# hadoop & APACHE Spark

## Jose Carlos Alcantara

# Outline

- Brief History
- Ecosystems
- Job processing
- Similarities
- Advantages/Disadvantages
- Benchmark
- MR Tuner
- Wrap Up

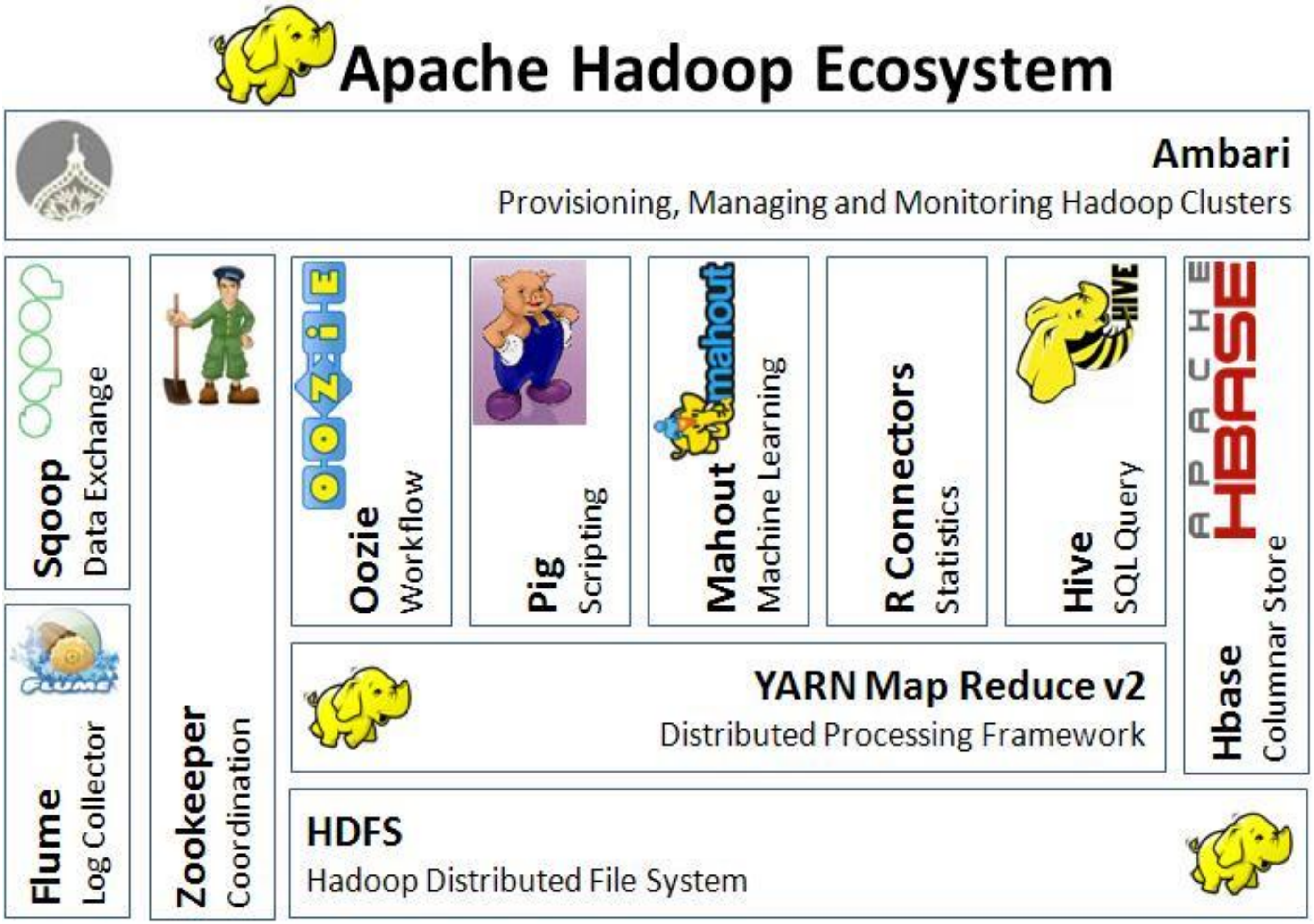
# Brief History: Hadoop

- Based on Google File System (GFS) - 2003
- Important idea: Map Reduce
- Started at Apache Nutch Project (a crawler)
- Moved to Hadoop Project - 2006
- Created by Doug Cutting
- Named after his son’s toy elephant

