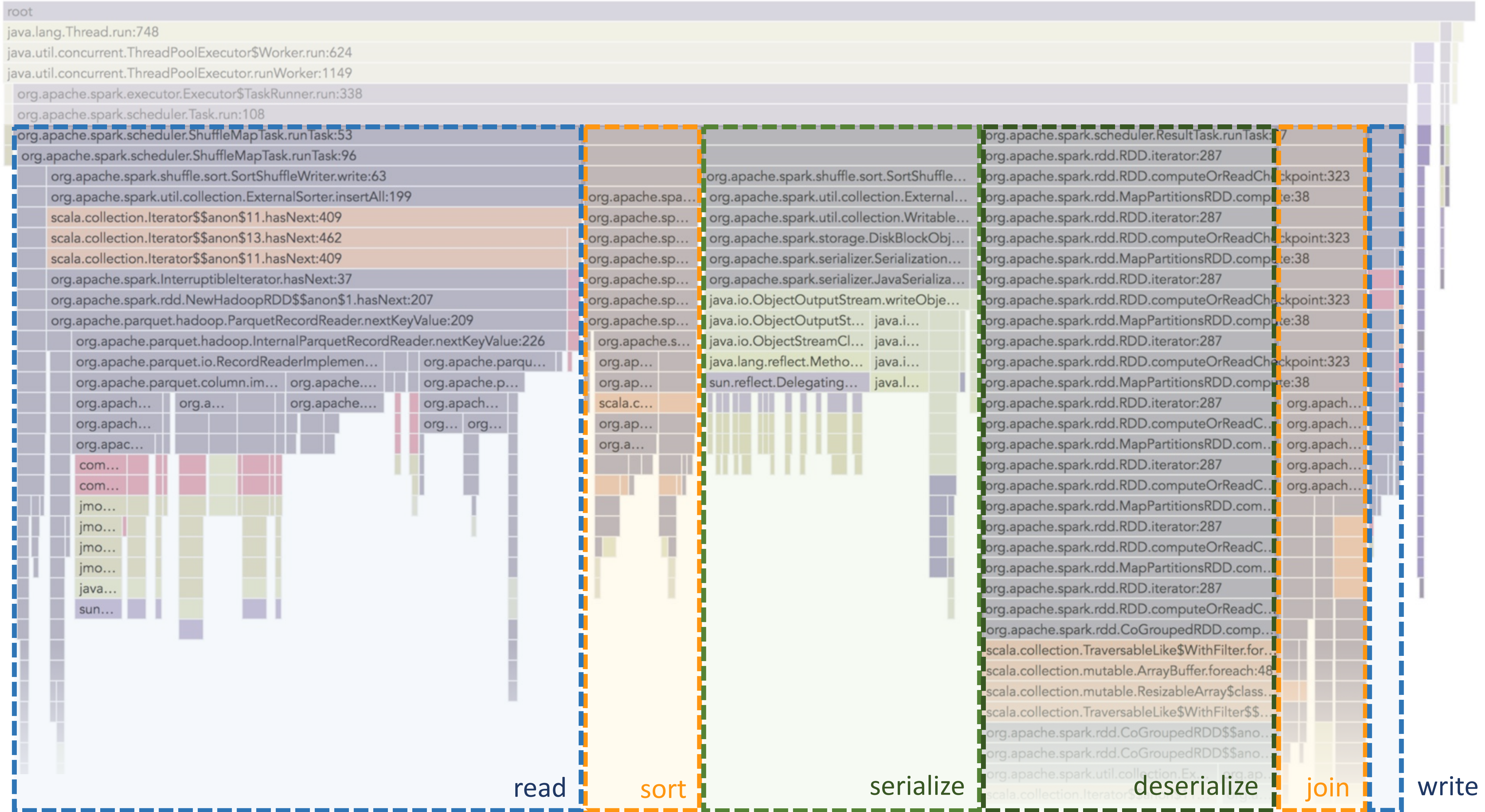

criteo

# In practice: Flame Graphs & CPU time

**Visualize expensive code paths from sampled stack traces**

```
A(){
    B(){
        D(){
        }
    }
    C(){
        D(){
        }
    }
}
```

