Build Markov transition matrix with Mapreduce and Spark

CISC849
Weiyang Chen

Content

The problem

Input

- The dataset is a set of transaction log files, the format of the log record is:
  Customer_id, **Transaction_id**, **Transaction_type**

- The transaction id values are unique and have the semantics of positioning a transaction along a time line. Each log file will contain some transactions of different customers. For example:

TJS41P2U9A, 97351659821043, LHS
R8NYNNV4NP, 97358894512859, MHN
WRBQZUT3G9, 9735950520184 7, LNS
NLJT50JA7K, 97363549658528, HHN
0VT2QJN9Q2, 97370708307103, LHN

- Each transaction is encoded by the following three quantities and expressed as a 3-letter token.
  Amount spent: Low, Normal, or High
  Whether the transaction includes high price ticket item: Normal or High
  Time elapsed since the last transaction: Large, Normal, or Small

Output

Customer_id1: Transition Matrix1
Customer_id2: Transition Matrix2
.....

future

present	LNL	MNL	HNL	LHL	.....
LNL					
MNL		0.001	0.342		
HNL					
.....					

Strategy

1. Read from hdfs

TJS41P2U9A, 97351659821043, LHS
R8NYNNV4NP, 97358894512859, MHN
WRBQZUT3G9, 97359505201847, LNS
NLJT50JA7K, 97363549658528, HHN
0VT2QJN9Q2, 97370708307103, LHN
……

2. Group by Customer_id

TJS41P2U9A, 97358894512859, LHS
TJS41P2U9A, 97351659821043, MHN
TJS41P2U9A, 97359505201847, LNS
……
NLJT50JA7K, 97363549658528, HHN
NLJT50JA7K, 97370708307103, LHS
……