# Brief History: Spark

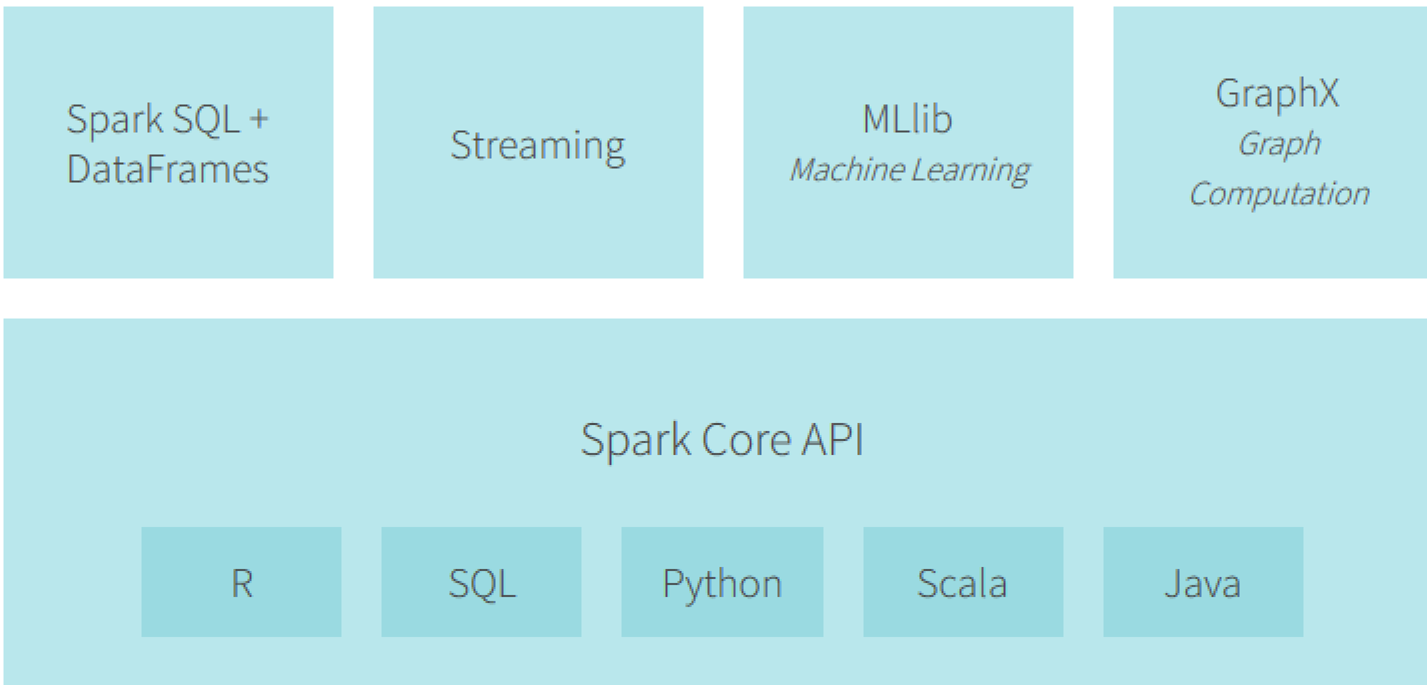
- Origin at UC Berkeley by Matei Zaharia 2009
- It was a class project: to build a cluster management framework supporting different kinds of cluster computing systems
- Goal: interactive and iterative processing
- Input target: HDFS data
- Donated to Apache Software Foundation in 2013

