Hadoop MapReduce over Lustre*

High Performance Data Division

Omkar Kulkarni

April 16, 2013

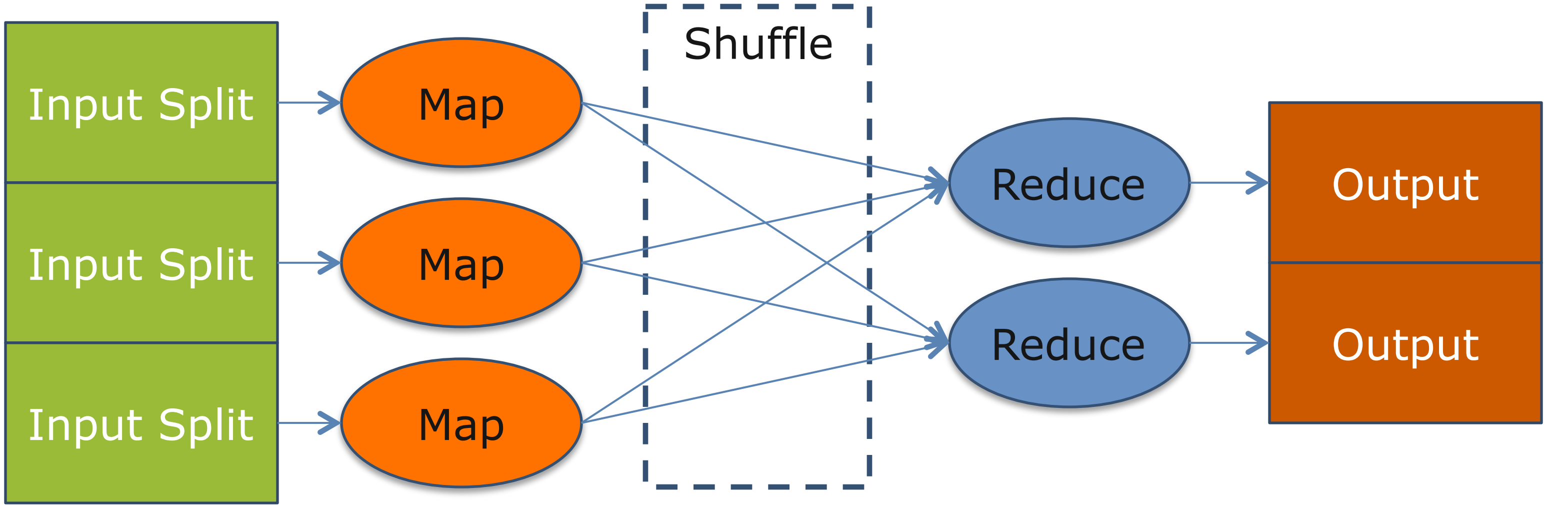
Agenda

- **Hadoop Intro**
- **Why run Hadoop on Lustre?**
- **Optimizing Hadoop for Lustre**
- **Performance**
- **What's next?**

A Little Intro of Hadoop

- Open source MapReduce framework for data-intensive computing
- Simple programming model – two functions: Map and Reduce
- Map: Transforms input into a list of key value pairs
  - Map(D) → List[Ki , Vi]
- Reduce: Given a key and all associated values, produces result in the form of a list of values
  - Reduce(Ki , List[Vi]) → List[Vo]
- Parallelism hidden by framework
  - Highly scalable: can be applied to large datasets (Big Data) and run on commodity clusters
- Comes with its own user-space distributed file system (HDFS) based on the local storage of cluster nodes

A Little Intro of Hadoop (cont.)



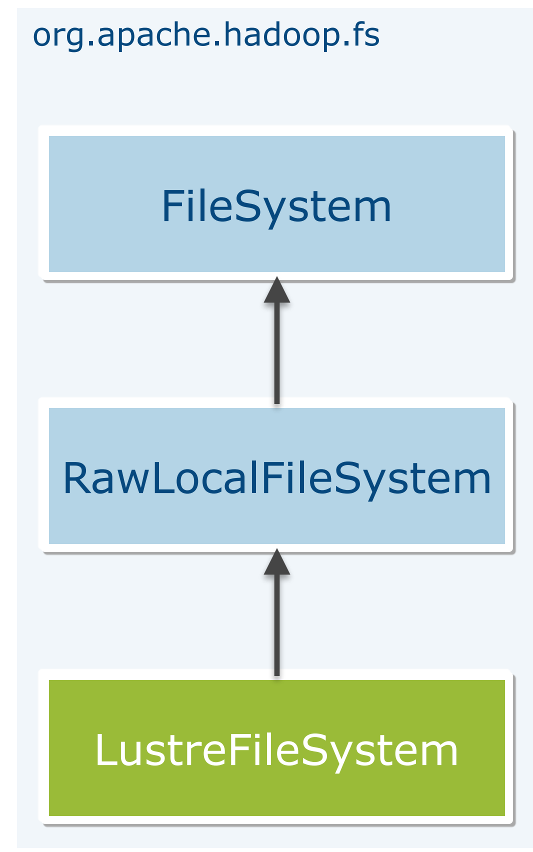
- Framework handles most of the execution
- Splits input logically and feeds mappers
- Partitions and sorts map outputs (Collect)
- Transports map outputs to reducers (Shuffle)
- Merges output obtained from each mapper (Merge)