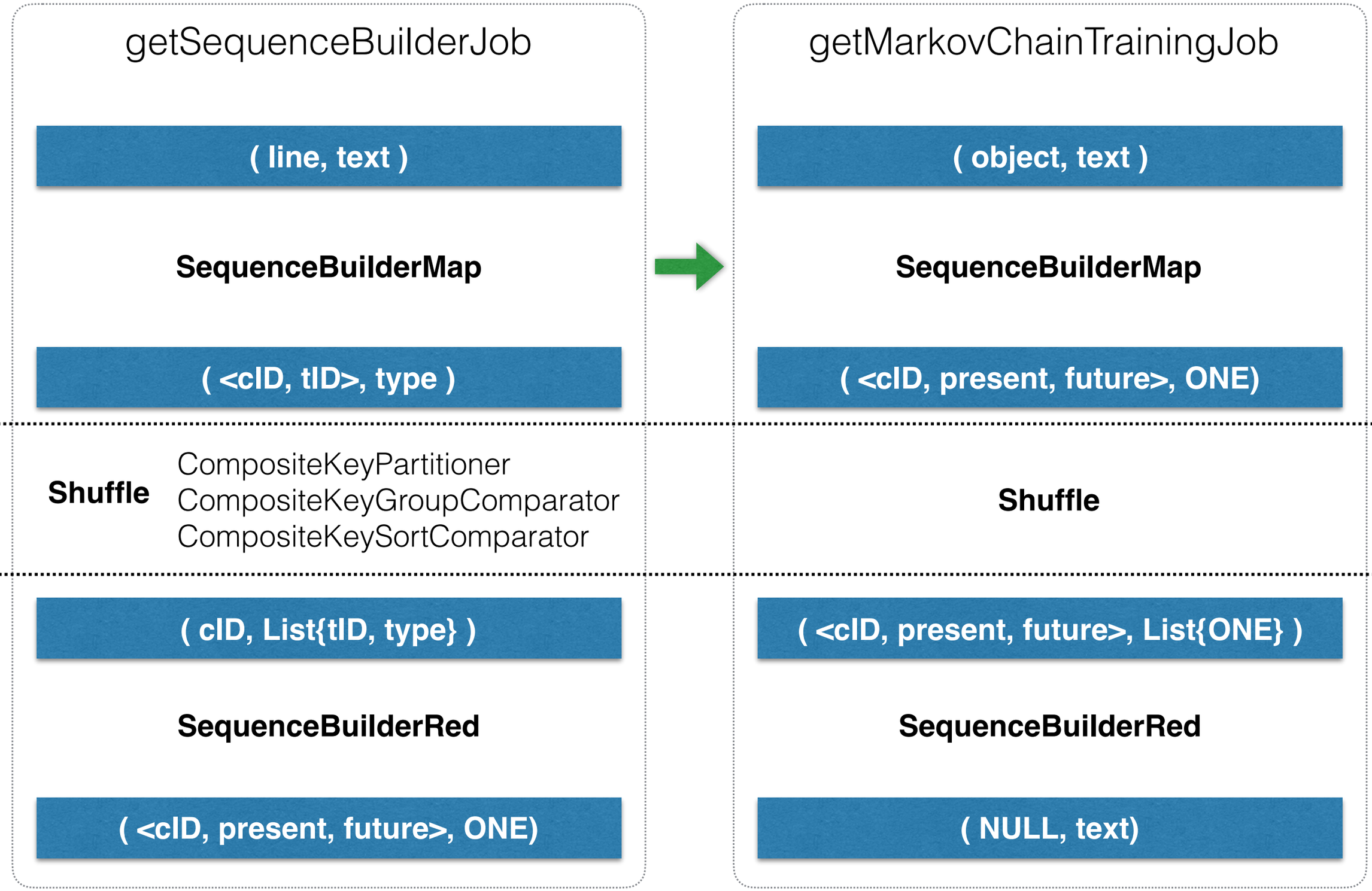
3. Sort Transaction_id

TJS41P2U9A, 97351659821043, LHS
97358894512859, MHN
97359505201847, LNS
……
NLJT50JA7K, 97363549658528, HHN
97370708307103, LHS
……

4. Build matrix

	LNL	MNL	HNL	LHL	…
LNL					
MNL					
HNL					
…					

Build with Mapreduce



Build with Spark

Stage 1

JavaRDD **<String>**
lines = sc.textFile(sortFile)

// Read file from hdfs

JavaPairRDD<CustomerID, Tuple2<TransationID, TransationType>>

JavaPairRDD **<String, Tuple2<Long, String>>**
pairs = lines.mapToPair

Shuffle

JavaPairRDD **<String, Iterable<Tuple2<Long, String>>>**
groups = pairs.groupByKey(40)

// Group RDDs by CustomerID

Stage 2

JavaPairRDD **<String, Iterable<Tuple2<Long, String>>>**
sorted = groups.mapValues

// Sort RDDs by TransationID

JavaPairRDD **<String, double[][]>**
rawMatrix = sorted.mapValues

// Build matrix for each customer

JavaPairRDD **<String, String>**
cookedMatrix = rawMatrix.mapValues

// Convert matrix to string

cookedMatrix.saveAsTextFile

// Save result to hdfs

Test Environment

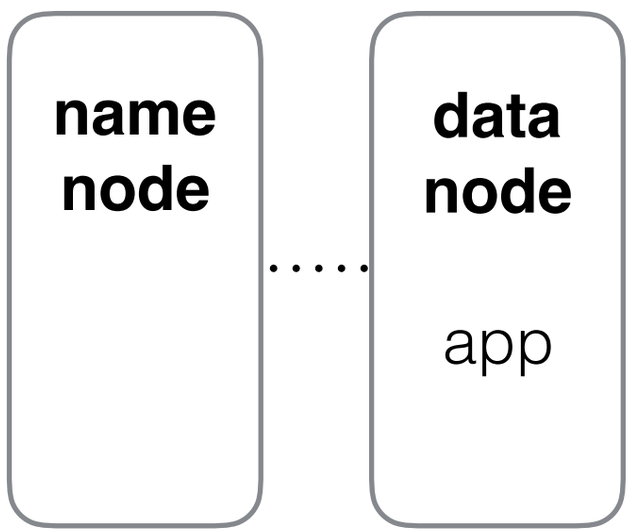
MacBook Pro
2.6 GHz Intel Core i5 with L3 shared cache
8 GB 1600 MHz DDR3
128G SSD

VM - CentOS 7
2 Cores
4096 MB memory
20G Storage

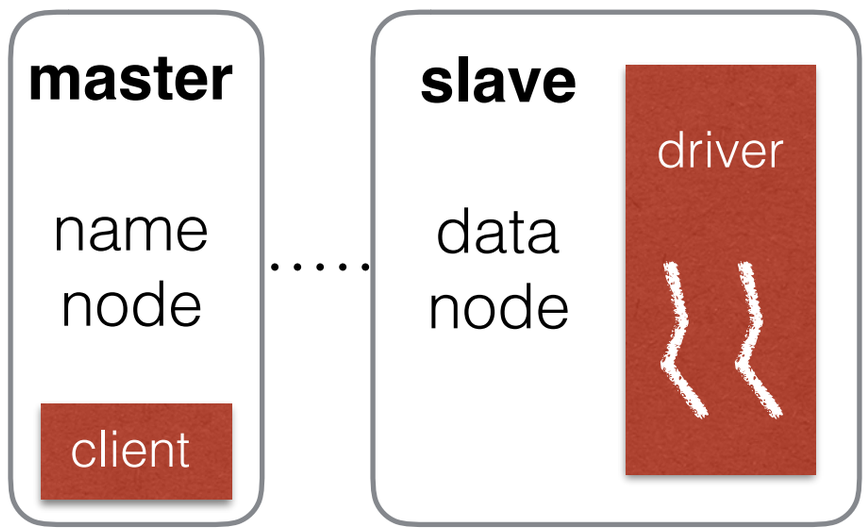
Mapreduce
OpenJDK 1.8.0
Maven 3.3.9
Hadoop 2.6.0