# Hadoop: Ecosystem



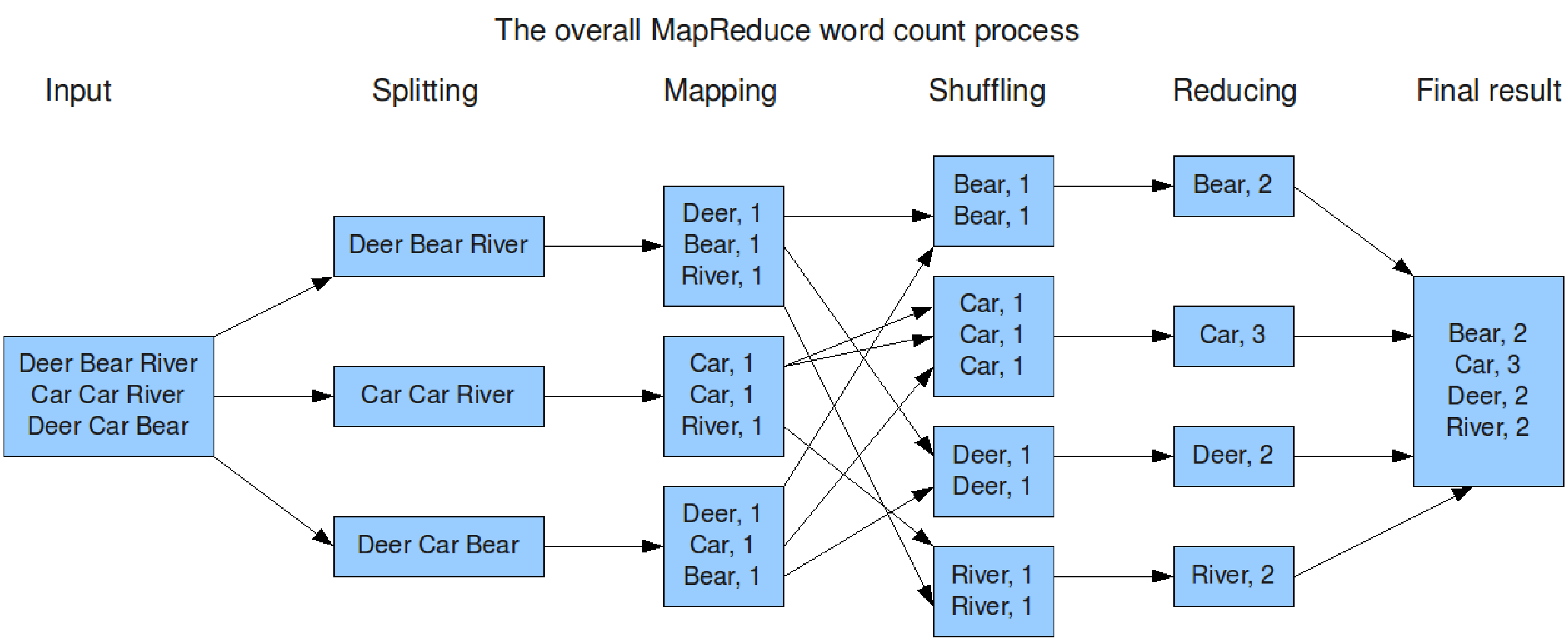

The Big Data Blog 2016

# Spark: Ecosystem



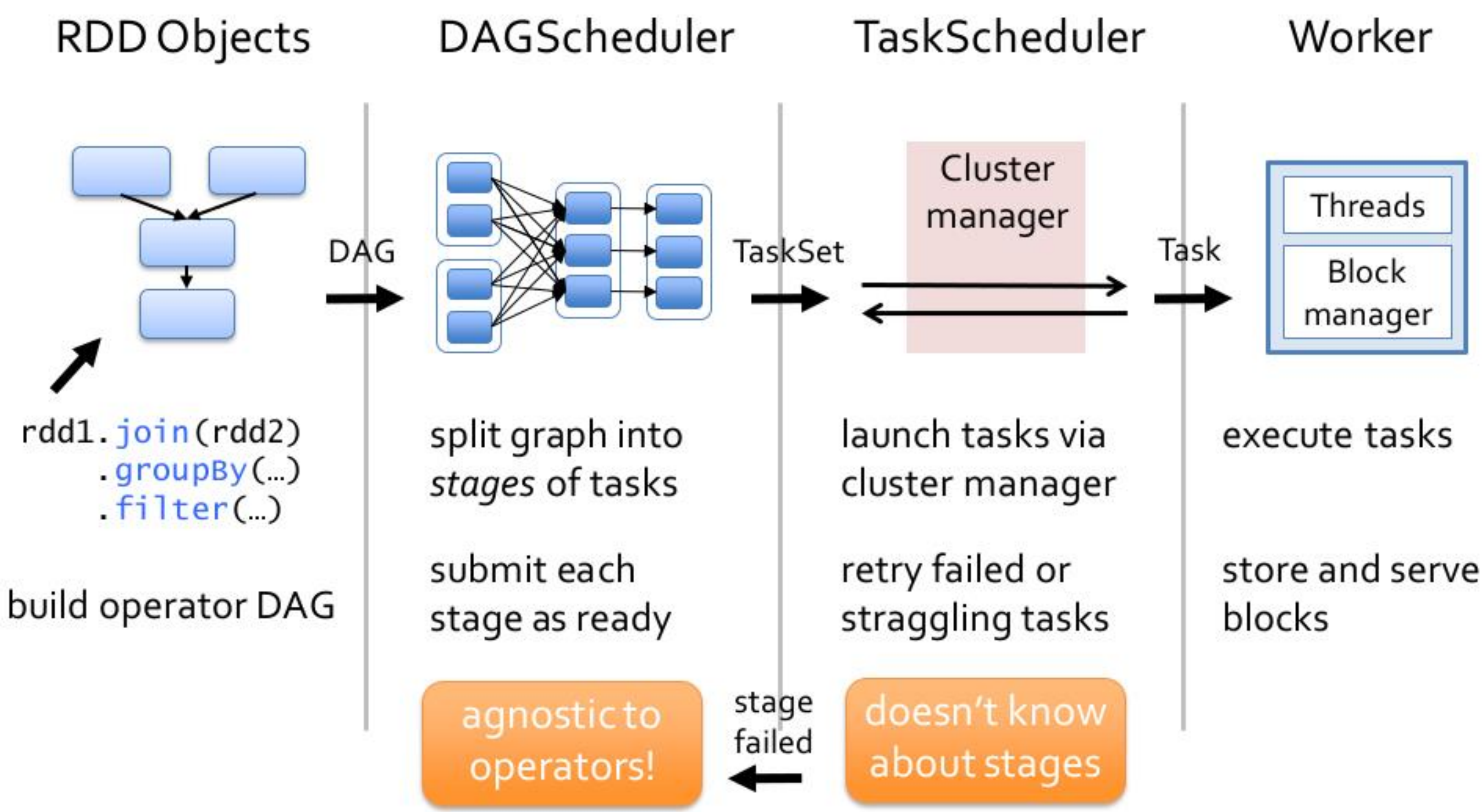

Databricks 2017

# Hadoop: MapReduce




Xiaochong Zhang 2013

# Spark: Layers




Quang-Nhat, Hoang 2015

# Spark: Resilient Distributed Datasets (RDDs)

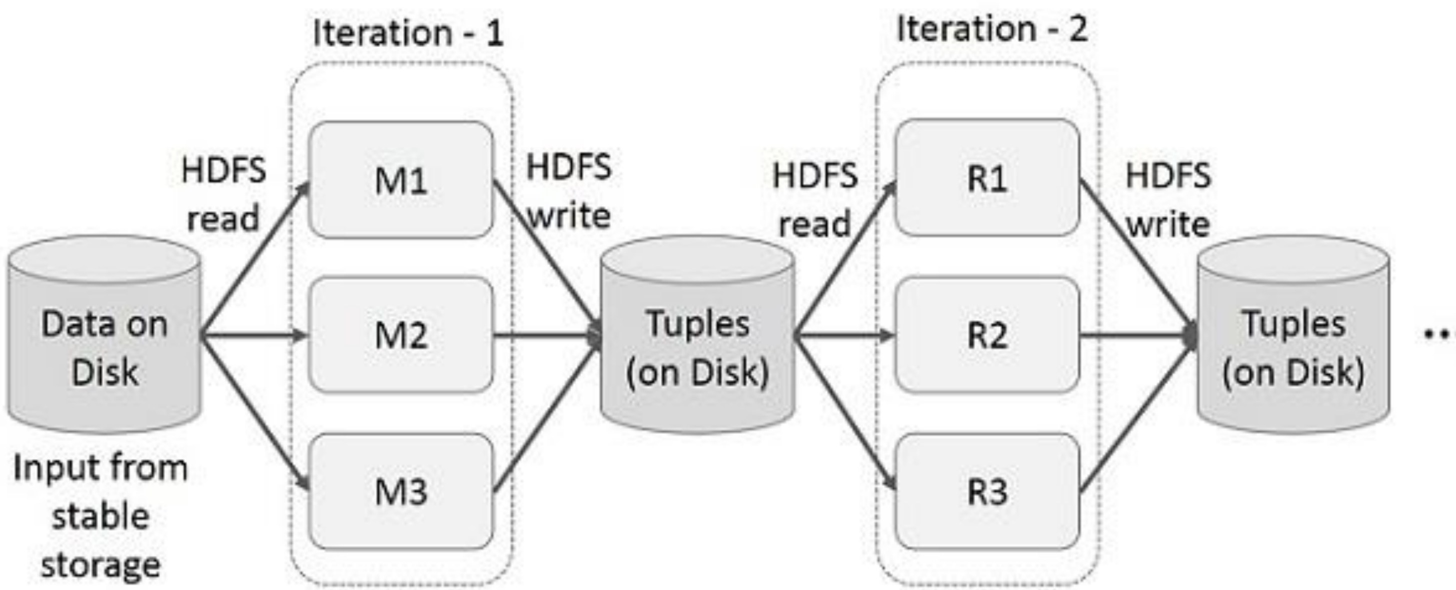
- Fault-tolerant collections of elements that can be operated on in parallel.
- Can reference a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.
- Spark can create RDDs from any storage source supported by Hadoop, including local filesystems or one of those listed previously. (Hess, Ken 2016)

# Spark: RDD

An RDD possesses five main properties:

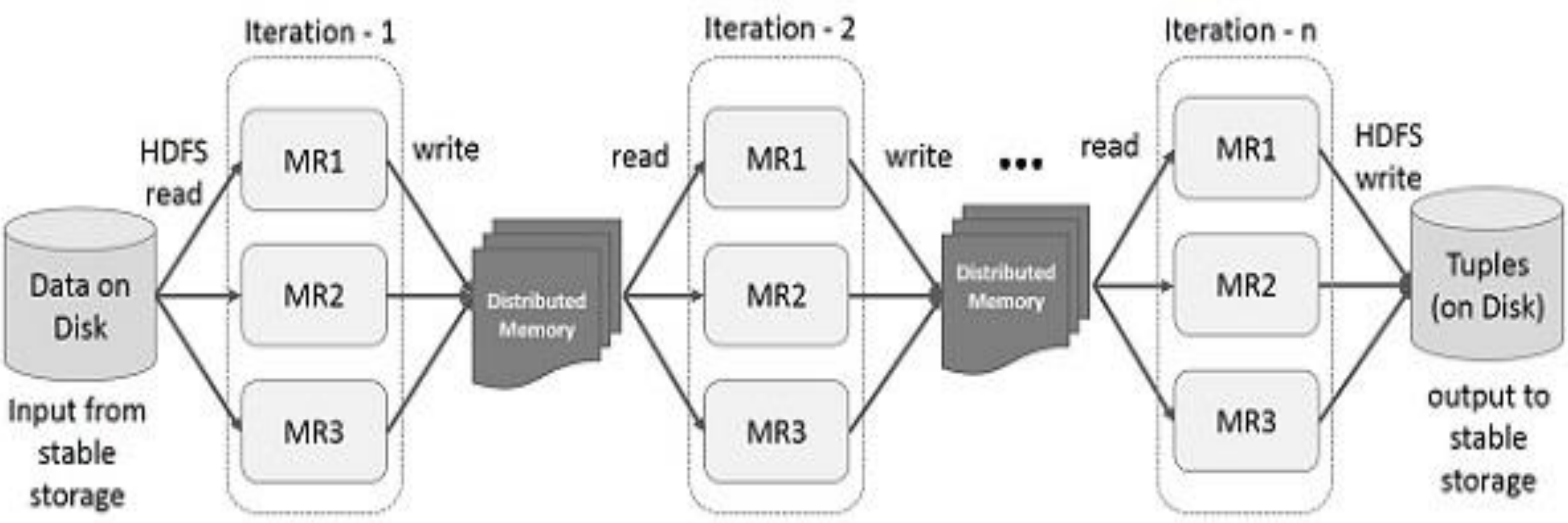
- A list of partitions
- A function for computing each split
- A list of dependencies on other RDDs
- Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
- Optionally, a list of preferred locations to compute each split on (e.g. block locations for an HDFS file)

# Hadoop: Iterative Operations




Tutorialspoint 2017

# Spark: Iterative Operations




Tutorialspoint 2017

# Hadoop & Spark

- Both big data frameworks
- Open source
- From Apache
- Different strengths (Hadoop- batch and reliable, spark-speed)
- They are able to work together

# Hadoop: Advantages/Disadvantages

- Has its own distributed storage system (scalable, commodity hardware)
- Writes all data back to physical storage after each operation to full recover from failure
- More advanced on security and support infrastructure
  - Kerberos authentication, access control lists (ACLs), Service Level Authorization, which ensures that clients have the right permissions.
- Setup to continuously gather information from websites and there were no requirements for this data in or near real-time.