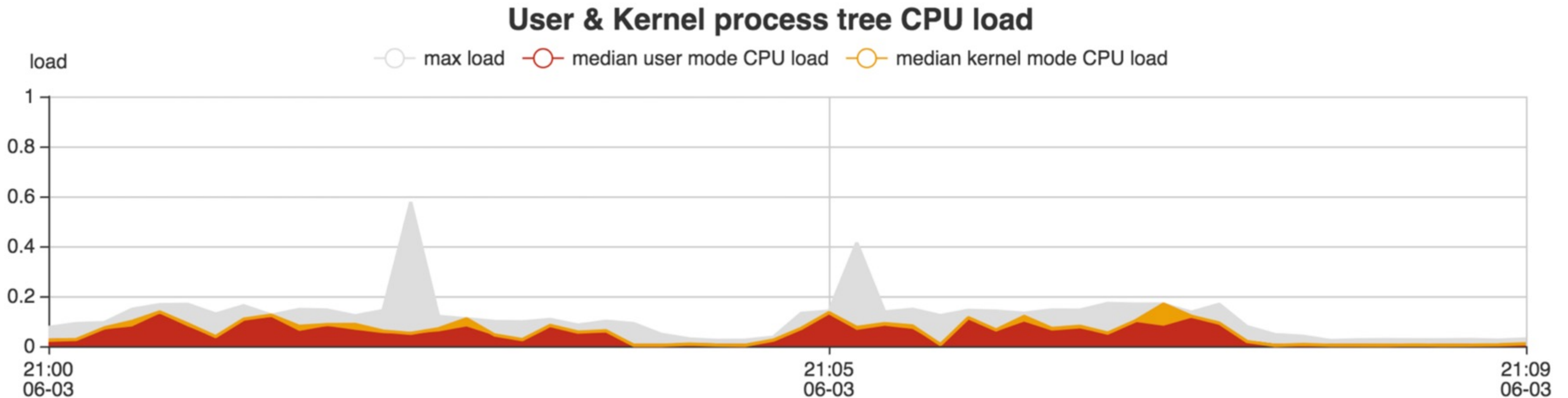



root
java.lang.Thread.run:748
java.util.concurrent.ThreadPoolExecutor$Worker.run:624
java.util.concurrent.ThreadPoolExecutor.runWorker:1149
org.apache.spark.executor.Executor$TaskRunner.run:338
org.apache.spark.scheduler.Task.run:108
org.apache.spark.scheduler.ShuffleMapTask.runTask:53
org.apache.spark.scheduler.ShuffleMapTask.runTask:96
org.apache.spark.shuffle.sort.SortShuffleWriter.write:63
org.apache.spark.util.collection.ExternalSorter.insertAll:199
scala.collection.Iterator$$anon$11.hasNext:409
scala.collection.Iterator$$anon$13.hasNext:462
scala.collection.Iterator$$anon$11.hasNext:409
org.apache.spark.InterruptibleIterator.hasNext:37
org.apache.spark.rdd.NewHadoopRDD$$anon$1.hasNext:207
org.apache.parquet.hadoop.ParquetRecordReader.nextKeyValue:209
org.apache.parquet.hadoop.InternalParquetRecordReader.nextKeyValue:226
org.apache.parquet.io.RecordReaderImplemen…
org.apache.parqu…
org.apache.parquet.column.im…
org.apache.…
org.apache.p…
org.apache.spark.shuffle.sort.SortShuffle…
org.apache.spark.util.collection.External…
org.apache.spark.util.collection.Writable…
org.apache.spark.storage.DiskBlockObj…
org.apache.spark.serializer.Serialization…
org.apache.spark.serializer.JavaSerializa…
java.io.ObjectOutputStream.writeObje…
java.io.ObjectOutputSt…
java.io.ObjectStreamCl…
java.lang.reflect.Metho…
sun.reflect.Delegating…
org.apache.spark.scheduler.ResultTask.runTask:77
org.apache.spark.rdd.RDD.iterator:287
org.apache.spark.rdd.RDD.computeOrReadCheckpoint:323
org.apache.spark.rdd.MapPartitionsRDD.compute:38
org.apache.spark.rdd.CoGroupedRDD.comp…
scala.collection.TraversableLike$WithFilter.for…
scala.collection.mutable.ArrayBuffer.foreach:48
scala.collection.mutable.ResizableArray$class…
scala.collection.TraversableLike$WithFilter$$…
org.apache.spark.rdd.CoGroupedRDD$$ano…
read
sort
serialize
deserialize
join
write

# In practice: Flame Graphs & CPU time

**Shuffle is expensive**

Not network, not IO, but serialization!



