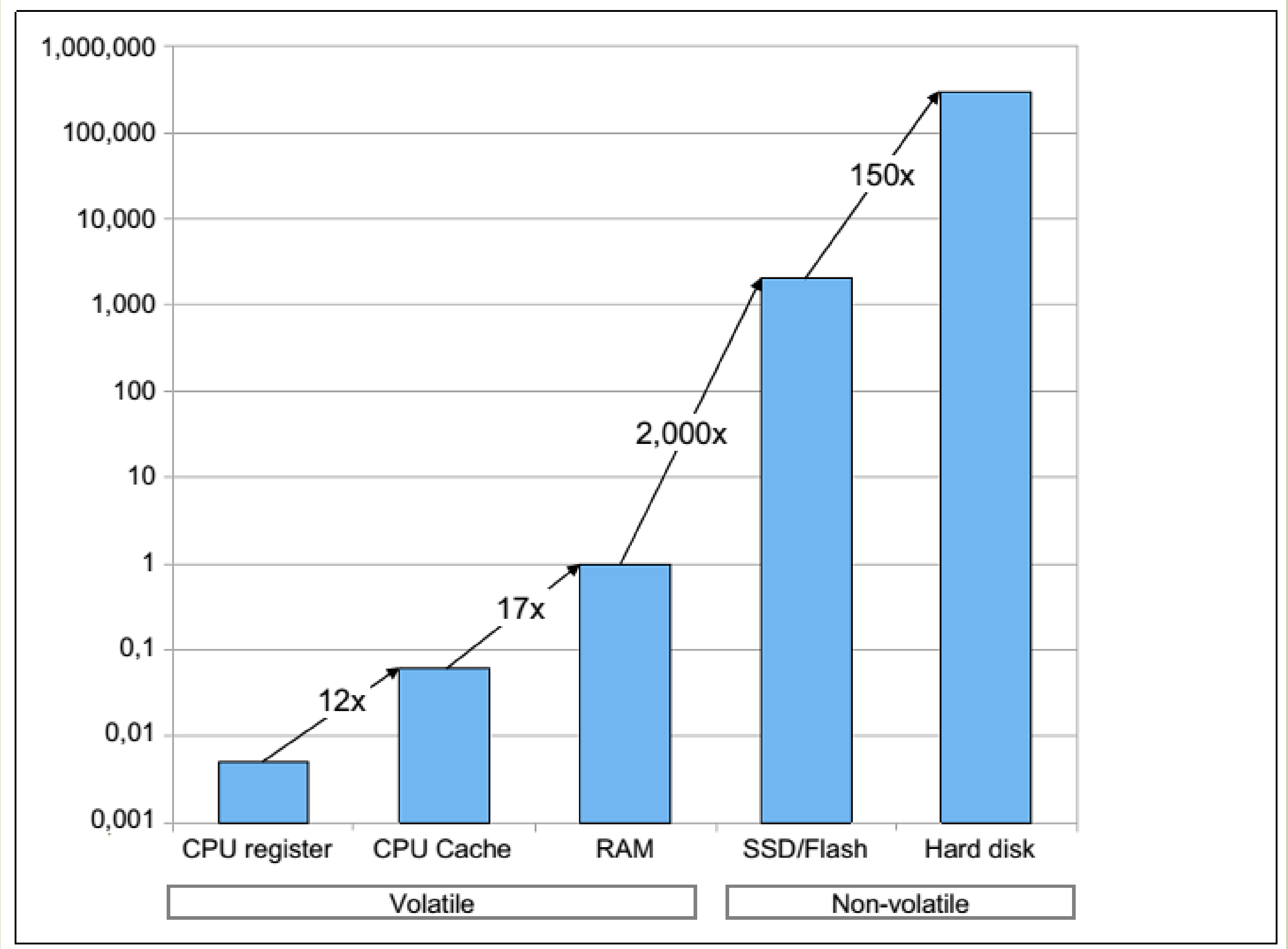
Memory-centric Distributed Computing / In-memory computing

Linh B. Ngo

MOTIVATION

Why do we want in-memory computing?