# Spark: Advantages/Disadvantages

- Requires an hdfs (so build on hadoop)
- Speed- operations in memory: copy from distributed physical storage to faster RAM, reducing this way reading and writing from hard drives (hadoop)
- Volatile RAM but Resilient Distributed Datasets to recover from failure

*"Spark has been shown to work well up to petabytes. It has been used to sort 100 TB of data 3X faster than Hadoop MapReduce on one-tenth of the machines."*

*(Xin ,Reynold 2014)*

# Spark: Advantages/Disadvantages

- Can handle advanced data processing tasks like real time, batch processing, stream processing, interactive queries and machine learning
- Ease of use in that it comes with user-friendly APIs for Scala (its native language), Java, Python, and Spark SQL
- Has an interactive mode so that developers and users alike can have immediate feedback for queries and other actions

# Cost

Spark systems cost more because of the large amounts of RAM required to run everything in memory. But what's also true is that Spark's technology reduces the number of required systems. So, you have significantly fewer systems that cost more. There's probably a point at which Spark actually reduces costs per unit of computation even with the additional RAM requirement. (Hess, Ken 2016)

# Hadoop vs. Spark: Components of Interest

| | | Word Count | Sort | K-Means (LR) | Page-Rank |
|---|---|---|---|---|---|
| Shuffle | Aggregation | √ | | √ | √ |
| | External sort | | √ | | |
| | Data transfer | | √ | | √ |
| Execution | Task parallelism | √ | √ | √ | √ |
| | Stage overlap | | √ | | |
| | Data pipelining | | | | √ |
| Caching | Input | | | √ | √ |
| | Intermediate data | | | | √ |

Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., & Özcan, F. (2015).

# Hadoop vs. Spark: WordCount

| Platform | Spark | MR | Spark | MR | Spark | MR |
|---|---|---|---|---|---|---|
| Input size (GB) | 1 | 1 | 40 | 40 | 200 | 200 |
| Number of map tasks | 9 | 9 | 360 | 360 | 1800 | 1800 |
| Number of reduce tasks | 8 | 8 | 120 | 120 | 120 | 120 |
| Job time (Sec) | 30 | 64 | 70 | 180 | 232 | 630 |
| Median time of map tasks (Sec) | 6 | 34 | 9 | 40 | 9 | 40 |
| Median time of reduce tasks (Sec) | 4 | 4 | 8 | 15 | 33 | 50 |
| Map Output on disk (GB) | 0.03 | 0.015 | 1.15 | 0.7 | 5.8 | 3.5 |

Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., & Özcan, F. (2015).

# Hadoop vs. Spark: Sort

| Platform | Spark | MR | Spark | MR | Spark | MR |
|---|---|---|---|---|---|---|
| Input size (GB) | 1 | 1 | 100 | 100 | 500 | 500 |
| Number of map tasks | 9 | 9 | 745 | 745 | 4000 | 4000 |
| Number of reduce tasks | 8 | 8 | 248 | 60 | 2000 | 60 |
| Job time | 32s | 35s | 4.8m | 3.3m | 44m | 24m |
| Sampling stage time | 3s | 1s | 1.1m | 1s | 5.2m | 1s |
| Map stage time | 7s | 11s | 1.0m | 2.5m | 12m | 13.9m |
| Reduce stage time | 11s | 24s | 2.5m | 45s | 26m | 9.2m |
| Map output on disk (GB) | 0.63 | 0.44 | 62.9 | 41.3 | 317.0 | 227.2 |

Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., & Özcan, F. (2015).

# Hadoop vs. Spark:K-Means

| Platform | Spark | MR | Spark | MR | Spark | MR |
|---|---|---|---|---|---|---|
| Input size (million records) | 1 | 1 | 200 | 200 | 1000 | 1000 |
| Iteration time 1st | 13s | 20s | 1.6m | 2.3m | 8.4m | 9.4m |
| Iteration time Subseq. | 3s | 20s | 26s | 2.3m | 2.1m | 10.6m |
| Median map task time 1st | 11s | 19s | 15s | 46s | 15s | 46s |
| Median reduce task time 1st | 1s | 1s | 1s | 1s | 8s | 1s |
| Median map task time Subseq. | 2s | 19s | 4s | 46s | 4s | 50s |
| Median reduce task time Subseq. | 1s | 1s | 1s | 1s | 3s | 1s |
| Cached input data (GB) | 0.2 | - | 41.0 | - | 204.9 | - |

Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., & Özcan, F. (2015).

# Hadoop vs. Spark: PageRank

| Platform | Spark-Naive | Spark-GraphX | MR | Spark-Naive | Spark-GraphX | MR |
|---|---|---|---|---|---|---|
| Input (million edges) | 17.6 | 17.6 | 17.6 | 1470 | 1470 | 1470 |
| Pre-processing | 24s | 28s | 93s | 7.3m | 2.6m | 8.0m |
| 1st Iter. | 4s | 4s | 43s | 3.1m | 37s | 9.3m |
| Subsequent Iter. | 1s | 2s | 43s | 2.0m | 29s | 9.3m |
| Shuffle data | 73.1MB | 69.4MB | 141MB | 8.4GB | 5.5GB | 21.5GB |

Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., & Özcan, F. (2015).

# Hadoop Improvement: MR Tuner

- Automatic toolkit for MapReduce job optimization.
- Novel Producer-Transporter-Consumer (PTC) Model
    - Characterizes the tradeoffs in the parallel execution among tasks.
- Relations among about twenty parameters, which have significant impact on the job performance.
- Efficient search algorithm to find the optimal execution plan.