Why Hadoop with Lustre?

- HPC moving towards Exascale. Simulations will only get bigger
- Need tools to run analyses on resulting massive datasets
- Natural allies:
  - Hadoop is the most popular software stack for big data analytics
  - Lustre is the file system of choice for most HPC clusters
- Easier to manage a single storage platform
  - No data transfer overhead for staging inputs and extracting results
  - No need to partition storage into HPC (Lustre) and Analytics (HDFS)
- Also, HDFS expects nodes with locally attached disks, while most HPC clusters have diskless compute nodes with a separate storage cluster

How to make them cooperate?

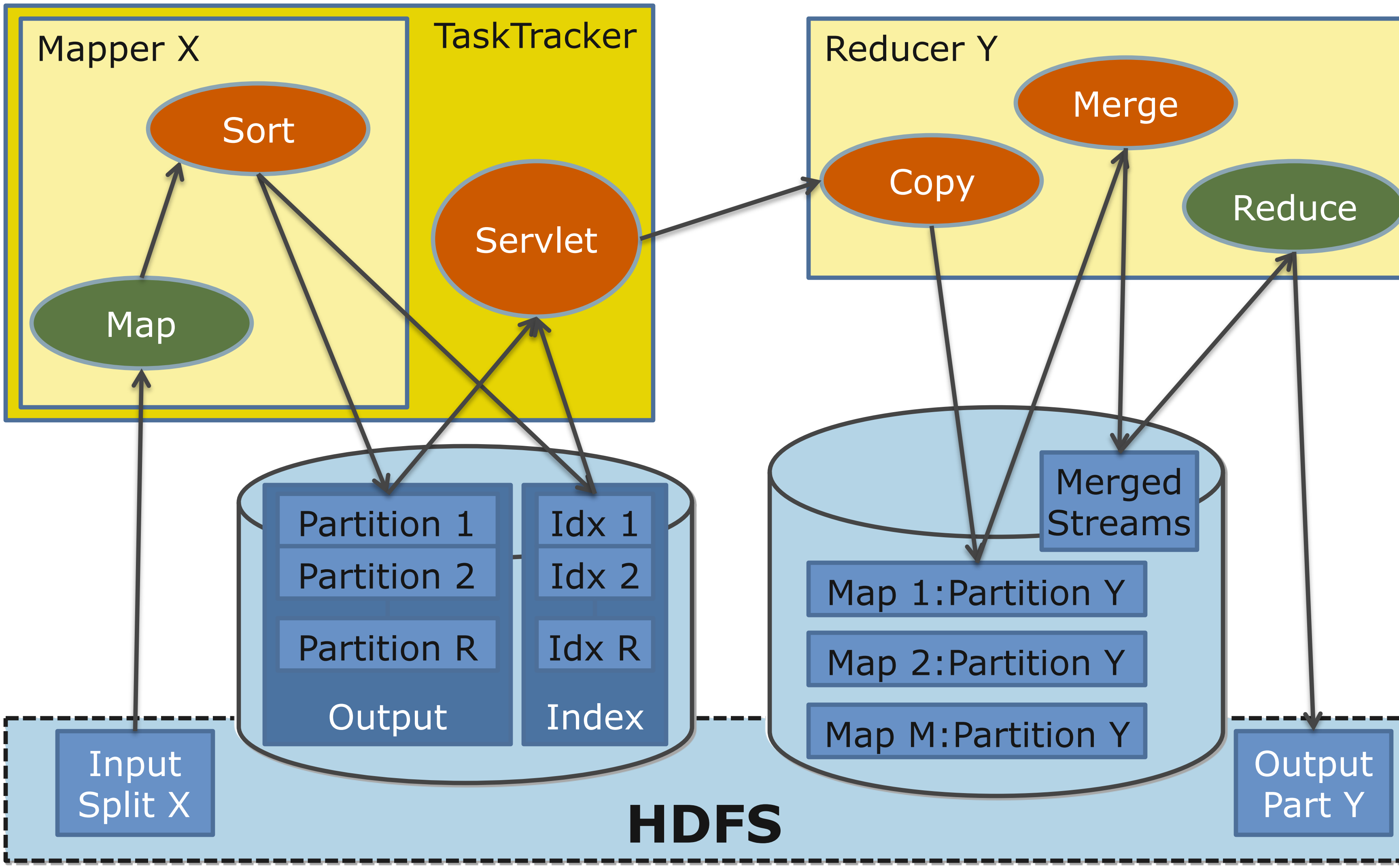
- Hadoop uses pluggable extensions to work with different file system types
- Lustre is POSIX compliant:
  - Use Hadoop's built-in LocalFileSystem class
  - Uses native file system support in Java
- Extend and override default behavior: LustreFileSystem
  - Defines new URL scheme for Lustre – lustre:///
  - Controls Lustre striping info
  - Resolves absolute paths to user-defined directory
  - Leaves room for future enhancements
- Allow Hadoop to find it in config files



