A Comparative Performance Evaluation of Flink

Dongwon Kim

POSTECH

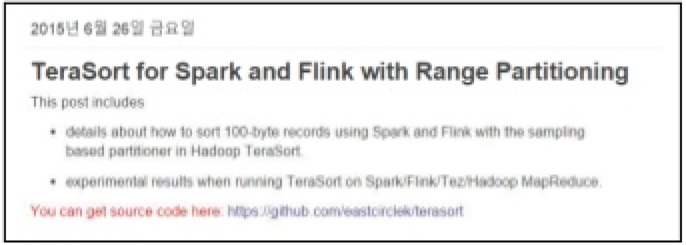
About Me

- Postdoctoral researcher @ POSTECH

- Research interest
  - Design and implementation of **distributed systems**
  - Performance optimization of **big data processing engines**

- Doctoral thesis
  - MR2: **Fault Tolerant MapReduce** with **the Push Model**

- Personal blog
  - http://eastcirclek.blogspot.kr
  - Why I'm here ☺



2015년 6월 26일 금요일

TeraSort for Spark and Flink with Range Partitioning

This post includes

- details about how to sort 100-byte records using Spark and Flink with the sampling based partitioner in Hadoop TeraSort.
- experimental results when running TeraSort on Spark/Flink/Tez/Hadoop MapReduce.

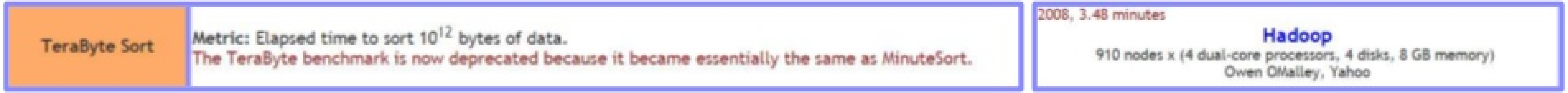
You can get source code here: https://github.com/eastcirclek/terasort

Outline

- **TeraSort for various engines**
- Experimental setup
- Results & analysis
- What else for better performance?
- Conclusion

TeraSort

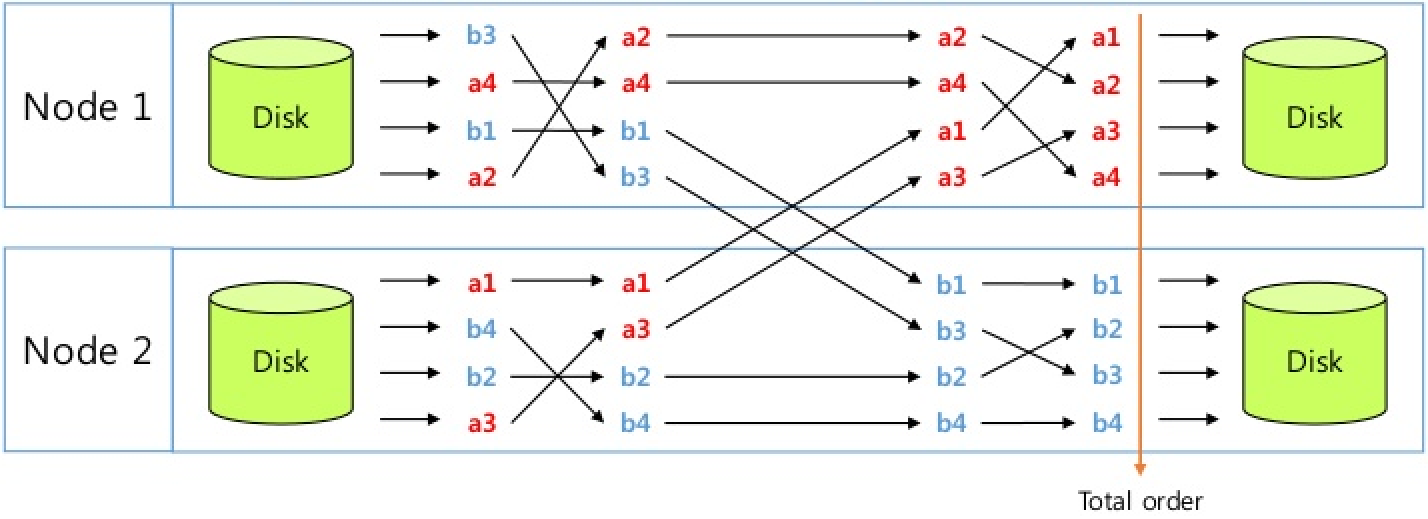
- Hadoop MapReduce program for the annual terabyte sort competition



TeraByte Sort	Metric: Elapsed time to sort 10^{12} bytes of data. The TeraByte benchmark is now deprecated because it became essentially the same as MinuteSort.	2008, 3.48 minutes **Hadoop** 910 nodes x (4 dual-core processors, 4 disks, 8 GB memory) Owen OMalley, Yahoo

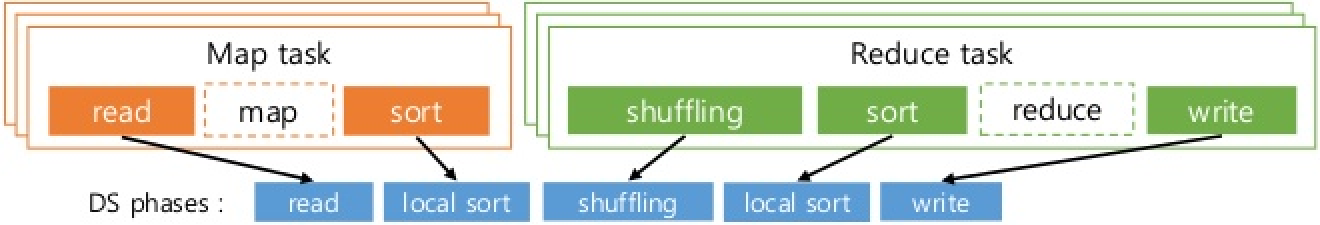
- TeraSort is essentially **distributed sort (DS)**

Typical DS phases : read | local sort | shuffling | local sort | write

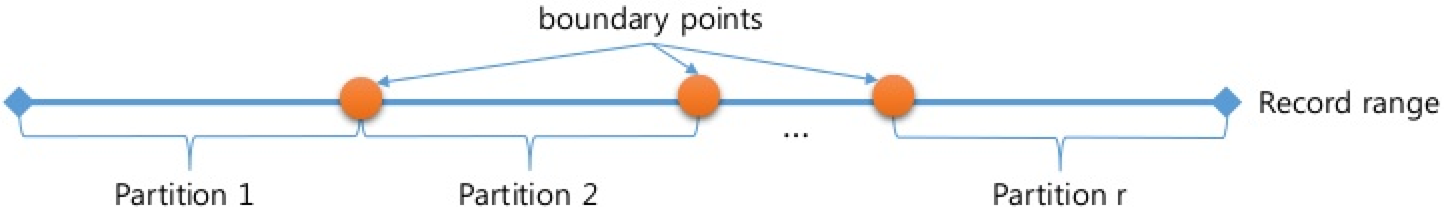


TeraSort for MapReduce

- Included in Hadoop distributions
  - with TeraGen & TeraValidate
- **Identity map & reduce functions**

