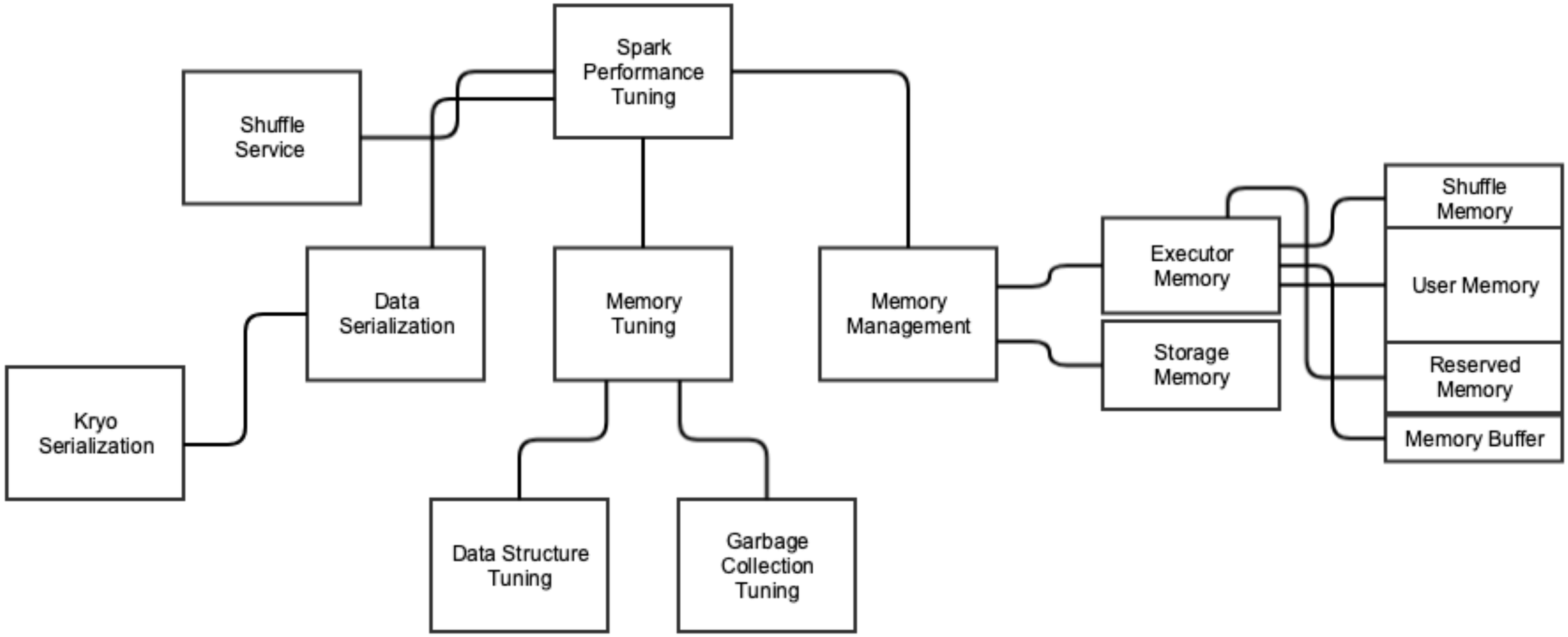
High Performance Spark

Chandra



Introduction

Storage performance

1. Small and Tiny Files Problems

- Small files not only wastes the block size on the filesystem, also causes Node manager failures due to excessive increase in the journal size.
- Also, there will be many round trips when reading the data files from the executors, causing lot of round trips and file permission validations etc., in reading the data in internal I/O and memory.
- So, always limit the file size to the block size of the file system. This varies differently in the case of block stores.

2. Malformed Partitions

3. Under Partitioning

4. Over Partitioning

Memory Management

Memory is classified into

- Storage Memory
- Execution Memory for Spark.

- The datasets are cached or persisted into the Storage Memory,
- Execution memory is used for shuffles, joins, sorts and other aggregations.