MOTIVATION


1,000,000
100,000
10,000
1,000
100
10
1
0,1
0,01
0,001
150x
2,000x
17x
12x
CPU register
CPU Cache
RAM
SSD/Flash
Hard disk
Volatile
Non-volatile

WHAT CAN MAPREDUCE OFFER?

- Pros:
  - Inherently parallel
  - Data locality
  - Resiliency and Error Recovery

- Cons:
  - High overhead costs for setting up (JVMs)
  - Difficulty in setting up iterative jobs due to inability to transition between MR stages (having to write back to HDFS first)

DO WE NOT LIKE MAPREDUCE ANYMORE?

- “If all you have is a hammer, everything becomes a nail”
- MapReduce is good for massive scale batch-oriented, single iteration jobs (initial filtering, data cleaning, …)
- Additional tools must be developed

WHAT HAVE BEEN DEVELOPED

- Apache Tez
  - Enable automated “chaining” of MR jobs via dataflow graphs
- Apache Spark
  - Fast processing via read-only in-memory data cache
- Tachyon
  - RAMdisk-based storage on top of persistent storage (HDFS) to support fast in-memory data access

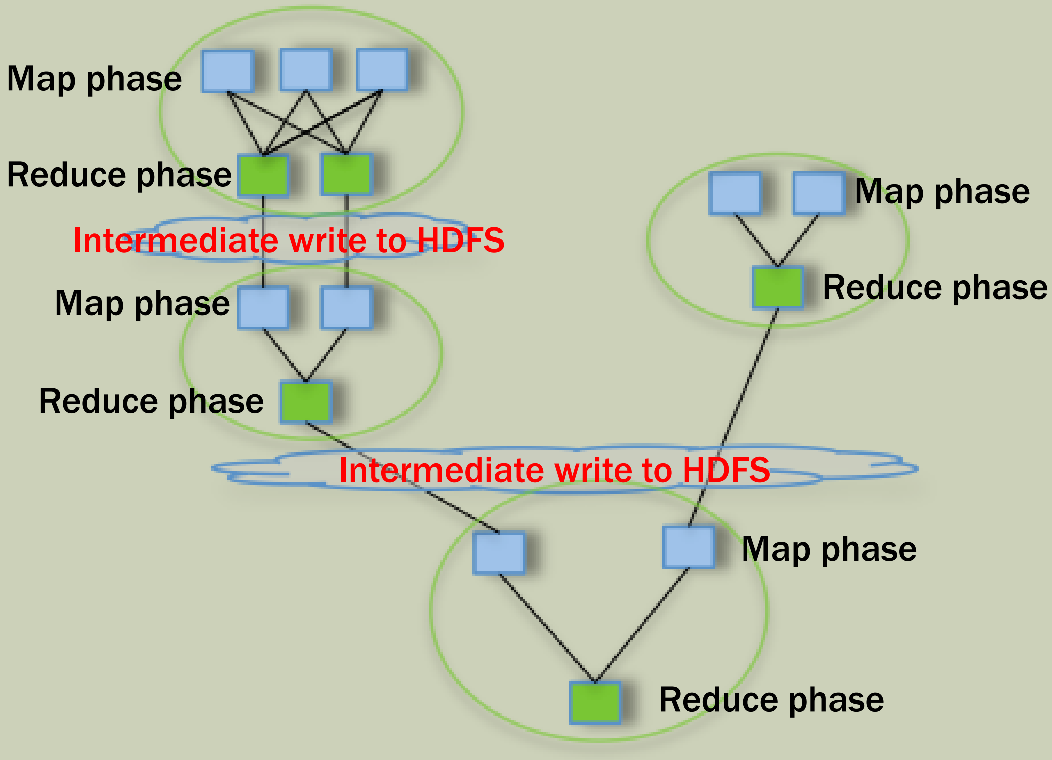
APACHE TEZ

- Extensible framework for building high performance batch and interactive data processing applications
- User-defined flows of data and tasks to support complex query logic that typically require multiple MR jobs
- Current status: Apache top-project, 0.8.4