Spark
OpenJDK 1.8.0
Maven 3.3.9
Hadoop 2.6.0
Spark 1.6.1

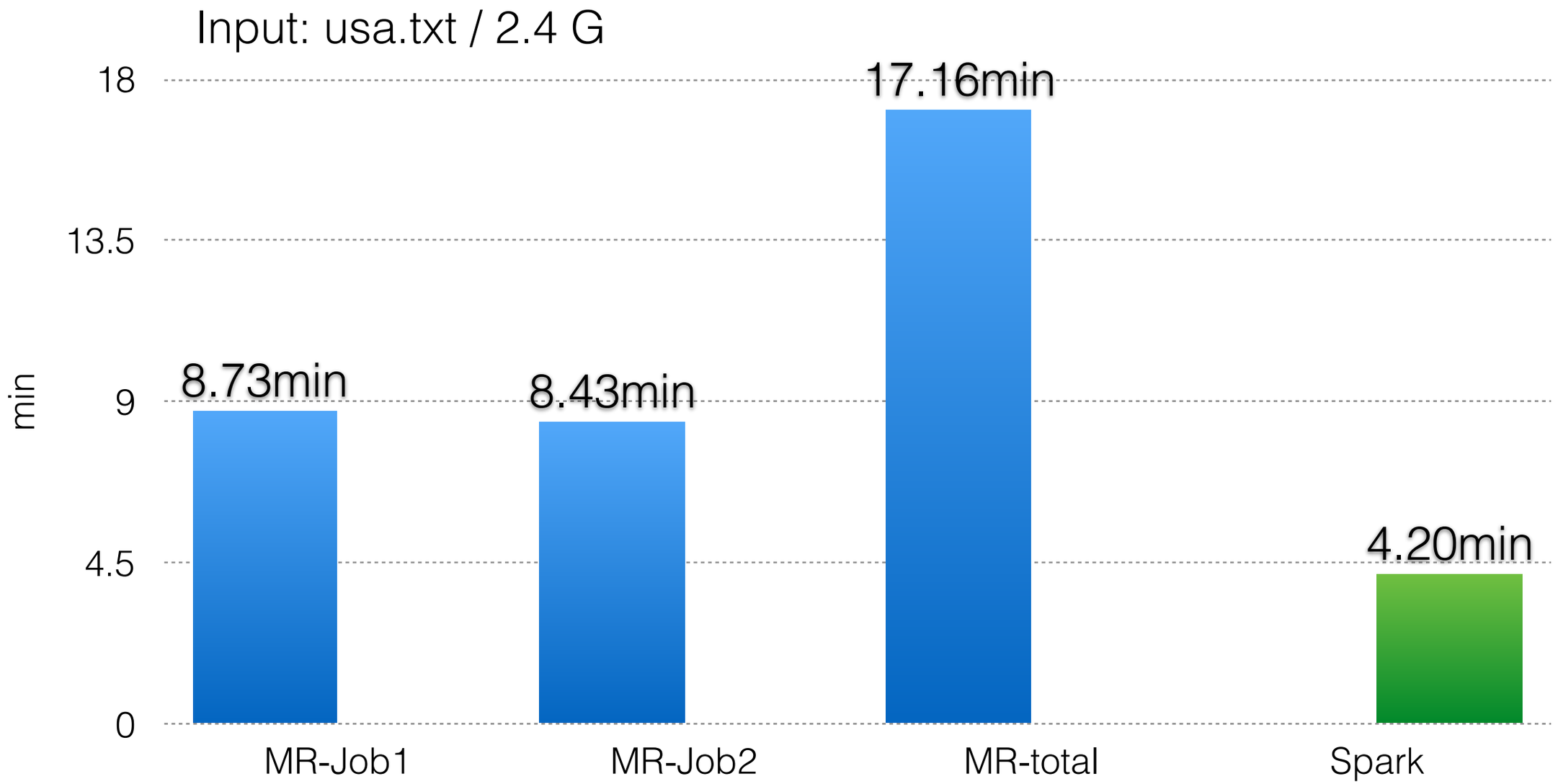
Hadoop - Yarn
hdfs



Spark - Standalone
hdfs



Result: Time Taken

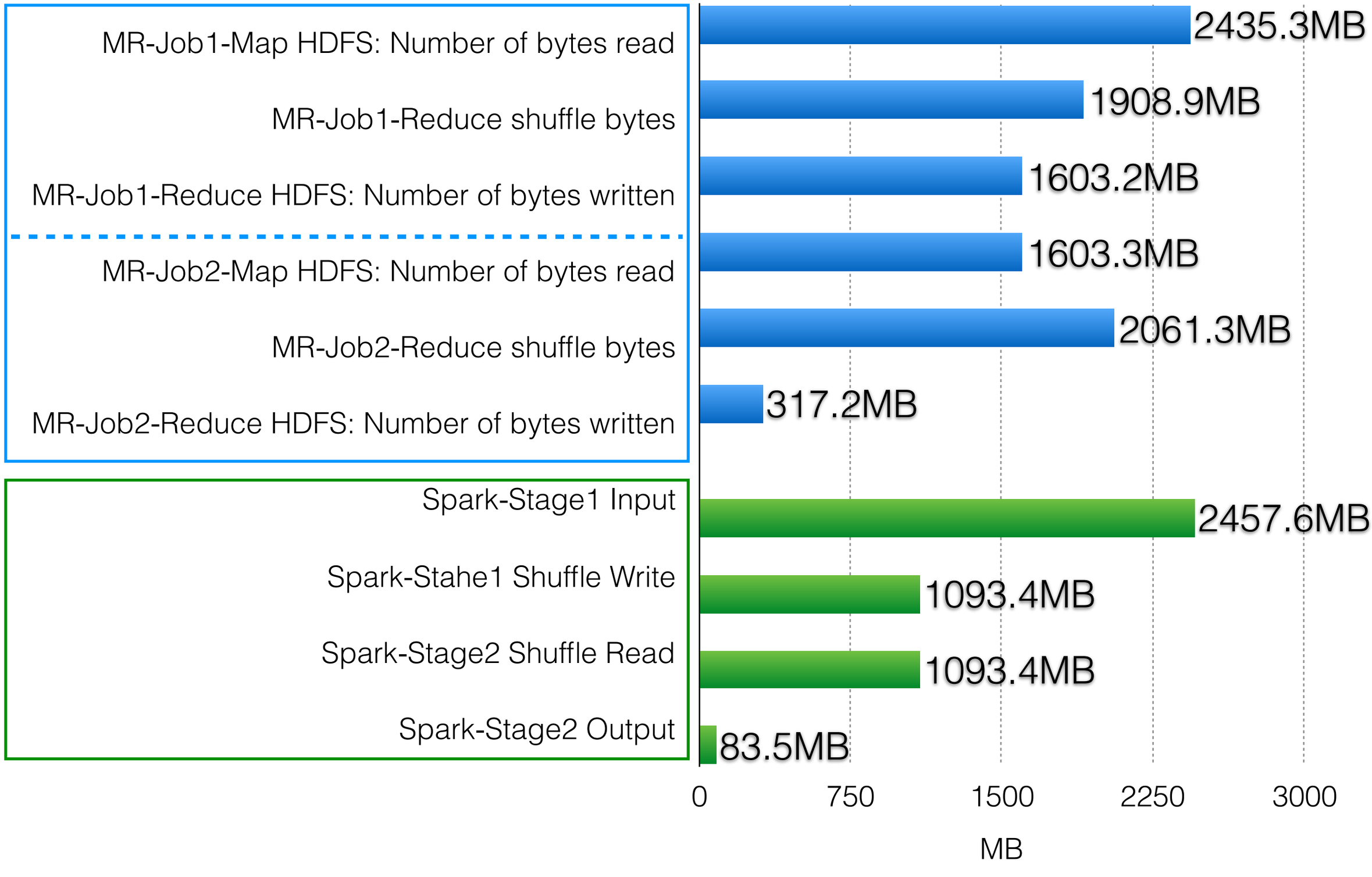


MR-Job1
19 Mappers
4 Reducers

MR-Job2
15 Mappers
4 Reducers

Spake
Stage 1: 19 tasks
Stage 2: 40 taskss

Result: I/O


MR-Job1-Map HDFS: Number of bytes read
MR-Job1-Reduce shuffle bytes
MR-Job1-Reduce HDFS: Number of bytes written
MR-Job2-Map HDFS: Number of bytes read
MR-Job2-Reduce shuffle bytes
MR-Job2-Reduce HDFS: Number of bytes written
Spark-Stage1 Input
Spark-Stahe1 Shuffle Write
Spark-Stage2 Shuffle Read
Spark-Stage2 Output
2435.3MB
1908.9MB
1603.2MB
1603.3MB
2061.3MB
317.2MB
2457.6MB
1093.4MB
1093.4MB
83.5MB
0
750
1500
2250
3000
MB

Conclusion