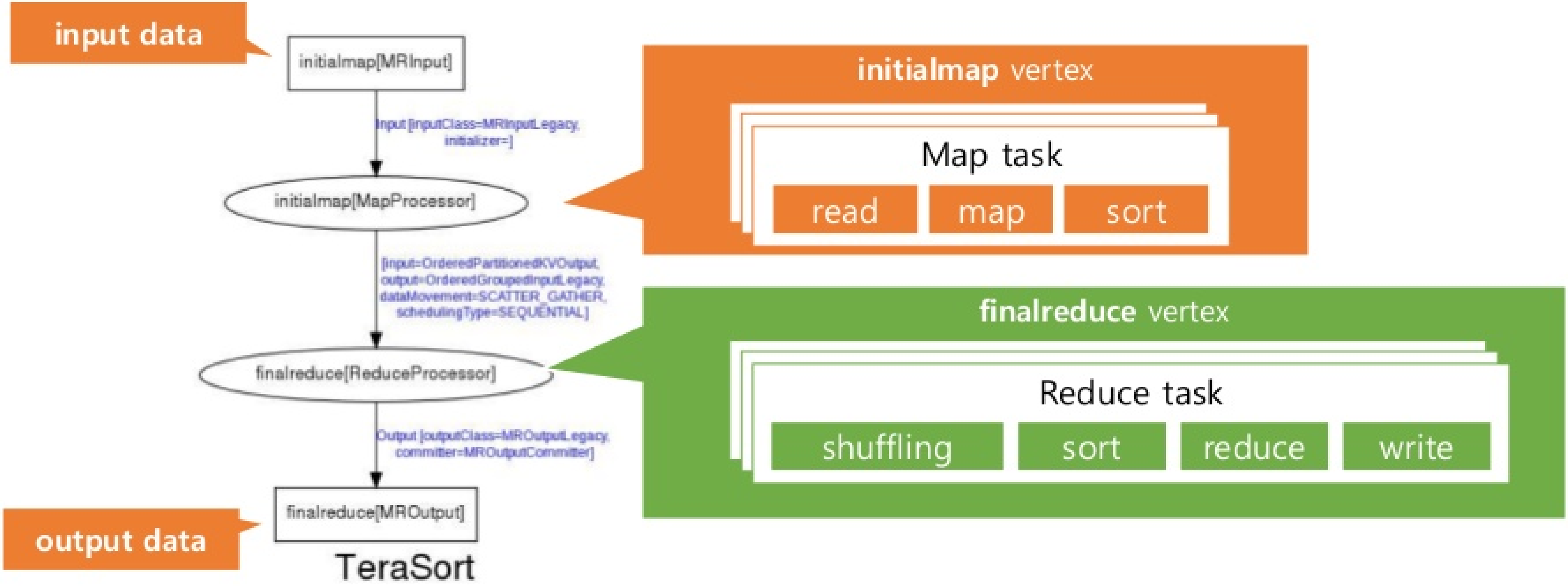


- Range partitioner built on sampling
  - To guarantee a total order & to prevent partition skew
  - Sampling to compute boundary points within few seconds



TeraSort for Tez

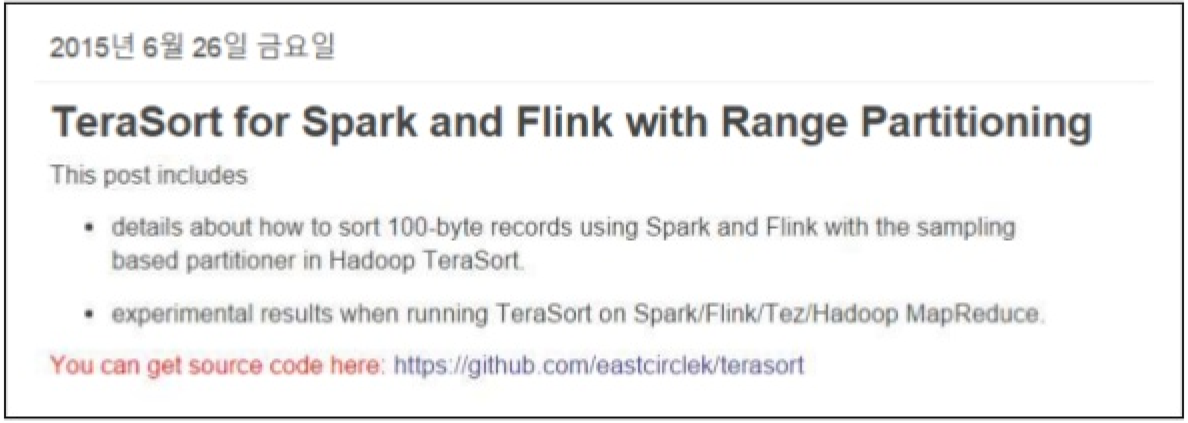
- **Tez** can execute **TeraSort for MapReduce** w/o any modification
  - mapreduce.framework.name = yarn-tez
- Tez DAG plan of **TeraSort for MapReduce**


input data
initialmap[MRInput]
Input [inputClass=MRInputLegacy, initializer=]
initialmap[MapProcessor]
[input=OrderedPartitionedKVOutput, output=OrderedGroupedInputLegacy, dataMovement=SCATTER_GATHER, schedulingType=SEQUENTIAL]
finalreduce[ReduceProcessor]
Output [outputClass=MROutputLegacy, committer=MROutputCommitter]
finalreduce[MROutput]
output data
TeraSort
initialmap vertex
Map task
read
map
sort
finalreduce vertex
Reduce task
shuffling
sort
reduce
write

TeraSort for Spark & Flink

- My source code in GitHub:
  - https://github.com/eastcirclek/terasort

- **Sampling-based range partitioner** from TeraSort for MapReduce
  - Visit my personal blog for a detailed explanation
  - http://eastcirclek.blogspot.kr



2015년 6월 26일 금요일

TeraSort for Spark and Flink with Range Partitioning

This post includes

- details about how to sort 100-byte records using Spark and Flink with the sampling based partitioner in Hadoop TeraSort.
- experimental results when running TeraSort on Spark/Flink/Tez/Hadoop MapReduce.