APACHE TEZ

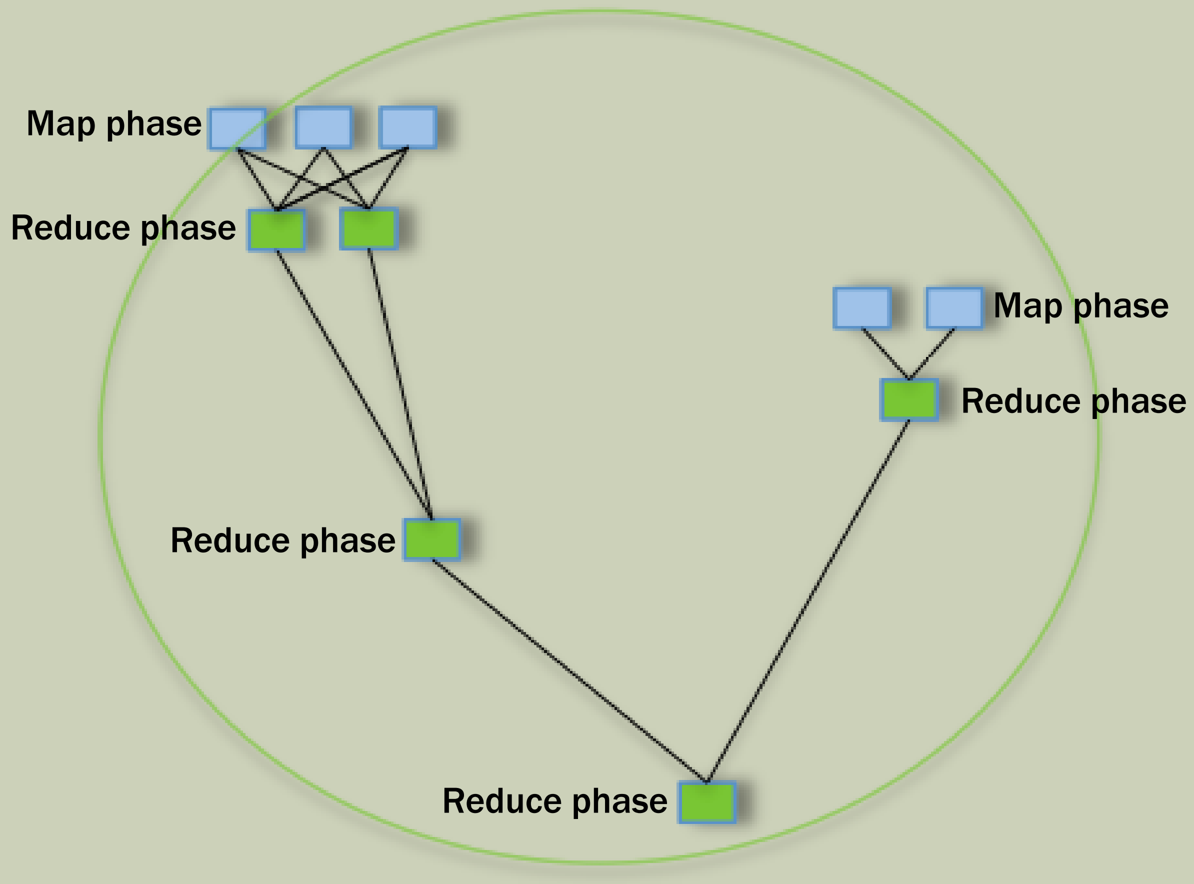
- Data processing is represented as a dataflow graph
  - Vertices represent the processing of data
  - Edges represent movement of data between the processing
- For MR jobs, Tez provides the ability to run a chain of Reduce stages as compared to a single Reduce stage