Enabling off heap:

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

Note ::-> Escape Garbage Collection using OffHeap

Dynamic Executor Allocation

Dynamic Executor allocation will be used for dynamically allocating the resources without actually reserving the resources during the job execution.

```
spark.dynamicAllocation.enabled = true
spark.dynamicAllocation.executorIdleTimeout = 3m
spark.dynamicAllocation.initialExecutors = 1
spark.dynamicAllocation.minExecutors = 2
spark.dynamicAllocation.maxExecutors = 50
spark.dynamicAllocation.cachedExecutorIdleTimeout = 2m
```

Internal Joins

- **Broadcast Join**
- **Hash Join**
- **Sort Merge Join**
-

Dataset Clustering

Clustering SQL

```
SET spark.sql.shuffle.partitions = 50
SELECT * FROM table1 CLUSTER BY cust_numbr
```

Formula

```
CLUSTER BY == DISTRIBUTE BY + SORT  == repartitionByKey + sortWithinPartitions
```

Clustering_scala

```
df.repartitionBy($cust_numbr).sortWithinPartitions()
```

Analyzed Logical Plan

```
== Analyzed Logical Plan ==
newcinsfx: string, org: string, logo: stri
Project [newcinsfx#1704, org#1710, logo#171
+- SubqueryAlias sqltextinfo015
   +- Sort [acct#1711 ASC NULLS FIRST], fa
      +- RepartitionByExpression [acct#171
         +- Project [newcinsfx#1704, org#17
            +- Join LeftOuter, ((cust_nbr#
               :- Join LeftOuter, (((acct#
               :  :- SubqueryAlias a
               :  +- Union
               :     :- Project [org#171
               :     :  +- Filter ((((m
               :     :     +- SubqueryA
               :     :        +- HiveTa
               :     +- Project [org#181
               :        +- Filter ((((m
               :           +- SubqueryA
```

Physical Plan

```
== Physical Plan ==
*(7) Sort [acct#1711 ASC NULLS FIRST], false,
+- Exchange hashpartitioning(acct#1711, 50)
   +- *(6) Project [newcinsfx#1704, org#1710,
      +- *(6) BroadcastHashJoin [cust_nbr#1947
         :- *(6) Project [org#1710, logo#1713,
         :  +- SortMergeJoin [acct#1711, org#1
         :     :- *(3) Sort [acct#1711 ASC NUL
         :     :  +- Exchange hashpartitioning
         :     :     +- Union
         :     :        :- *(1) Filter ((((isr
         :     :        :  +- HiveTableScan [c
         :     :        +- *(2) Filter ((((isr
         :     :           +- HiveTableScan [c
         :     +- *(4) Sort [acct#1944 ASC NUL
         :        +- Exchange hashpartitioning
         :           +- HiveTableScan [org#194
         +- BroadcastExchange HashedRelationBr
            +- *(5) Project [org#2192, acct#21
               +- *(5) Filter ((((isnotnull(en
                  +- HiveTableScan [acct#2193,
```

Optimized Logical Plan

```
== Optimized Logical Plan ==
Sort [acct#1711 ASC NULLS FIRST], false
+- RepartitionByExpression [acct#1711], 50
   +- Project [newcinsfx#1704, org#1710, logo#1713,
      +- Join LeftOuter, ((cust_nbr#1703 = cust_nbr
         :- Project [org#1710, logo#1713, acct#1711
         :  +- Join LeftOuter, (((((acct#1711 = acc
         :     :- Union
         :     :  :- Project [org#1710, logo#1713,
         :     :  :  +- Filter ((((isnotnull(mt_eff
         :     :  :     +- HiveTableRelation `bip_s
         :     :  +- Project [org#1827, logo#1830,
         :     :     +- Filter ((((isnotnull(mt_eff
         :     :        +- HiveTableRelation `bip_s
         :     +- Project [org#1943, logo#1945, acc
         :        +- HiveTableRelation `bip_sdb`.`s
         +- Project [org#2192, acct#2193 AS cust_nb
            +- Filter ((((isnotnull(end_dt#2268) &&
               +- HiveTableRelation `bip_sdb`.`s_d1
```

Broadcasting the Joins

When we have to join smaller datasets with large files, we avoid the shuffle and broadcast the data to all the executors , so data is available in the spark memory directly.
When using spark sql we have spark.sql.autoJoinBroadcastThreshold and is set to -1, but you can configure that to a value which you can afford for broadcasting to the executor.
There are various ways of doing a broadcast :

```
spark.sql.autoJoinBroadcastThresold = 10mb
customer.join(broadcast(districts),customer_distric
t_code=district_code)

select /*+ BROADCAST(D) */ from customer c join
districts D on C.customer_district_code =
D.district_code
```