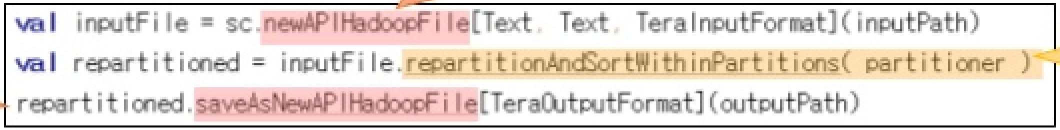
You can get source code here: https://github.com/eastcirclek/terasort

TeraSort for Spark

- Code

Create a new RDD to read from HDFS

partitions = # blocks

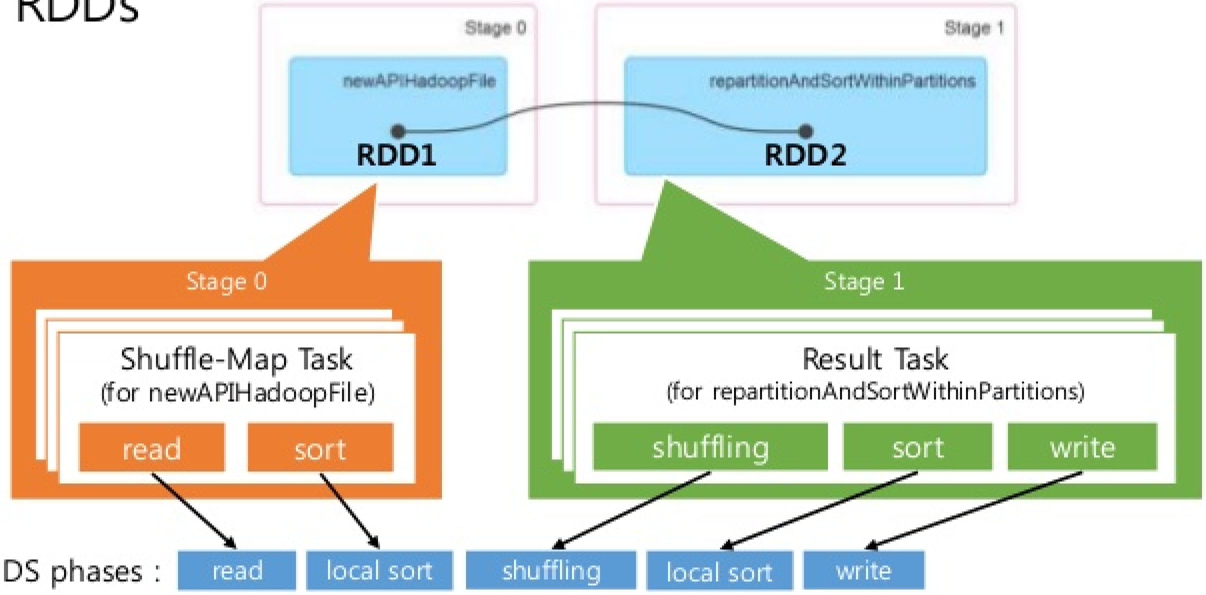


```
val inputFile = sc.newAPIHadoopFile[Text, Text, TeraInputFormat](inputPath)
val repartitioned = inputFile.repartitionAndSortWithinPartitions( partitioner )
repartitioned.saveAsNewAPIHadoopFile[TeraOutputFormat](outputPath)
```

Repartition the parent RDD
based on the user-specified partitioner

Write output to HDFS

- Two RDDs



Stage 0

Shuffle-Map Task
(for newAPIHadoopFile)

read | sort

Stage 1

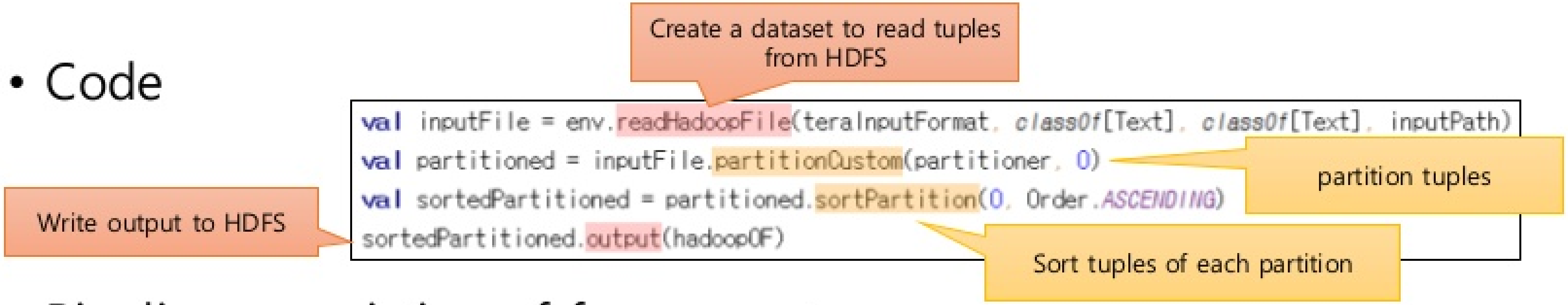
Result Task
(for repartitionAndSortWithinPartitions)

shuffling | sort | write

DS phases : read | local sort | shuffling | local sort | write

TeraSort for Flink

- Code



```
val inputFile = env.readHadoopFile(teraInputFormat, classOf[Text], classOf[Text], inputPath)
val partitioned = inputFile.partitionCustom(partitioner, 0)
val sortedPartitioned = partitioned.sortPartition(0, Order.ASCENDING)
sortedPartitioned.output(hadoopOF)
```

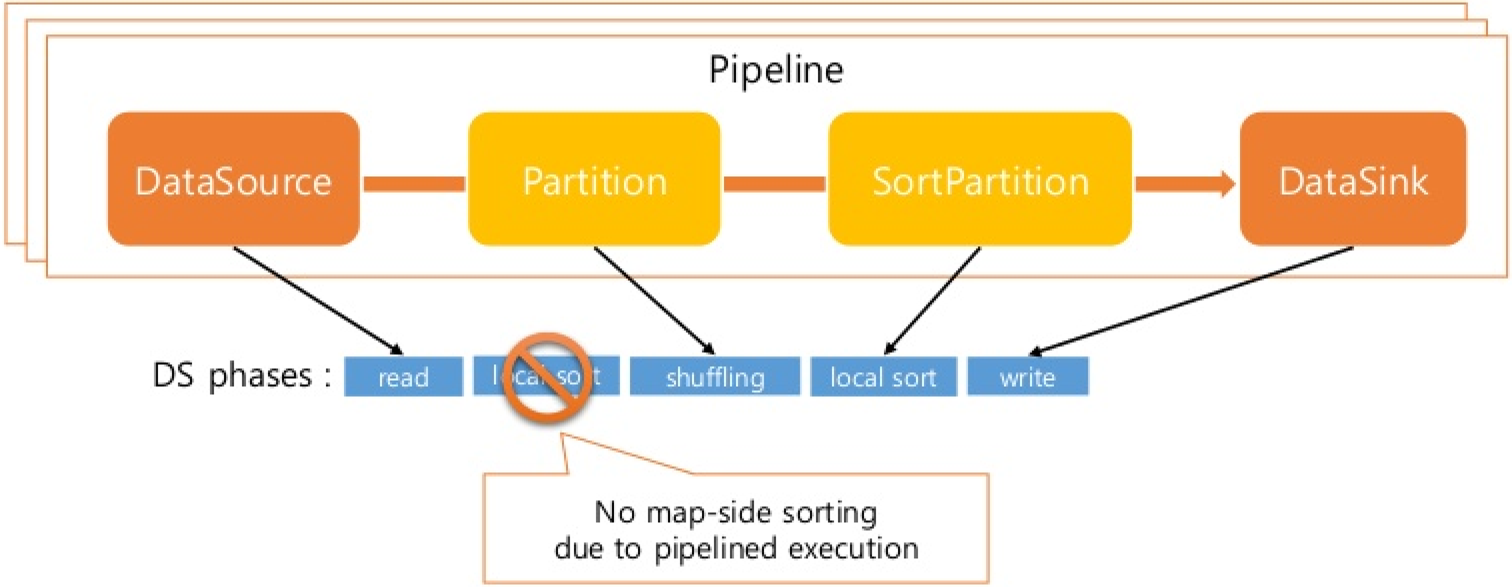
Create a dataset to read tuples from HDFS