APACHE TEZ



Pig/Hive - MR

APACHE TEZ



Pig/Hive - Tez

APACHE TEZ: BUILDING BLOCKS

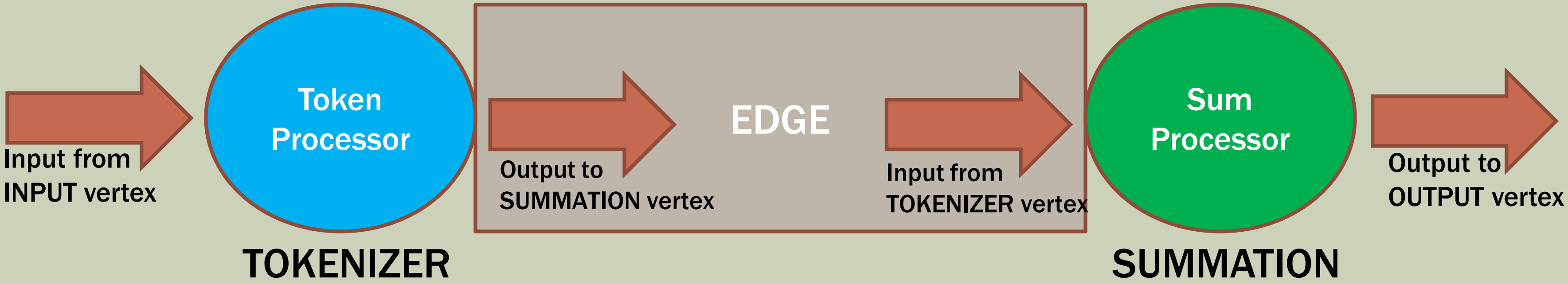
- Vertex: Combination of a set of Inputs, a Processor, and a set of Outputs
  - Input: A pipe through which a processor can accept input data from a data source
  - Processor: The entity responsible for consuming inputs and producing outputs
  - Output: A pipe through which a processor can generate output data

APACHE TEZ: BUILDING BLOCKS

- Edge: A logical entity that represents a number of physical connections between the tasks of two connected vertices
  - Logical: representing the logical edge between 2 vertices
  - Physical: representing the connection between a task of the Source vertex and a task of a Destination vertex
  - A Logical Edge can contain multiple Physical Connections

WORDCOUNT DAG

INPUT: default input from files
OUTPUT: default output to be written to file



APACHE SPARK

- “Lightning-fast cluster computing”
- Integrated into Hortonwork’s and Cloudera’s Hadoop 2.0 production release
- 1.6.0

TARGET SYSTEMS/APPLICATIONS

- Systems:
  - COTS Clusters for large scale data analytics
  - Highly scalable
- Applications:
  - Reuse intermediate results across multiple computations
  - Common in many iterative machine learning and graph algorithms

RESILIENT DISTRIBUTED DATASETS (RDD)

- An abstraction to enable efficient data reuse in a broad range of application
- RDD is implemented in Spark