# MRTuner: Parallelism

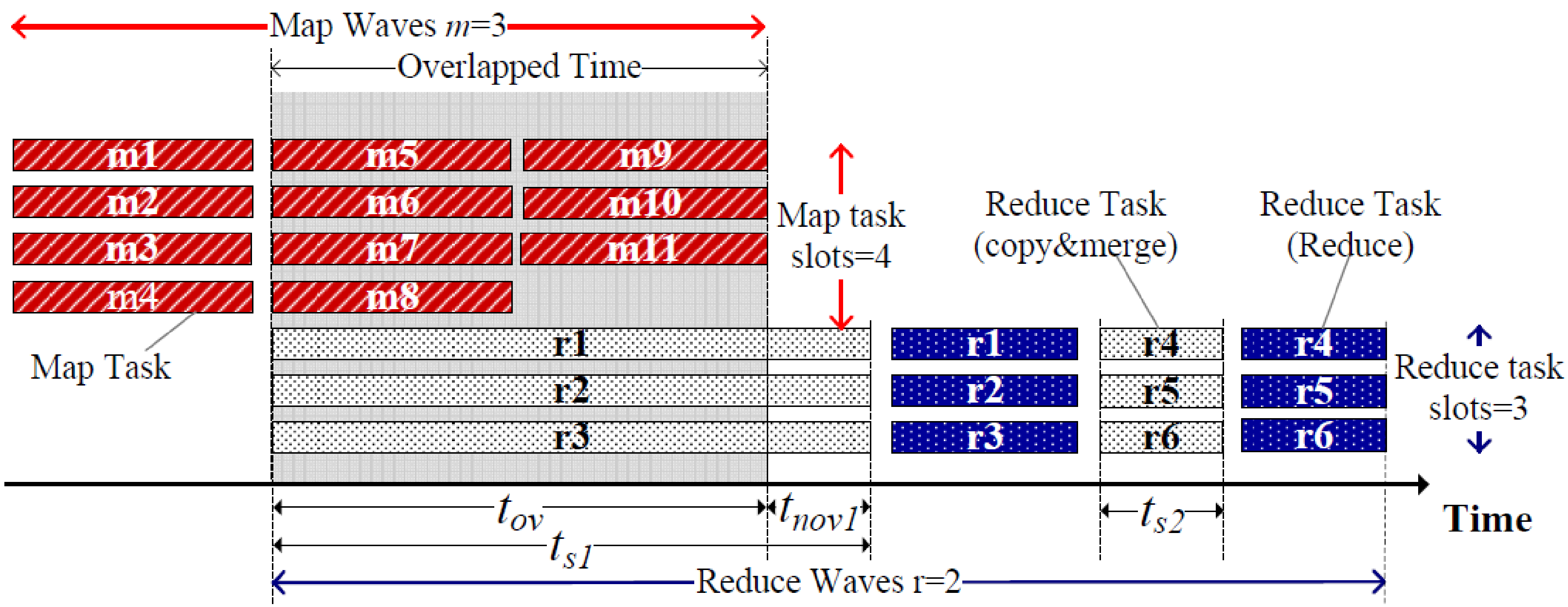


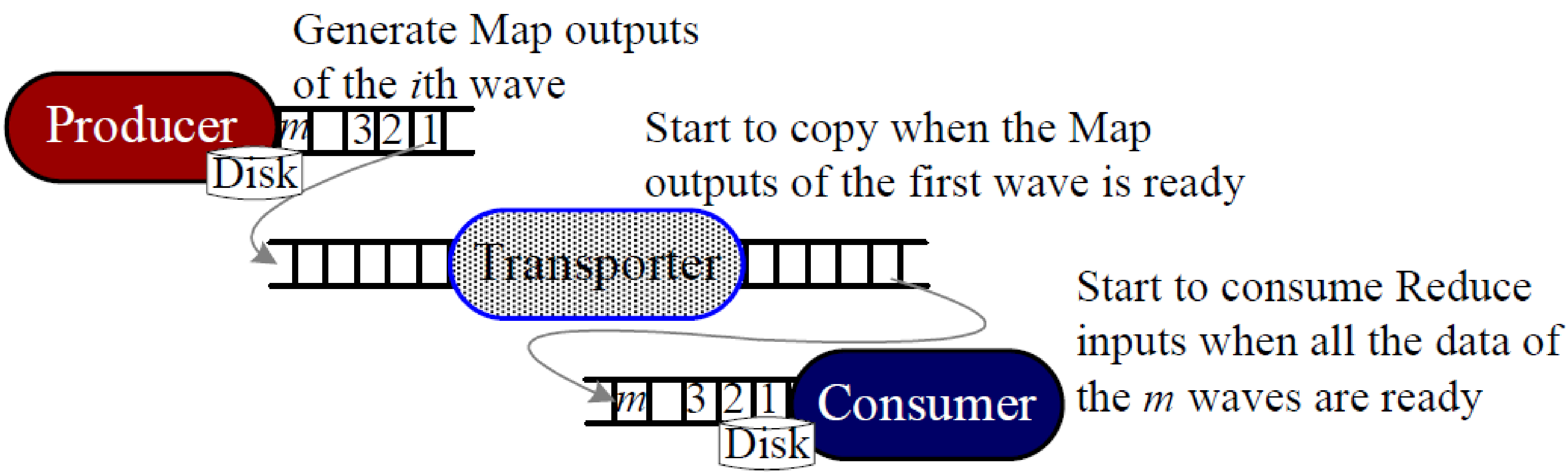

**Figure 1: The Pipelined Execution of a MapReduce Job**

Shi, J., Zou, J., Lu, J., Cao, Z., Li, S., & Wang, C. (2014).