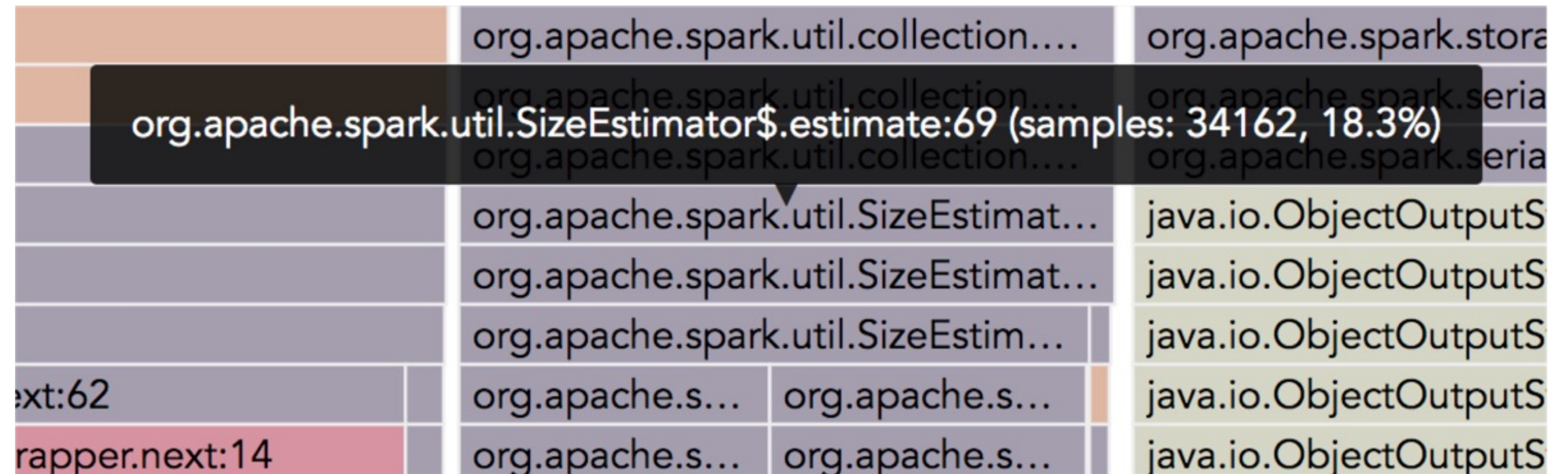
# In practice: Flame Graphs & CPU time

## Pick models accordingly

- Serialization/deserialization efficiency often more important than memory footprint
- **Use specialized serializers** (can allow further optimizations by Spark)