Sort, Shuffle & Merge

- M → Number of Maps, R → Number of Reduces
- Map output records (Key-Value pairs) organized into R partitions
- Partitions exist in memory. Records within a partition are sorted
- A background thread monitors the buffer, spills to disk if full
- Each spill generates a spill file and a corresponding index file
- Eventually, all spill files are merged (partition-wise) into a single file
- Final index is file created containing R index records
- Index Record = [Offset, Compressed Length, Original Length]
- A Servlet extracts partitions and streams to reducers over HTTP
- Reducer merges all M streams on disk or in memory before reducing

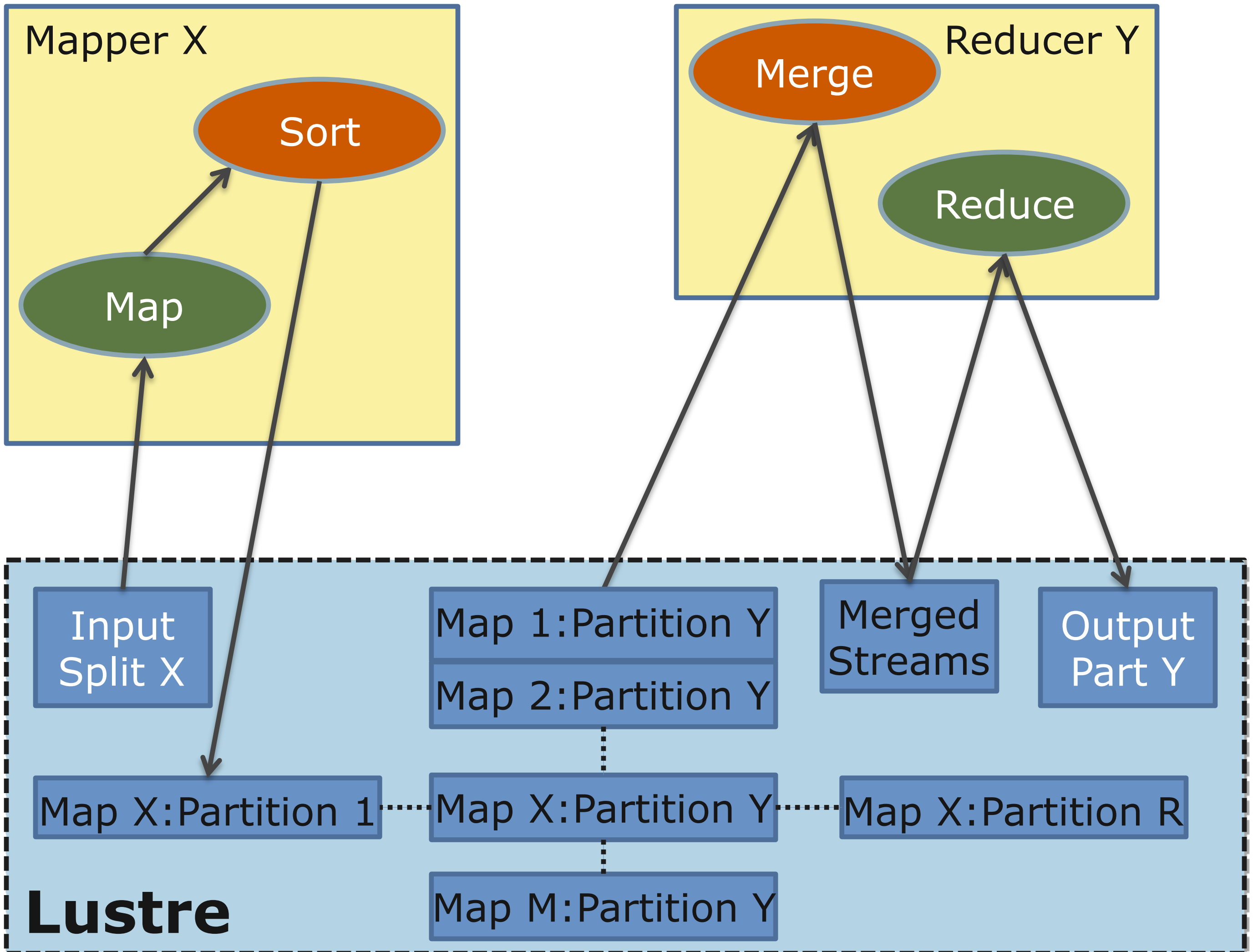
Sort, Shuffle & Merge (Cont.)