partition tuples

Sort tuples of each partition

Write output to HDFS

- Pipelines consisting of four operators

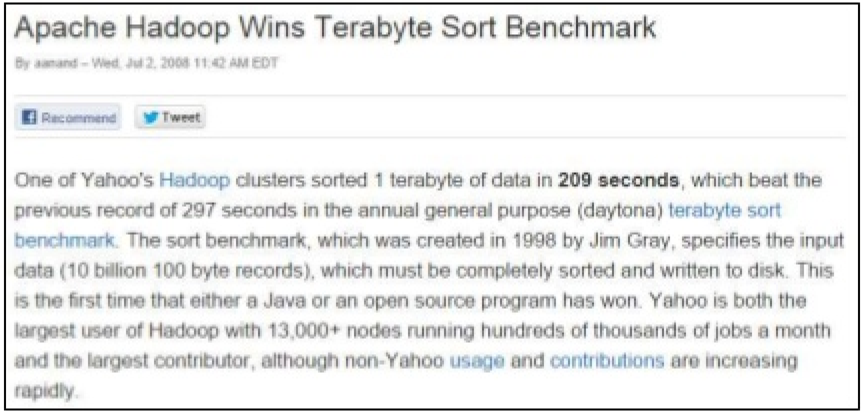


No map-side sorting due to pipelined execution

Importance of TeraSort

- Suitable for measuring the **pure performance** of big data engines
  - No data transformation (like map, filter) with user-defined logic
  - **Basic facilities of each engine are used**

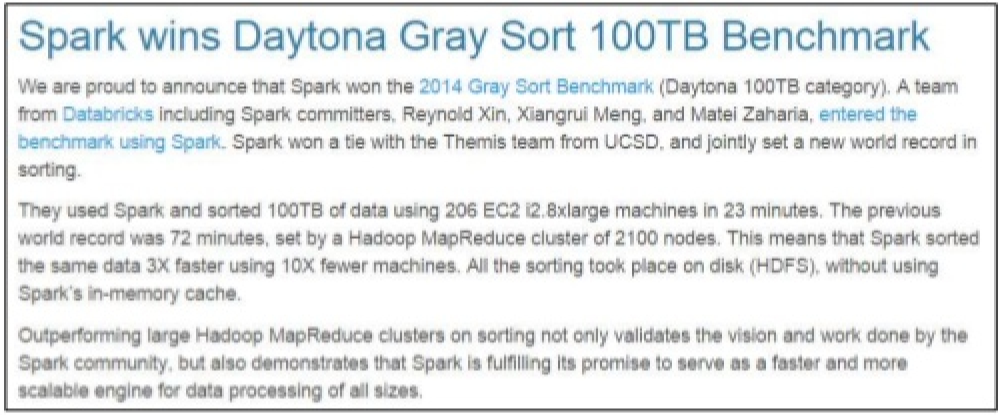
- "Winning the sort benchmark" is a great means of PR



Apache Hadoop Wins Terabyte Sort Benchmark

By aanand – Wed, Jul 2, 2008 11:42 AM EDT

One of Yahoo's Hadoop clusters sorted 1 terabyte of data in **209 seconds**, which beat the previous record of 297 seconds in the annual general purpose (daytona) terabyte sort benchmark. The sort benchmark, which was created in 1998 by Jim Gray, specifies the input data (10 billion 100 byte records), which must be completely sorted and written to disk. This is the first time that either a Java or an open source program has won. Yahoo is both the largest user of Hadoop with 13,000+ nodes running hundreds of thousands of jobs a month and the largest contributor, although non-Yahoo usage and contributions are increasing rapidly.



Spark wins Daytona Gray Sort 100TB Benchmark

We are proud to announce that Spark won the 2014 Gray Sort Benchmark (Daytona 100TB category). A team from Databricks including Spark committers, Reynold Xin, Xiangrui Meng, and Matei Zaharia, entered the benchmark using Spark. Spark won a tie with the Themis team from UCSD, and jointly set a new world record in sorting.

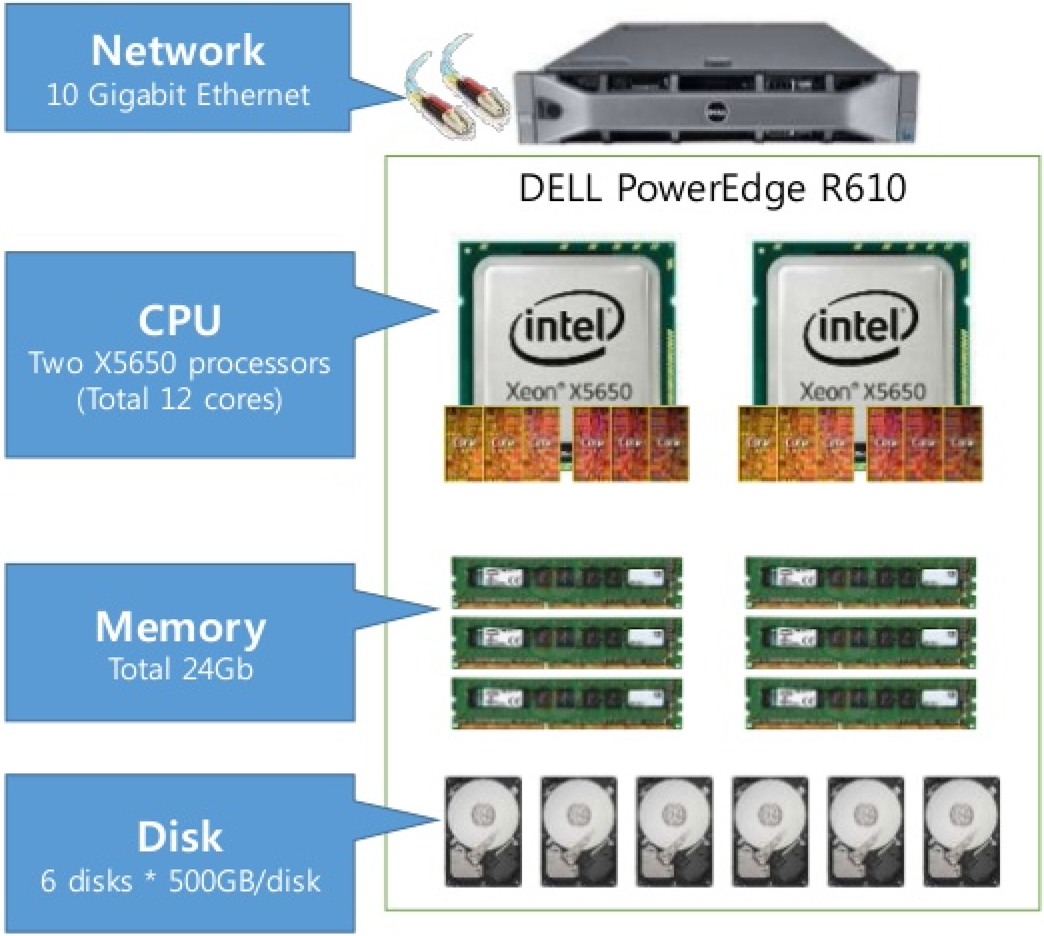
They used Spark and sorted 100TB of data using 206 EC2 i2.8xlarge machines in 23 minutes. The previous world record was 72 minutes, set by a Hadoop MapReduce cluster of 2100 nodes. This means that Spark sorted the same data 3X faster using 10X fewer machines. All the sorting took place on disk (HDFS), without using Spark's in-memory cache.