In this example join: **-40% CPU time!**

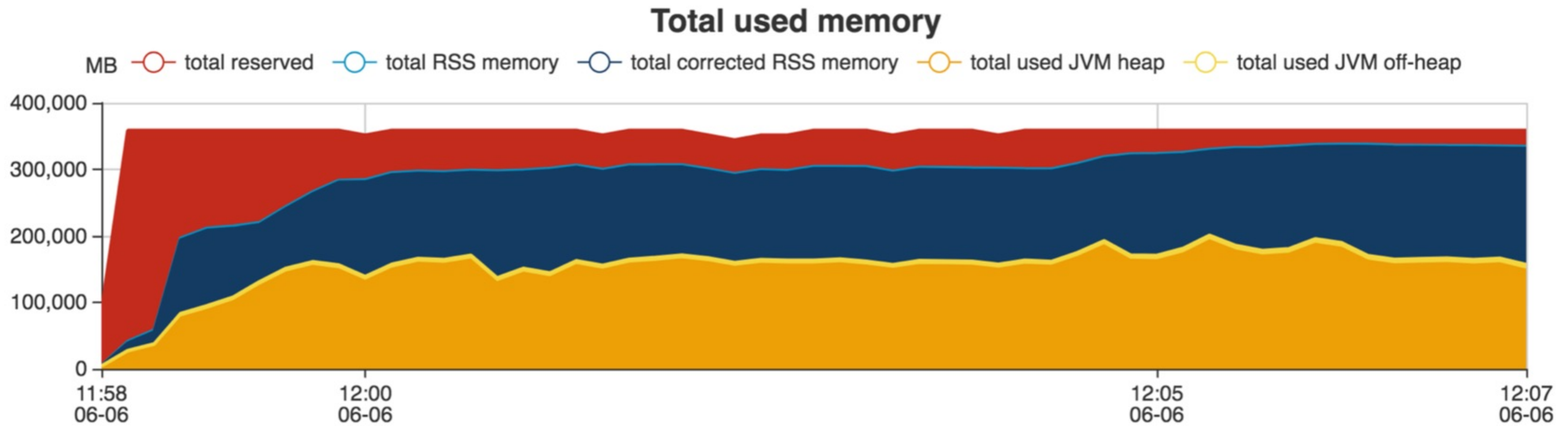
Models also impact cost of **size estimation**
Can be very expensive (we have seen up to 30% CPU time)



# In practice: Memory

Understanding the memory usage

- JVM heap & off-heap memory used
- Physical memory used (RSS memory)
- Reserved memory

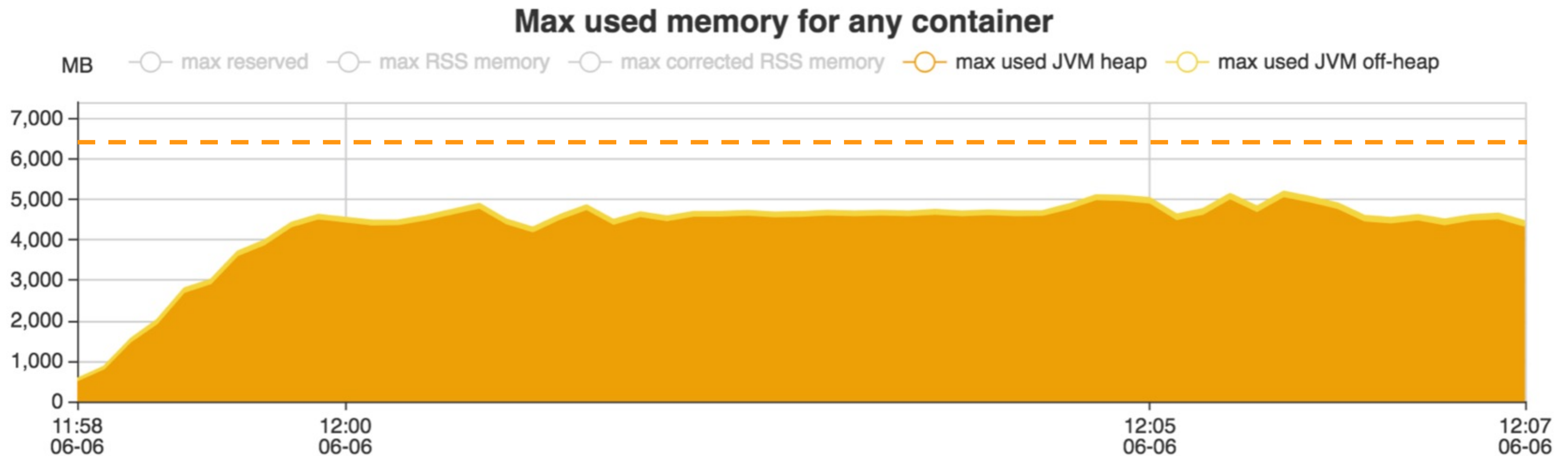


# In practice: Memory

## Dimensioning and tuning memory

Setting executor & container memory to accommodate the largest executor:

- Avoiding OutOfMemoryErrors (spark.executor.memory)

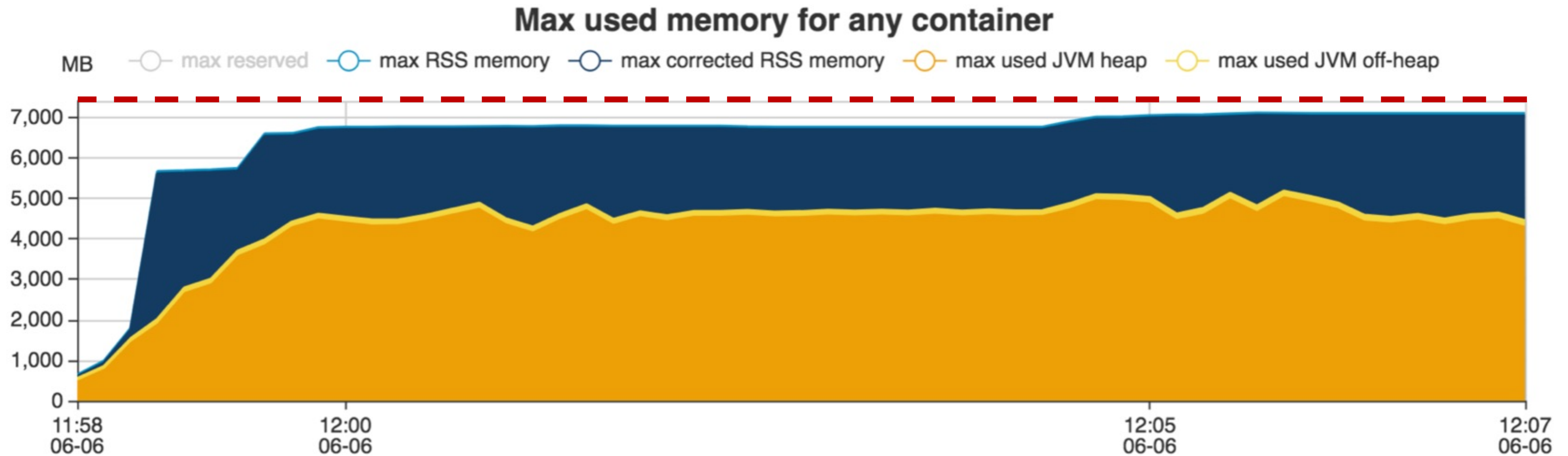


# In practice: Memory

## Dimensioning and tuning memory

Setting executor & container memory to accommodate the largest executor:

- Avoiding OutOfMemoryErrors (spark.executor.memory)