# Tuning Parallelism

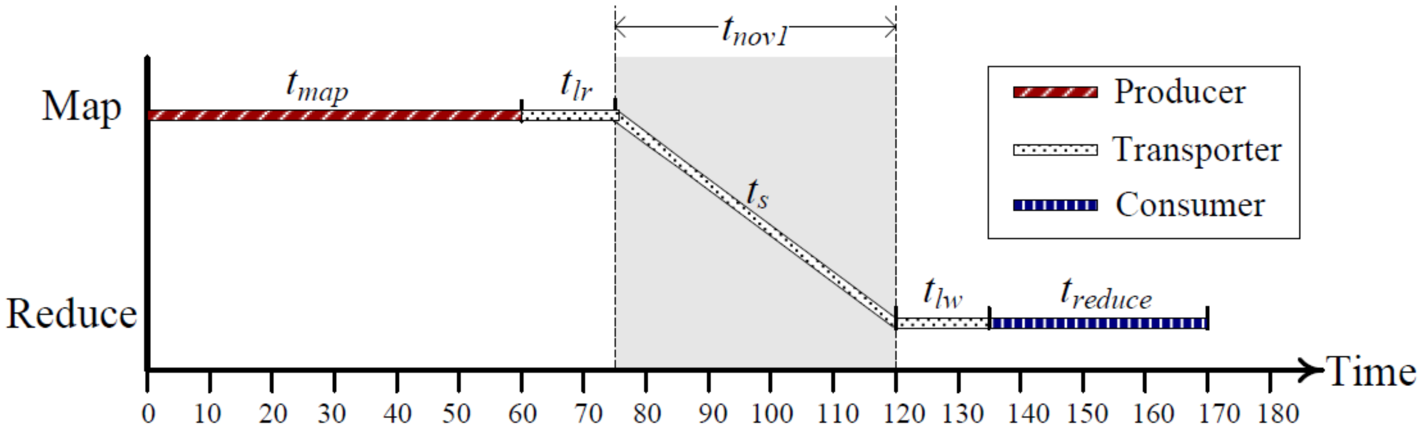
- Number of Map task waves
- Map output compression option
- Copy Speed in the Shuffle phase
- Number of reduce task waves

# MRTuner: PTC Model



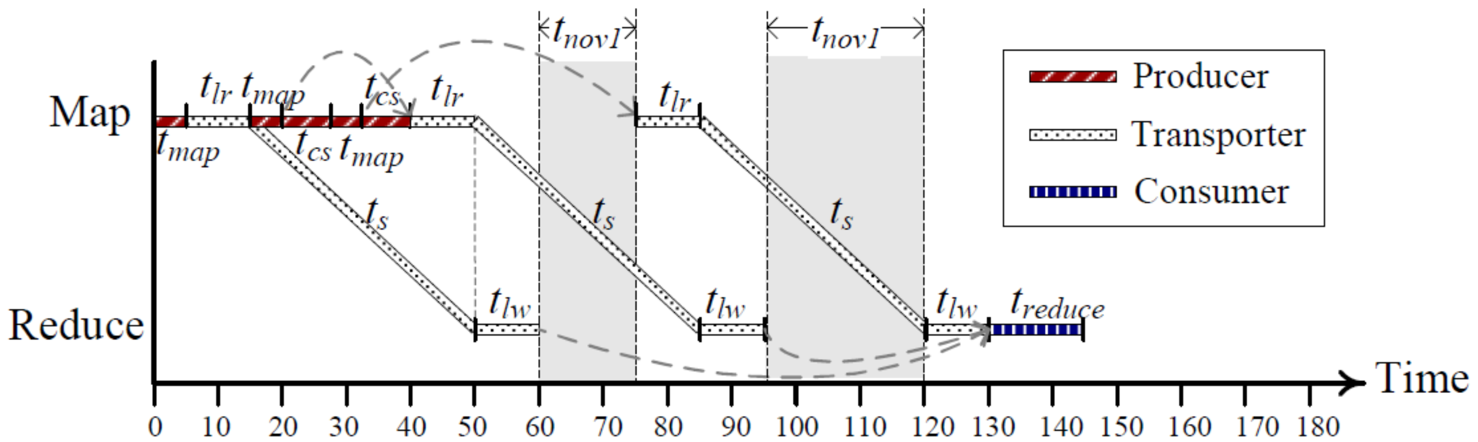

Shi, J., Zou, J., Lu, J., Cao, Z., Li, S., & Wang, C. (2014).

# Tuning with PTC Model




Top: T=170

Bottom: T= 145




Shi, J., Zou, J., Lu, J., Cao, Z., Li, S., & Wang, C. (2014).