•

Tuning the Garbage Collector

JVM supports different types of garbage collectors : G1GC, CMSGC, Parallel GC.

```
-XX:+UseG1GC -XX:+PrintFlagsFinal -XX:
+PrintReferenceGC -verbose:gc -XX:+PrintGCDetails -
XX:+PrintGCTimeStamps -XX:
+PrintAdaptiveSizePolicy -XX:
+UnlockDiagnosticVMOptions -XX:+G1SummarizeConcMark
-Xms25g -Xmx25g -
XX:InitiatingHeapOccupancyPercent=35 -
XX:ConcGCThread=10
```

Incase of very huge workloads, ideally we should move to ParallelGC setting.

CBO

spark.sql.cbo.enabled = true
spark.sql.cbo.joinReorder.enabled = true
spark.sql.cbo.histogram.enabled = true

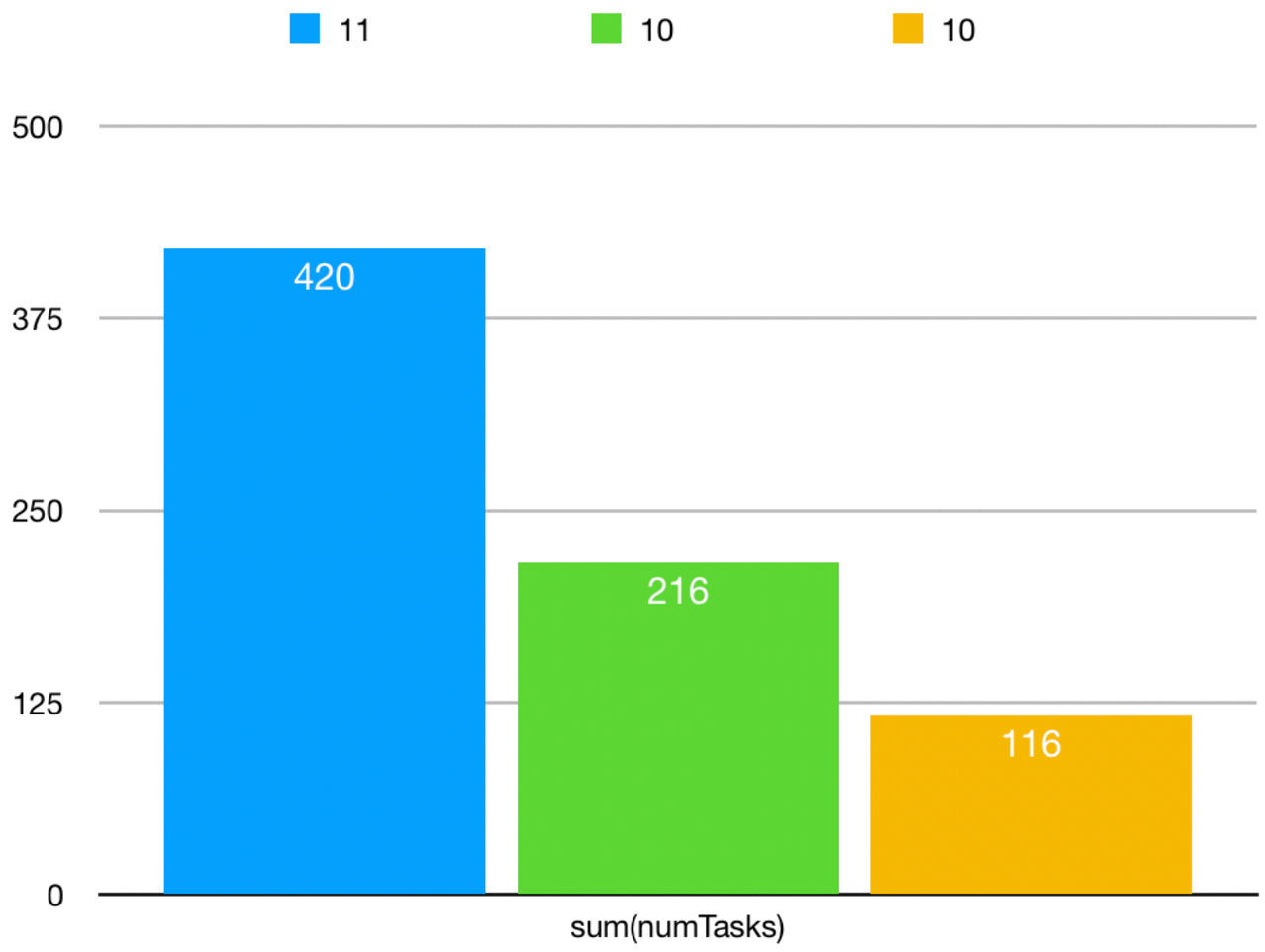
Computing Stats

- **Table Statistics**
    - Analyse table <TABLE_NAME> COMPUTE STATISTICS;
- **Column Statistics**
    - Analyse table <TABLE_NAME> COMPUTE STATISTICS COLUMN column1,column2;
- Partition Statistics

Tuning Comparisons

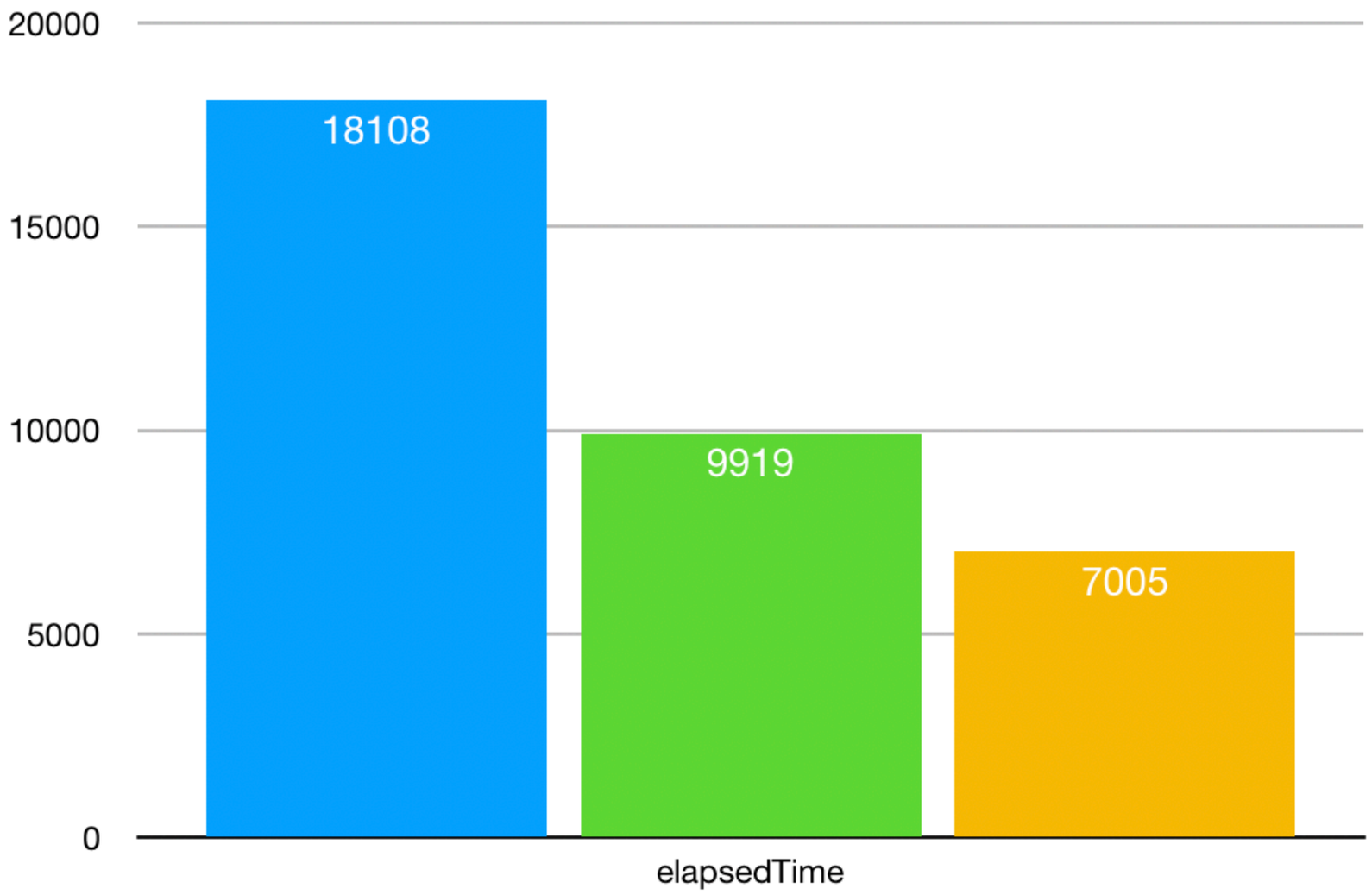
- Plain SQL with Dynamic Allocation Enabled (a)
- (a) + Cost Based Optimizer Enabled (b)
- (a) + (b) + Clustering Key Enabled