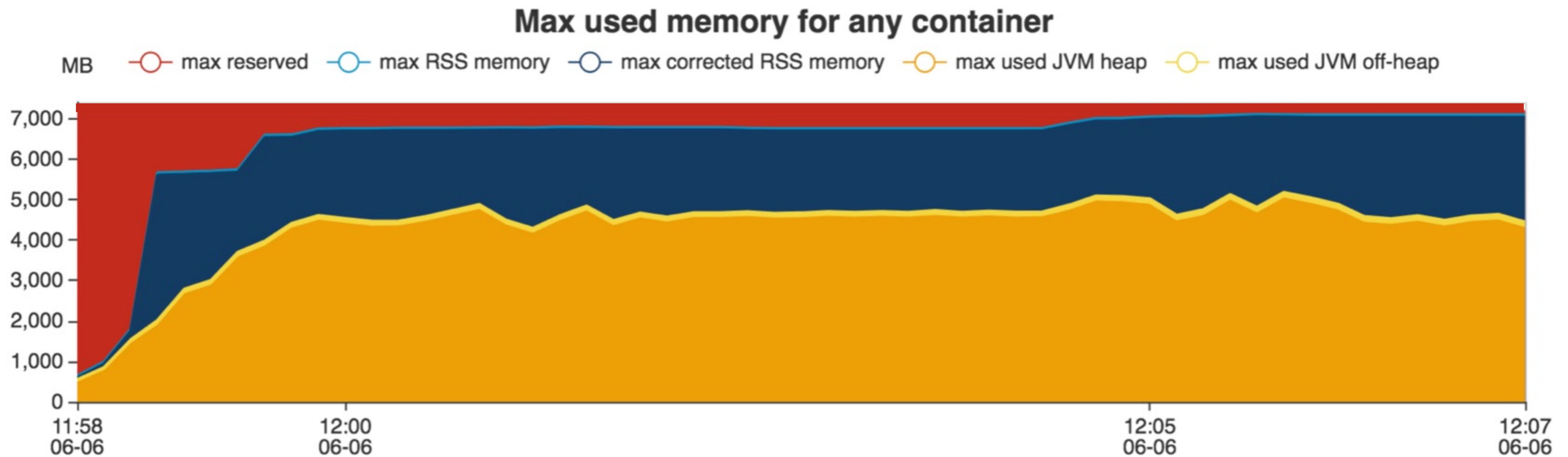


# In practice: Memory

## Dimensioning and tuning memory

Setting executor & container memory to accommodate the largest executor:

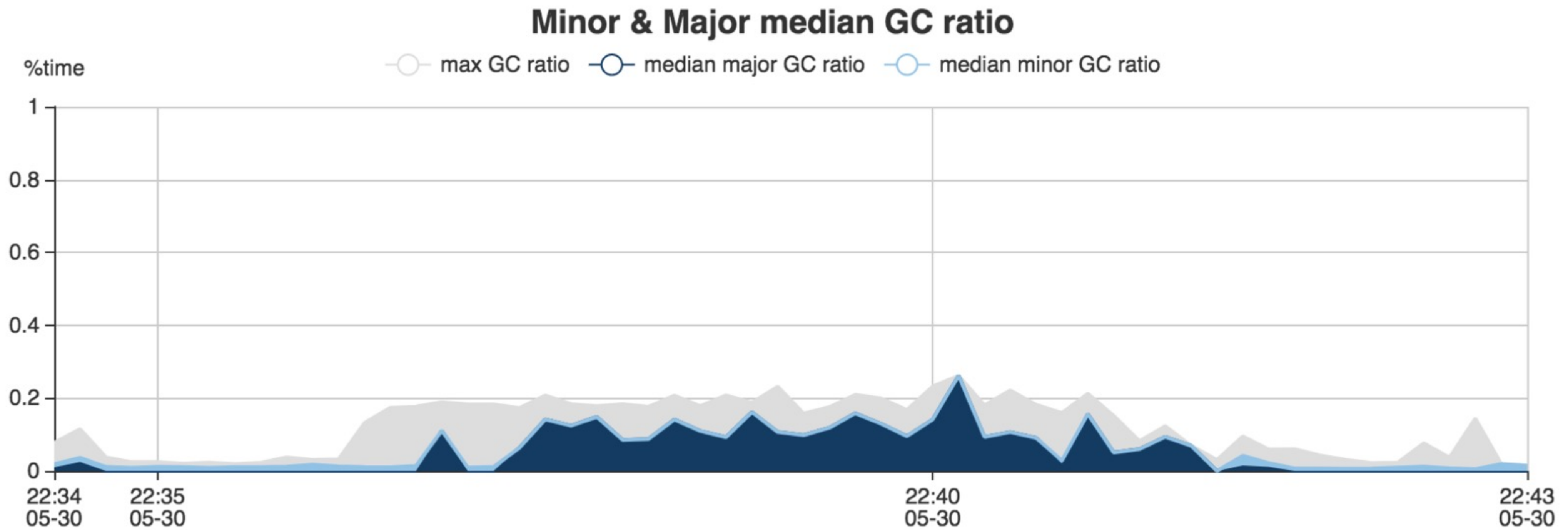
- Avoiding OutOfMemoryErrors (`spark.executor.memory`)
- Avoiding YARN killing containers for exceeding reserved memory (`spark.yarn.executor.memoryOverhead`)



# In practice: Memory

## Dimensioning and tuning memory

Keeping Garbage Collection under control



# Recap

Babar helped us a lot, we hope it can help you too!

https://github.com/criteo/babar

# THANK YOU!

criteo.