# MR Tuner Results

**Table 5: The Comparison between Hadoop-X and MRTuner**

| JobName | ID | Clu-ster | Input (GB) | Hadoop -X(sec) | MRTuner (sec) | Speed -up |
|---|---|---|---|---|---|---|
| Terasort | TS-1 | $\mathcal{A}$ | 10 | 469 | 278 | 1.7 |
| Terasort | TS-2 | $\mathcal{A}$ | 50 | 2109 | 1122 | 1.87 |
| Terasort | TS-3 | $\mathcal{B}$ | 200 | 767 | 295 | 2.60 |
| Terasort | TS-4 | $\mathcal{B}$ | 1000 | 6274 | 2192 | 2.86 |
| N-Gram | NG-1 | $\mathcal{A}$ | 0.18 | 4364 | 192 | 22.7 |
| N-Gram | NG-2 | $\mathcal{A}$ | 0.7 | N/A | 661 | $\infty$ |
| N-Gram | NG-3 | $\mathcal{A}$ | 1.4 | N/A | 1064 | $\infty$ |
| N-Gram | NG-4 | $\mathcal{B}$ | 1.4 | 1100 | 249 | 4.41 |
| N-Gram | NG-5 | $\mathcal{B}$ | 2.8 | 1292 | 452 | 2.86 |
| N-Gram | NG-6 | $\mathcal{B}$ | 5.6 | 1630 | 930 | 1.75 |
| PR(Trans.) | PR-1 | $\mathcal{A}$ | 3.23 | 962 | 446 | 2.2 |
| PR(Deg.) | PR-2 | $\mathcal{A}$ | Inter | 49 | 41 | 1.2 |
| PR(Iter.) | PR-3 | $\mathcal{A}$ | Inter | 933 | 639 | 1.5 |
| PR(Trans.) | PR-4 | $\mathcal{B}$ | 3.23 | 148 | 65 | 2.28 |
| PR(Deg.) | PR-5 | $\mathcal{B}$ | Inter | 24 | 22 | 1.09 |
| PR(Iter.) | PR-6 | $\mathcal{B}$ | Inter | 190 | 82 | 2.32 |

Shi, J., Zou, J., Lu, J., Cao, Z., Li, S., & Wang, C. (2014).

# MR Tuner Results

- The search latency of *MRTuner* is a few orders of magnitude faster than that of the state-of-the-art cost-based optimizer
- The effectiveness of the optimized execution plan is also significantly improved.
- MR Tuner can find much better execution plans compared with existing MR optimizers

# Wrap up

- What are Hadoop and spark
- Features of each
- Pros/Cons
- Benchmark
- MRTuner

- So...which one is better?

# Questions?

Diolch
Kiitos
Sheun
umesc
Dekuju/Dekujeme
Kasih
Todah
Mamnoon
Shnorhakalutiun
Hvala
Shokriya
Dziękuję
Shokrun
Spaas
Mul
Ači
Gamsahapnida
Ngiyabonga
Xie
Dank
Takk
Cam
Gra
or
Dhanyavadagalu
Waad
Kop
Salamat
Merci
al
Dhanyavaad
Dakujem
Dankie
Go
Grazie
Faleminderit
Dhanyavad
Khopjai
Kruthagnathalu
Daw
ThankYou
kun
Arigatou
krap
Shukriya
Or
Dhonnobaad
Tack
ederim
Asante
Gracias
Nandree
Blagodariya
Gomapsupnida
Euxaristo
Kun
Hain
Dhan
Grazzi
raibh
Terima
Fyrir
Enkosi
Danke
dank
daa

# References


- Shi, J., Zou, J., Lu, J., Cao, Z., Li, S., & Wang, C. (2014). MRTuner: a toolkit to enable holistic optimization for mapreduce jobs. *Proceedings of the VLDB Endowment*, *7*(13), 1319-1330.
- Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., & Özcan, F. (2015). Clash of the titans: Mapreduce vs. spark for large scale data analytics. *Proceedings of the VLDB Endowment*, *8*(13), 2110-2121.
- Wikimedia Foundation (2017), *Apache Hadoop*, retrieved March 9, 2017 from https://en.wikipedia.org/wiki/Apache_Hadoop

# References


- Wikimedia Foundation (2017), *Apache Spark*, retrieved March 9, 2017 from https://en.wikipedia.org/wiki/Apache_Spark
- Hoang, Nhat (2015), *Spark job submission breakdown,* retrieved March 9, 2017 from https://hxquangnhat.com/2015/04/03/arch-spark-job-submission-breakdown/
- Tutorialspoint (2017), *Resilient Distributed Datasets,* retrieved March 9, 2017 from https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm
- Databricks (2017), *Apache Spark,* retrieved March 9, 2017 from https://databricks.com/spark/about

# References


- The Big Data Blog (2016), *Hadoop Ecosystem Overview*, retrieved March 9, 2017 from http://thebigdatablog.weebly.com/blog/category/big-data
- Phatak, Madhukara (2015), *History of Apache Spark : Journey from Academia to Industry,* retrieved March 9, 2017 from http://blog.madhukaraphatak.com/history-of-spark/
- Zhang, Xiaochong (2013), *A Simple Example to Demonstrate how does the MapReduce work,* retrieved March 9, 2017 from http://xiaochongzhang.me/blog/?p=338

# References


- Marr, Bernard (2015), *Spark Or Hadoop -- Which Is The Best Big Data Framework?,* retrieved March 9, 2017 from https://www.forbes.com/sites/bernardmarr/2015/06/22/spark-or-hadoop-which-is-the-best-big-data-framework/#36cfe2ca127e
- Hess, Ken (2016), *Hadoop vs. Spark: The New Age of Big Data,* retrieved March 9, 2017 from http://www.datamation.com/data-center/hadoop-vs.-spark-the-new-age-of-big-data.html
- Xin, Reynold (2014) *Apache Spark officially sets a new record in large-scale sorting,* retrieved March 9, 2017 from https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html