- Dataset1 (6 Months - 10 GBs)
- Dataset2 (6 Months - Smaller Dataset)
- Dataset3 (6 Months - Medium Size 1 GB)



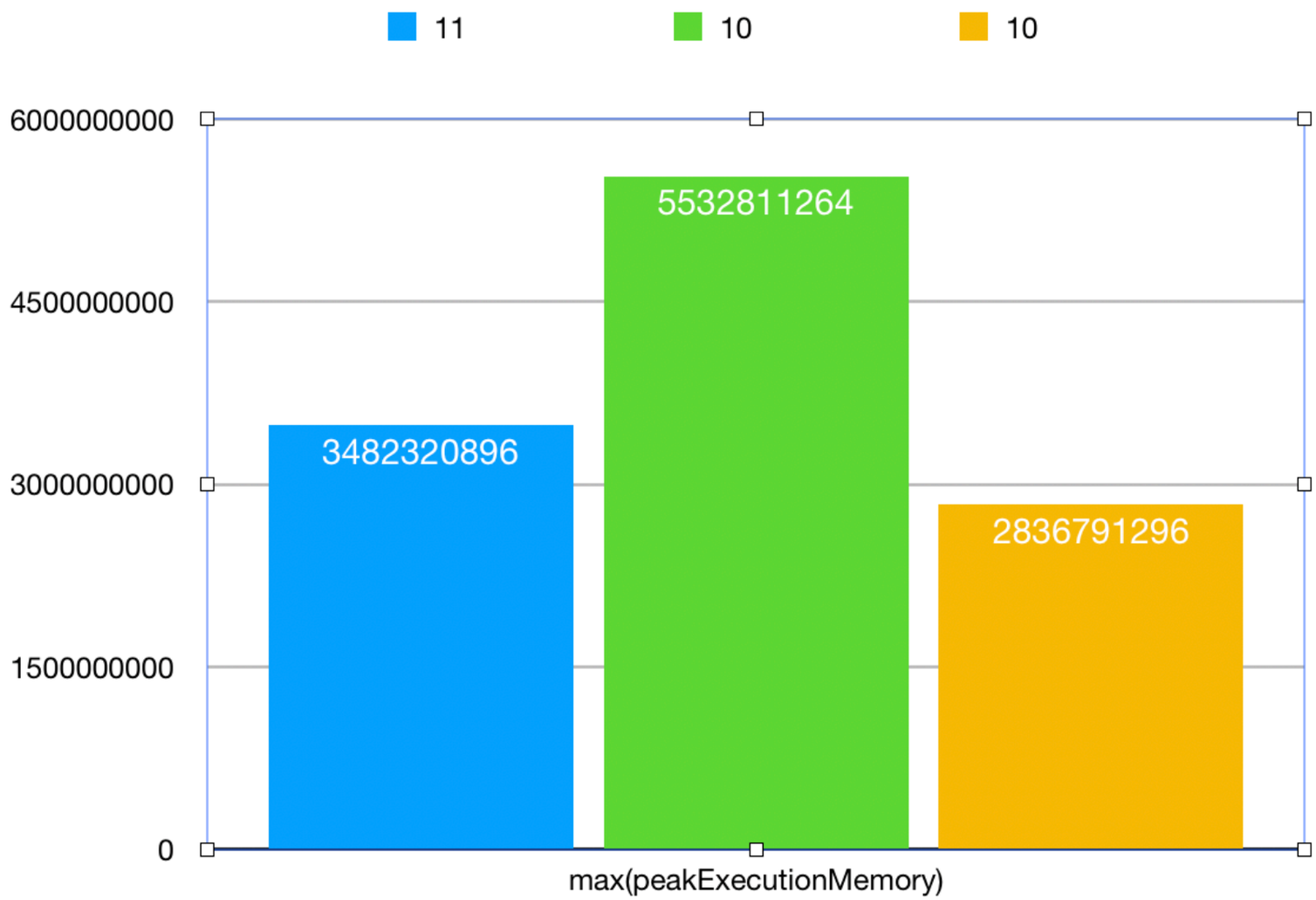