Optimized Shuffle for Lustre

- Why? Biggest (but inevitable) bottleneck – bad performance on Lustre!
- How? Shared File System $\longrightarrow$ HTTP transport is redundant
- How would reducers access map outputs?
  - First Method: Let reducers read partitions from map outputs directly
    - But, index information still needed
  - Either, let reducers read index files, as well
    - Results in (M*R) small (24 bytes/record) IO operations
  - Or, let Servlet convey index information to reducer
    - Advantage: Read entire index file at once, and cache it
    - Disadvantage: Seeking partition offsets + HTTP latency
  - Second Method: Let mappers put each partition in a separate file
    - Three birds with one stone: No index files, no disk seeks, no HTTP

Optimized Shuffle for Lustre (Cont.)



Performance Tests

- Standard Hadoop benchmarks were run on the Rosso cluster
- Configuration – Hadoop (Intel Distro v1.0.3):
  - 8 nodes, 2 SATA disks per node (used only for HDFS)
  - One with dual configuration, i.e. master and slave
- Configuration – Lustre (v2.3.0):
  - 4 OSS nodes, 4 SATA disks per node (OSTs)
  - 1 MDS, 4GB SSD MDT
  - All storage handled by Lustre, local disks not used

TestDFSIO Benchmark

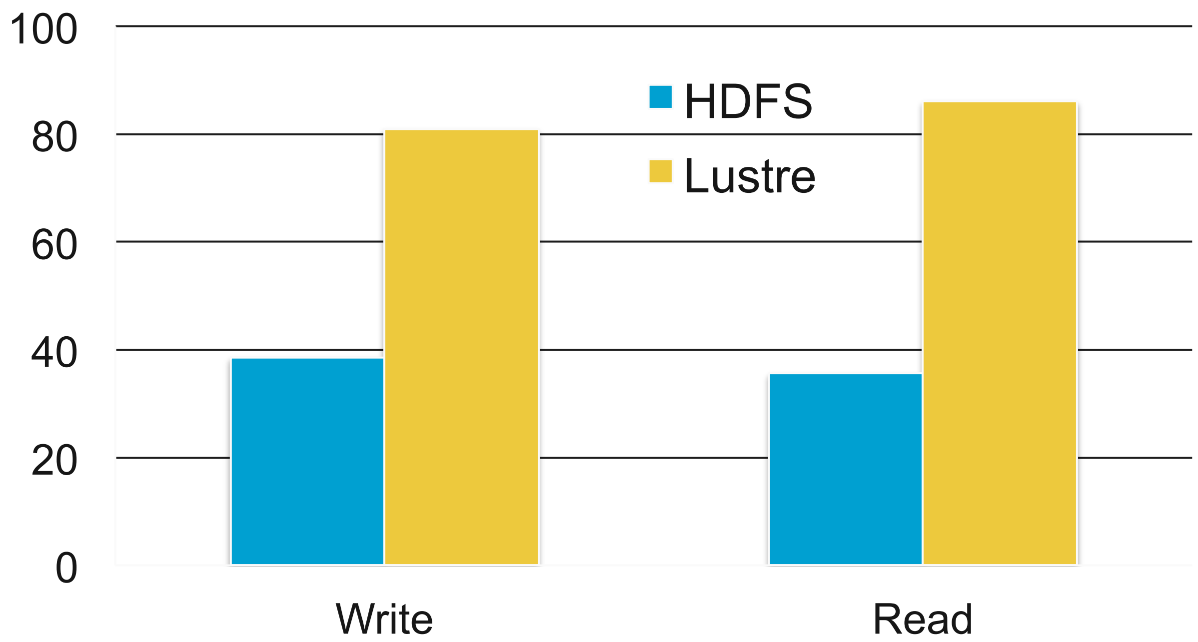
- Tests the raw performance of a file system
- Write and read very large files (35G each) in parallel
- One mapper per file. Single reducer to collect stats
- Embarrassingly parallel, does not test shuffle & sort