CHARACTERISTICS OF RDD

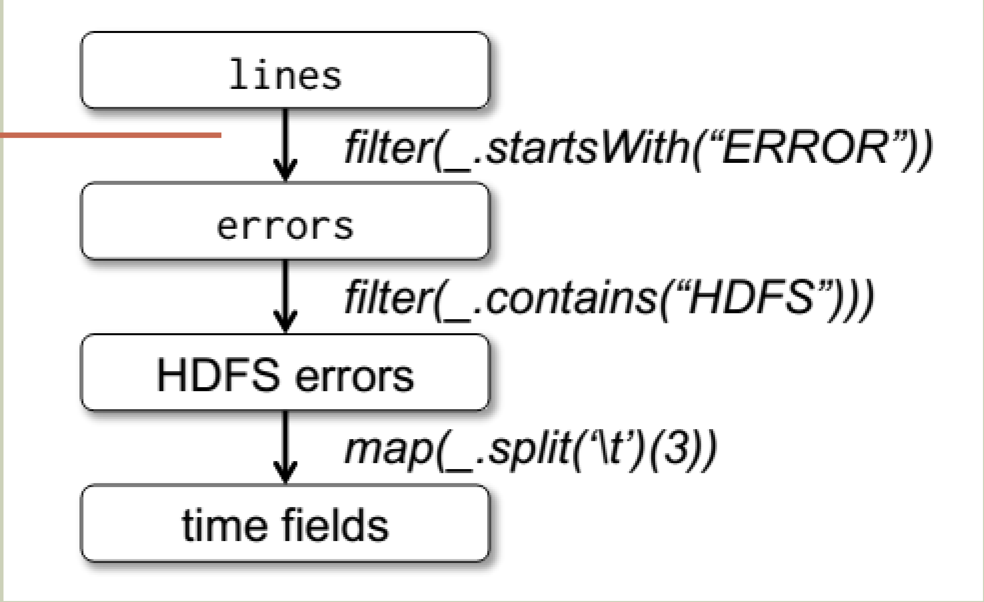
- Read-only, partitioned collection of records
- Created (aka *written*) through deterministic operations on data
  - In stable storage
  - From other RDDs
  - Coarse-grained operations: map, join, filter …
- Do not need to be materialized at all time
  - RDDs are recoverable via **data lineage**