Outperforming large Hadoop MapReduce clusters on sorting not only validates the vision and work done by the Spark community, but also demonstrates that Spark is fulfilling its promise to serve as a faster and more scalable engine for data processing of all sizes.

Outline

- TeraSort for various engines
- **Experimental setup**
  - Machine specification
  - Node configuration
- Results & analysis
- What else for better performance?
- Conclusion

Machine specification (42 identical machines)



Network
10 Gigabit Ethernet

DELL PowerEdge R610

CPU
Two X5650 processors
(Total 12 cores)

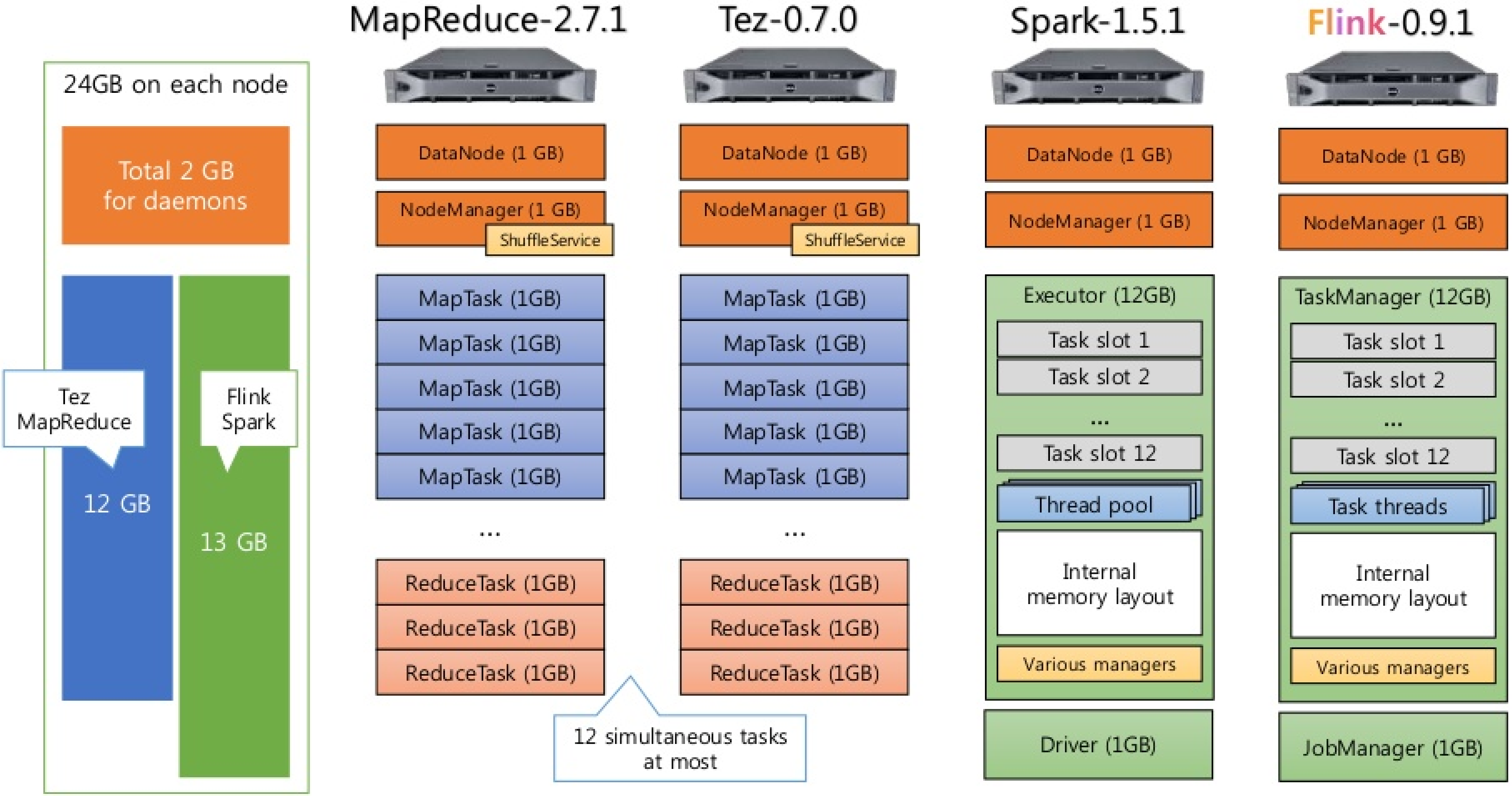
Memory
Total 24Gb

Disk
6 disks * 500GB/disk

Results can be different in newer machines

	My machine	Spark team
Processor	Intel Xeon X5650 (Q1, 2010)	Intel Xeon E5-2670 (Q1, 2012)
Cores	6 * 2 processors	8 * 4 processors
Memory	24GB	244GB
Disks	6 HDD's	8 SSD's

Node configuration



24GB on each node

Total 2 GB for daemons

Tez MapReduce: 12 GB

Flink Spark: 13 GB

MapReduce-2.7.1

- DataNode (1 GB)
- NodeManager (1 GB)
  - ShuffleService
- MapTask (1GB)
- MapTask (1GB)
- MapTask (1GB)
- MapTask (1GB)
- MapTask (1GB)
- …
- ReduceTask (1GB)
- ReduceTask (1GB)
- ReduceTask (1GB)

Tez-0.7.0

- DataNode (1 GB)
- NodeManager (1 GB)
  - ShuffleService
- MapTask (1GB)
- MapTask (1GB)
- MapTask (1GB)
- MapTask (1GB)
- MapTask (1GB)
- …
- ReduceTask (1GB)
- ReduceTask (1GB)
- ReduceTask (1GB)

12 simultaneous tasks at most

Spark-1.5.1

- DataNode (1 GB)
- NodeManager (1 GB)
- Executor (12GB)
  - Task slot 1
  - Task slot 2
  - …
  - Task slot 12
  - Thread pool
  - Internal memory layout
  - Various managers
- Driver (1GB)

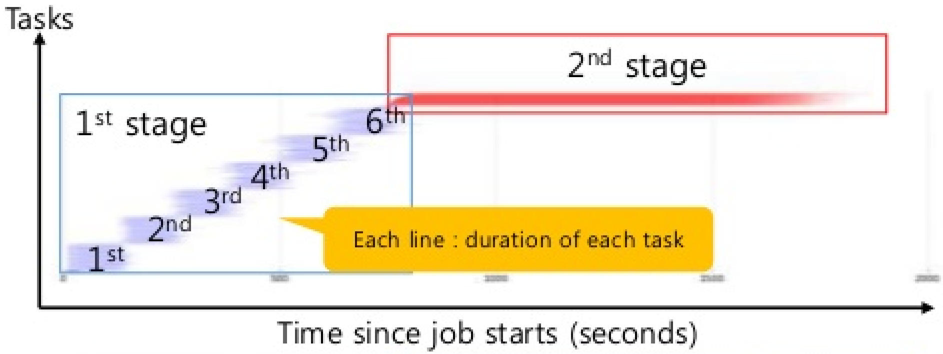
Flink-0.9.1

- DataNode (1 GB)
- NodeManager (1 GB)
- TaskManager (12GB)
  - Task slot 1
  - Task slot 2
  - …
  - Task slot 12
  - Task threads
  - Internal memory layout
  - Various managers
- JobManager (1GB)