Tuning Comparisons

Num Tasks



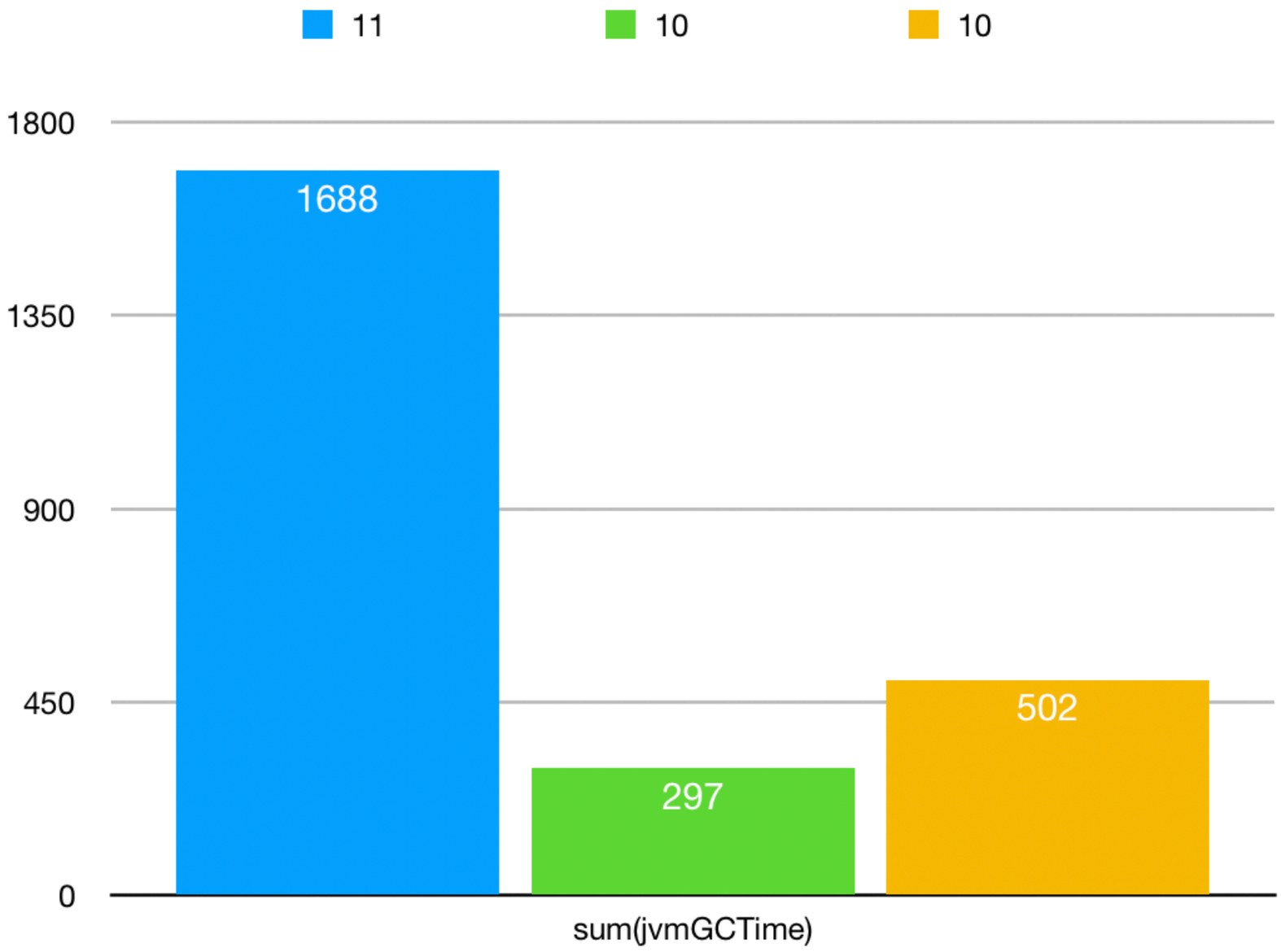
Tuning Comparisons

Elapsed time