CHARACTERISTICS OF RDD

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
errors.persist()
```

Stable data in HDFS

Persistence in-memory RDD



APPLICATIONS NOT SUITABLE FOR SPARK

- Applications that make asynchronous fine-grained updates to shared state:
  - Storage system for web applications (Traditional applications for standard databases)
  - Incremental web crawler
  - …

APPLICATIONS SUITABLE FOR SPARK

- Batch Analytics/Pleasantly Parallel Applications
- Examples:
  - Logistic Regression
  - PageRank
  - Machine Learning

TACHYON

- Reliable data sharing at memory speed across cluster computing framework
- Integrated into Bekerley Data Analytics Stack (BDAS)
- 0.8

TACHYON: MOTIVATION

- Caching data in memory during computation
- Need in-memory data storage
- Similar to Spark’s concepts:
  - Tachyon stays in memory
  - No data replication, only data lineage
  - Raw data are loaded into Tachyon from persistent storage on disk (e.g.: HDFS)

TACHYON: DESIGN ASSUMPTIONS

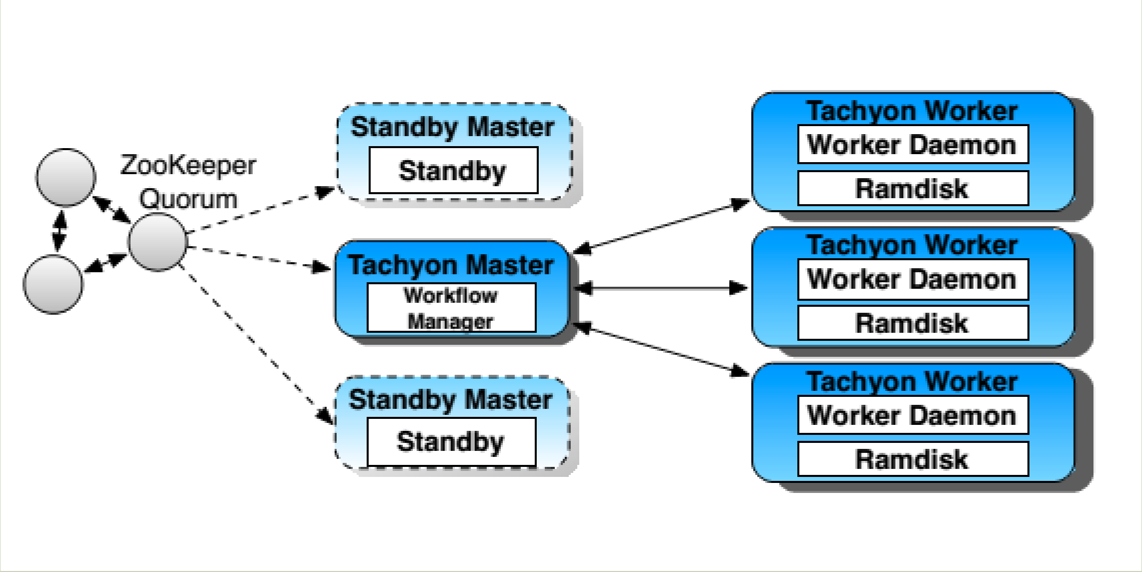
- Immutable data (write once, read many)
- Deterministic jobs (multiple runs produce same results)
- Locality-based scheduling
- Working set data fits in memory (Full data requires disk storage)
- Program size << data size (move computation to data)

TACHYON: KEY ARCHITECTURAL DETAILS

- Two layers
- Master-slave Architecture
- Lineage Overhead Garbage Collection
- Data Eviction
- Master Fault-Tolerance
- Check-pointing
- Resource Allocation

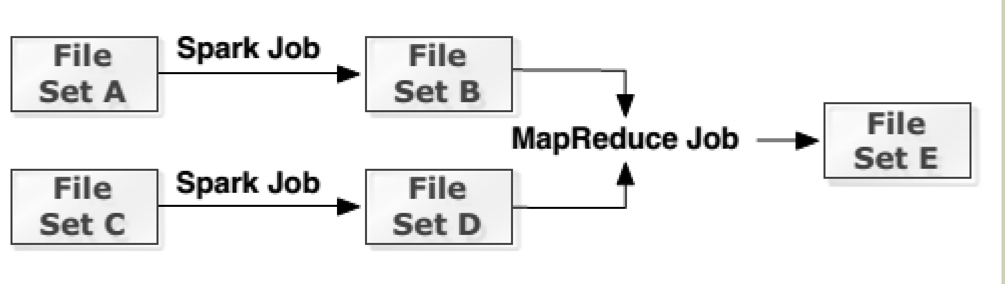
TACHYON: KEY ARCHITECTURAL DETAILS

- Two layers
  - Lineage Layer: In-memory Tachyon daemons
  - Persistent Layer: Replication-based storage systems (HDFS, S3, …)
- Master-slave Architecture
  - Master: Lineage metadata and workflow manager
  - Workers: memory-mapped files in RAMdisk



TACHYON: KEY ARCHITECTURAL DETAILS

- Lineage Overhead:
  - Metadata
  - Job binaries
  - Garbage collected
- Data Eviction
  - Access frequency
  - Access temporal locality



Return	Signature
Global Unique Lineage Id	createDependency(inputFiles, outputFiles, binaryPrograms, executionConfiguration, dependencyType)
Dependency Info	getDependency(lineageId)

TACHYON: KEY ARCHITECTURAL DETAILS

- Master Fault-Tolerance
  - Standby secondary master
  - Zookeeper to elect new master
  - Persistent layer to rebuild from lineage
- Check-pointing
  - Check-point for the entire Tachyon system, not just application
  - Check-point key elements in the lineage chain
  - Check-point hot files
  - Avoid check-point temporary files

TACHYON: KEY ARCHITECTURAL DETAILS

- Resource Allocation
  - How to balance resources between actual work and recomputation work
  - Priority compatibility: recomputation priority follow job priority
  - Resource sharing (no fixed allocation to recomputation)
  - Avoid cascading recomputation

TACHYON: LIMITATION/FUTURE WORK

- Enabling random access abstraction
- Enabling mutable data
- Enabling multi-tenancy (memory sharing among users)
- Hierarchical storage (NVRAM, SSDs …)
- Optimizing checkpoint algorithm

FUTURE DIRECTION

- Discardable Distributed Memory: Supporting Memory Storage in HDFS
- Taking lessons from Spark and Tachyon (Hortonworks)
- To be integrated into HDFS core libraries