Outline

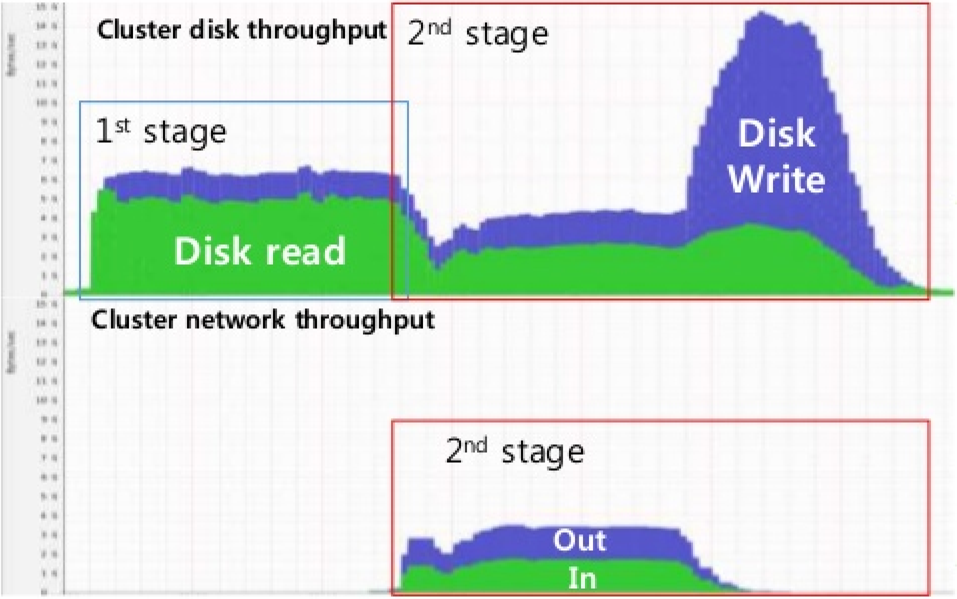
- TeraSort for various engines
- Experimental setup
- **Results & analysis**
  - **Flink** is faster than other engines due to its **pipelined execution**
- What else for better performance?
- Conclusion

How to read a swimlane graph & throughput graphs



- 6 waves of 1st stage tasks
- 1 wave of 2nd stage tasks

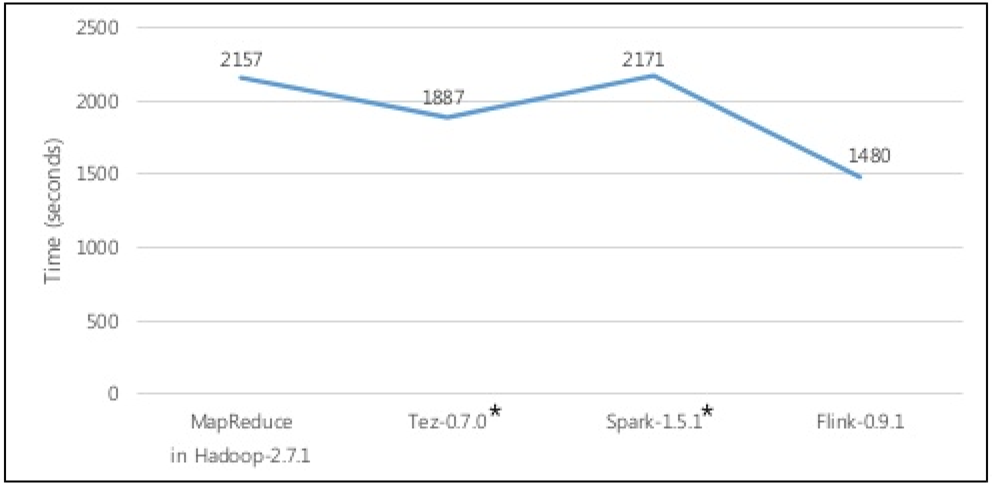
- Two stages are hardly overlapped

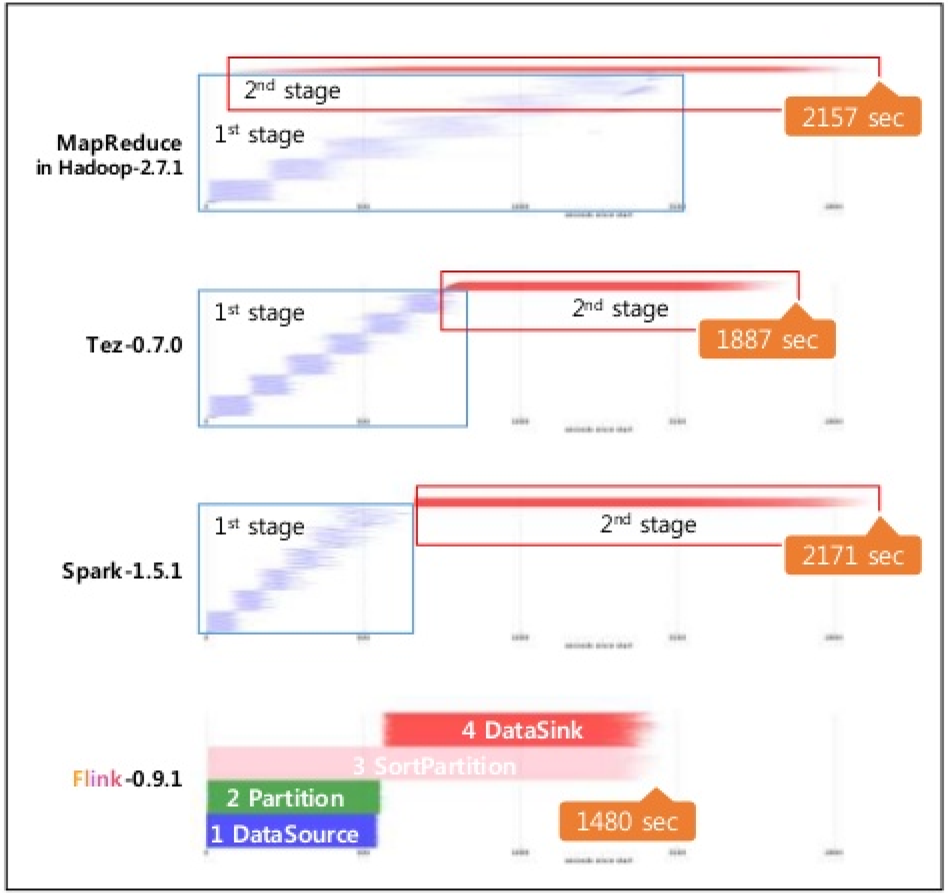


Different patterns for different stages

No network traffic during 1st stage

Result of sorting 80GB/node (3.2TB)