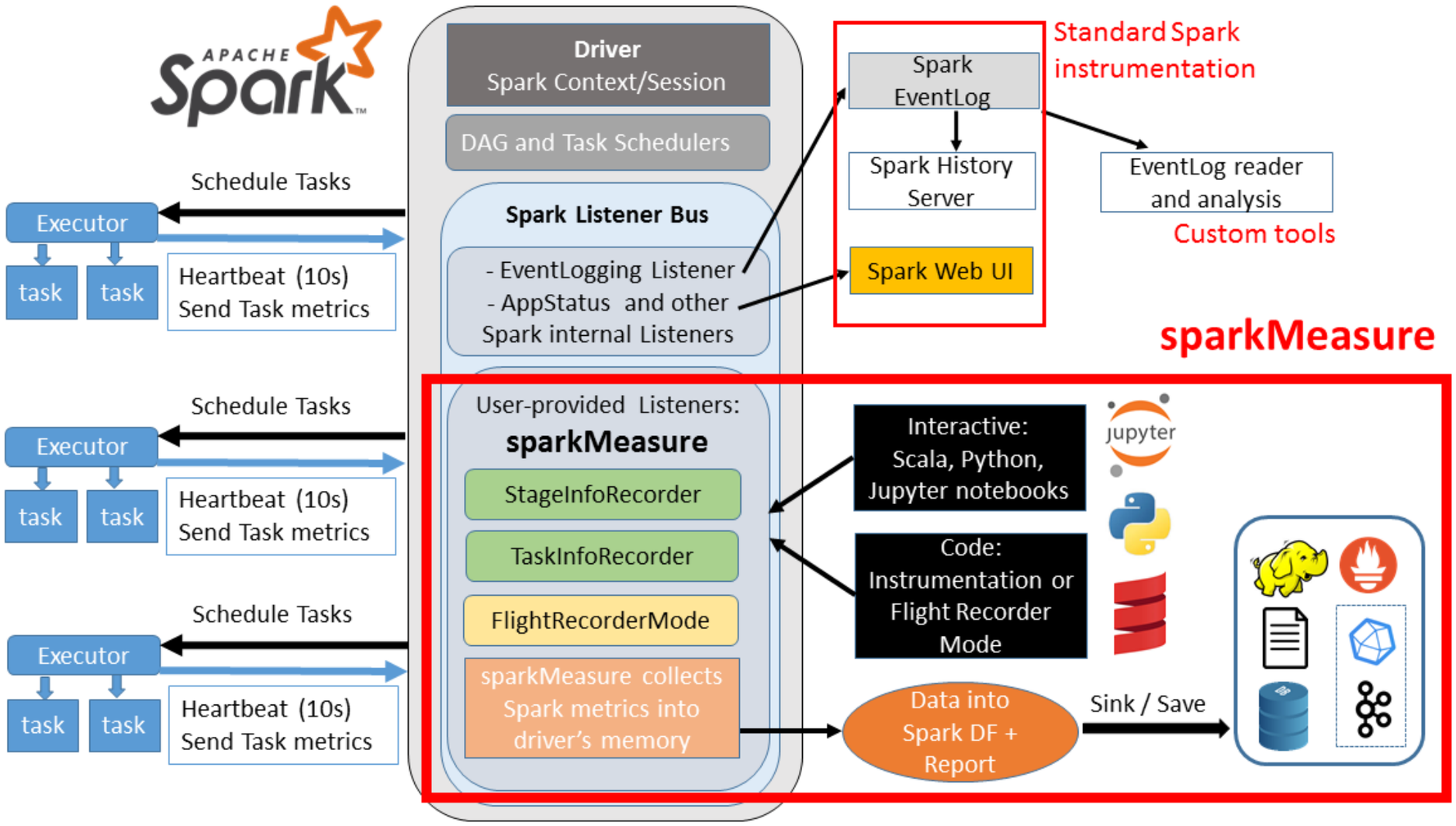
Tuning Comparisons

Execution Memory



Tuning Comparisons

JVM GC Time



Instrumentation

Spark Measure (Courtesy: CERN)