Throughput

$$\left(\frac{\sum filesize}{\sum time}\right)$$

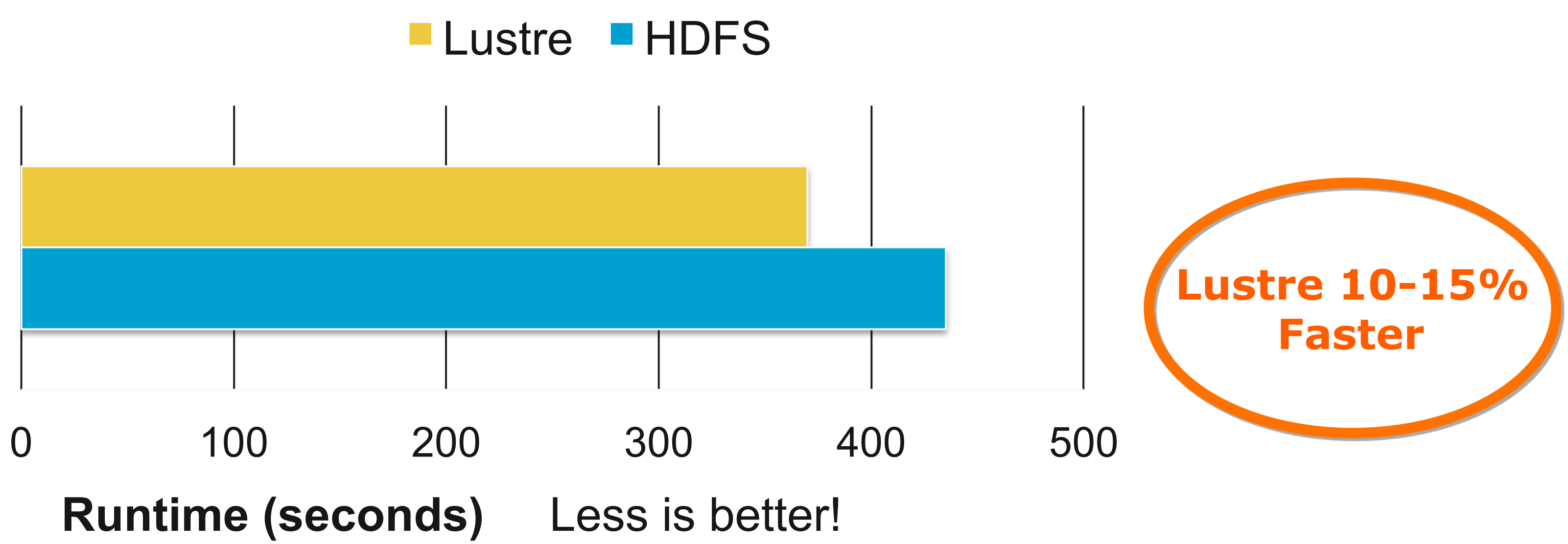
MB/s

More is better!



Terasort Benchmark

- Distributed sort: The primary Map-Reduce primitive
- Sort a 1 Billion records, i.e. approximately 100G
  - Record: Randomly generated 10 byte key + 90 bytes garbage data
- Terasort only supplies a custom partitioner for keys, the rest is just default map-reduce behavior.
- Block Size: 128M, Maps: 752 @ 4/node, Reduces: 16 @ 2/node



Work in progress

- Planned so far
  - More exhaustive testing needed
  - Test at scale: Verify that large scale jobs don't throttle MDS
  - Port to IDH 3.x (Hadoop 2.x): New architecture, More decoupled
  - Scenarios with other tools in the Hadoop Stack: Hive, HBase, etc.

- Further Work
  - Experiment with caching
  - Scheduling Enhancements
  - Exploiting Locality

intel®