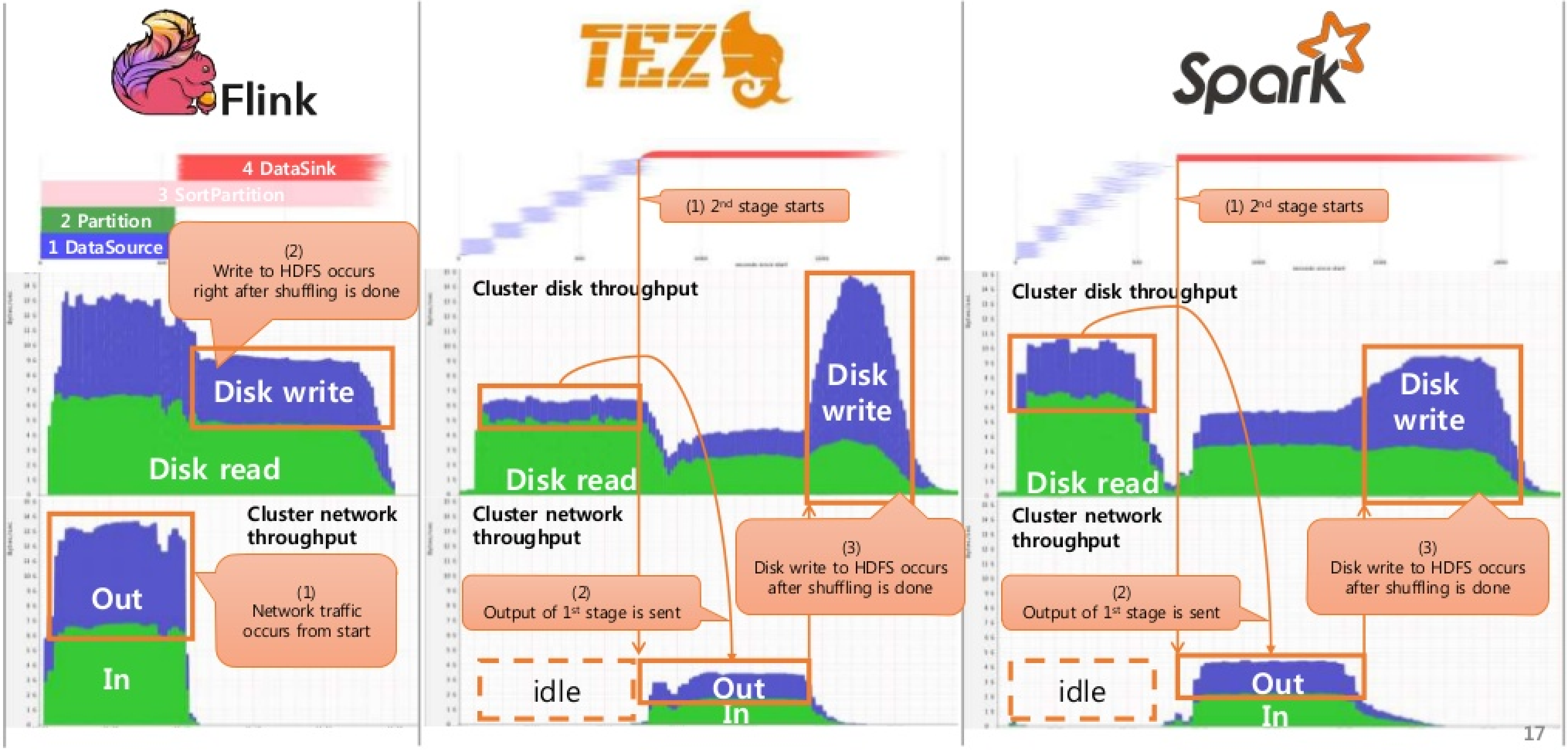




- **Flink** is the fastest due to its **pipelined execution**
  - **Tez** and **Spark** do not overlap 1st and 2nd stages
  - **MapReduce** is slow despite overlapping stages

* **Map output compression turned on for Spark and Tez**

Tez and Spark do not overlap 1st and 2nd stages



Flink:

(1) Network traffic occurs from start

(2) Write to HDFS occurs right after shuffling is done

Tez:

(1) 2nd stage starts

(2) Output of 1st stage is sent

(3) Disk write to HDFS occurs after shuffling is done

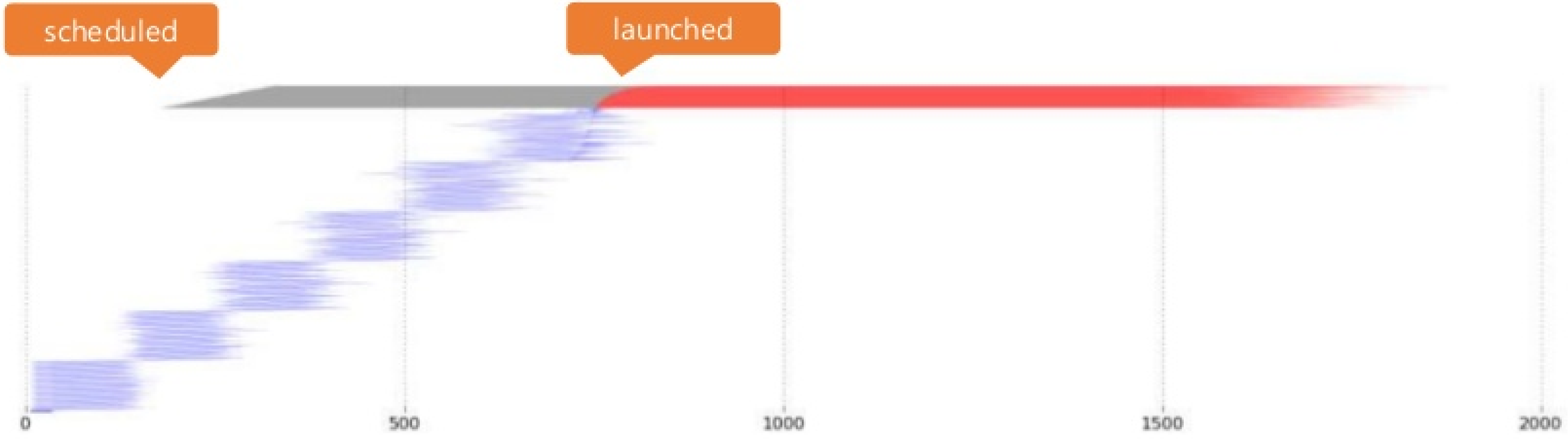
Spark:

(1) 2nd stage starts

(2) Output of 1st stage is sent

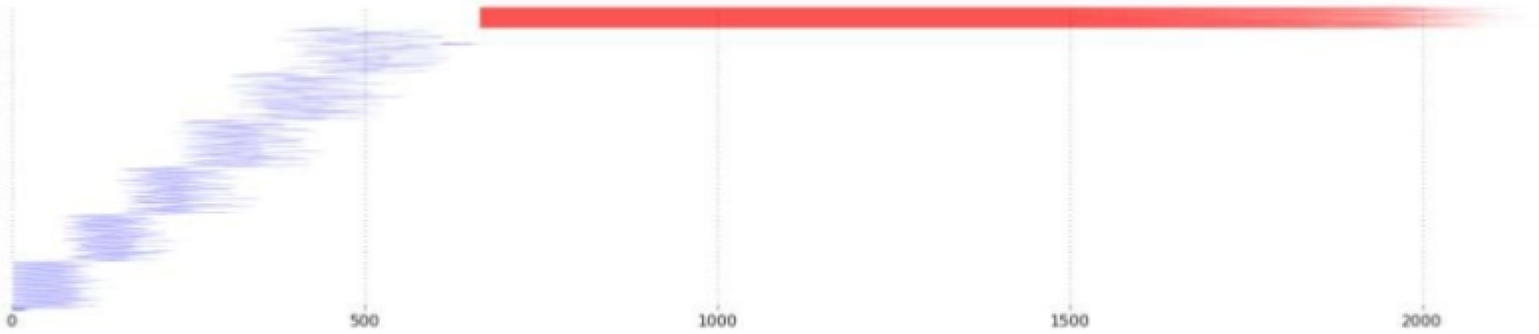
(3) Disk write to HDFS occurs after shuffling is done

Tez does not overlap 1st and 2nd stages

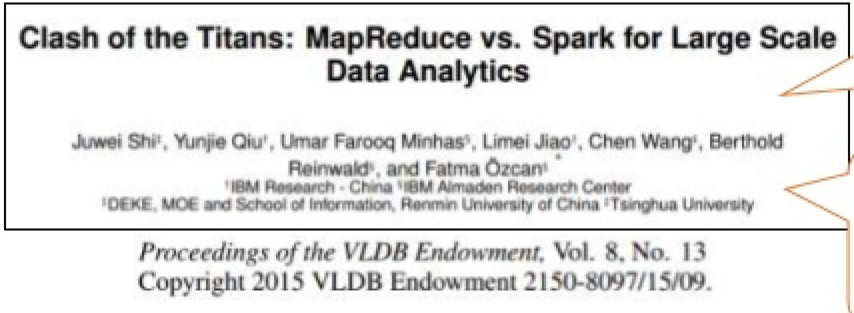


- Tez has parameters to control the degree of overlap
  - tez.shuffle-vertex-manager.min-src-fraction : 0.2
  - tez.shuffle-vertex-manager.max-src-fraction : 0.4
- However, 2nd stage is **scheduled early** but **launched late**

Spark does not overlap 1st and 2nd stages



- Spark cannot execute multiple stages simultaneously
  - also mentioned in the following VLDB paper (2015)



Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics

Juwei Shi[2], Yunjie Qiu[1], Umar Farooq Minhas[3], Limei Jiao[1], Chen Wang[4], Berthold Reinwald[3], and Fatma Özcan[3]

[1]IBM Research - China [3]IBM Almaden Research Center
[2]DEKE, MOE and School of Information, Renmin University of China [4]Tsinghua University

Proceedings of the VLDB Endowment, Vol. 8, No. 13
Copyright 2015 VLDB Endowment 2150-8097/15/09.

Experimental results of this paper

- **Spark** is faster than **MapReduce** for WordCount, K-means, PageRank.
- **MapReduce** is faster than **Spark** for **Sort**.

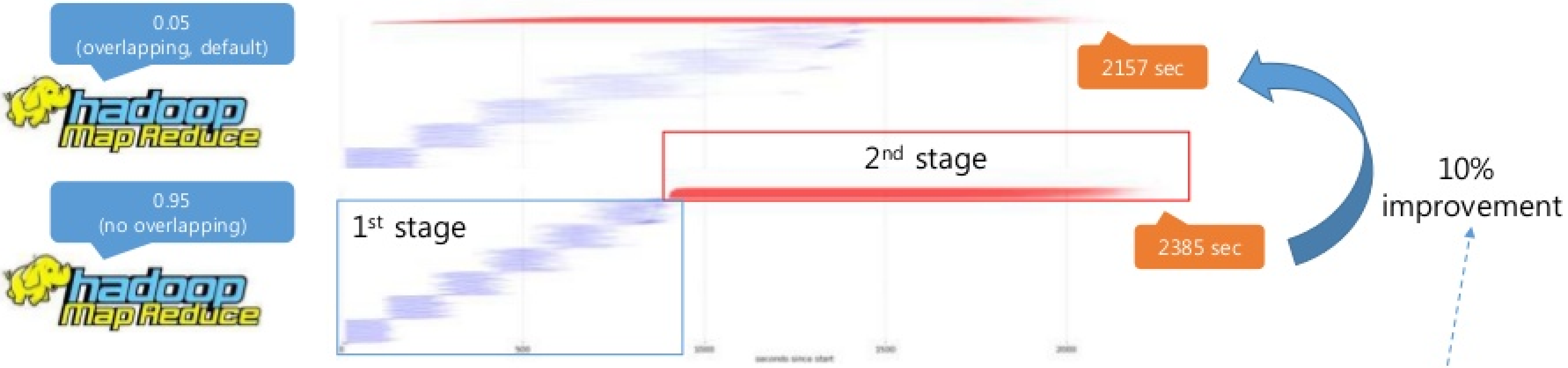
Spark **doesn't support the overlap** between shuffle write and read stages.

…

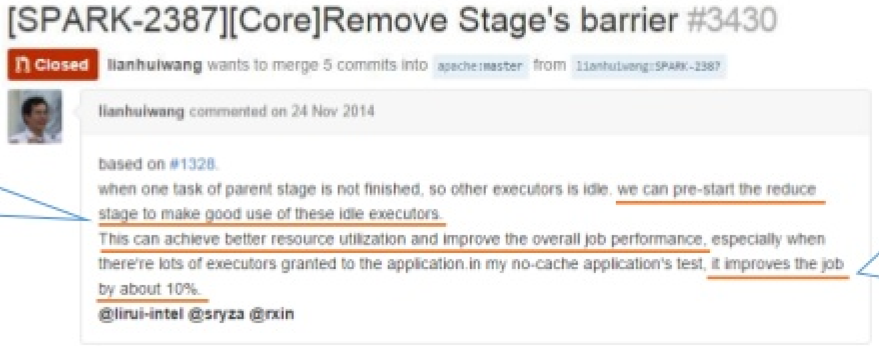
Spark **may want to support this overlap** in the future to improve performance.

MapReduce is slow despite overlapping stages

- mapreduce.job.reduce.slowstart.completedMaps : [0.0, 1.0]



- Wang's attempt to overlap spark stages



[SPARK-2387][Core]Remove Stage's barrier #3430

lianhuiwang wants to merge 5 commits into apache:master from lianhuiwang:SPARK-2387

lianhuiwang commented on 24 Nov 2014

based on #1328.
when one task of parent stage is not finished, so other executors is idle. we can pre-start the reduce stage to make good use of these idle executors.
This can achieve better resource utilization and improve the overall job performance, especially when there're lots of executors granted to the application.in my no-cache application's test, it improves the job by about 10%.
@lirui-intel @sryza @rxin

Wang proposes to overlap stages to achieve better utilization

10%???

Why Spark & MapReduce improve just 10%?