Hadoop

The new world of Big Data
(programming model)

Overview

- Background
- Google MapReduce
- The Hadoop Ecosystem
  - Core components:
    - Hadoop MapReduce
    - Hadoop Distributed File System (HDFS)
  - Other selected Hadoop projects:
    - HBase
    - Hive
    - Pig

What is Hadoop?

- Hadoop is an ecosystem of tools for processing “Big Data”.
- Hadoop is an open source project.

The Apache Software Foundation
http://www.apache.org/

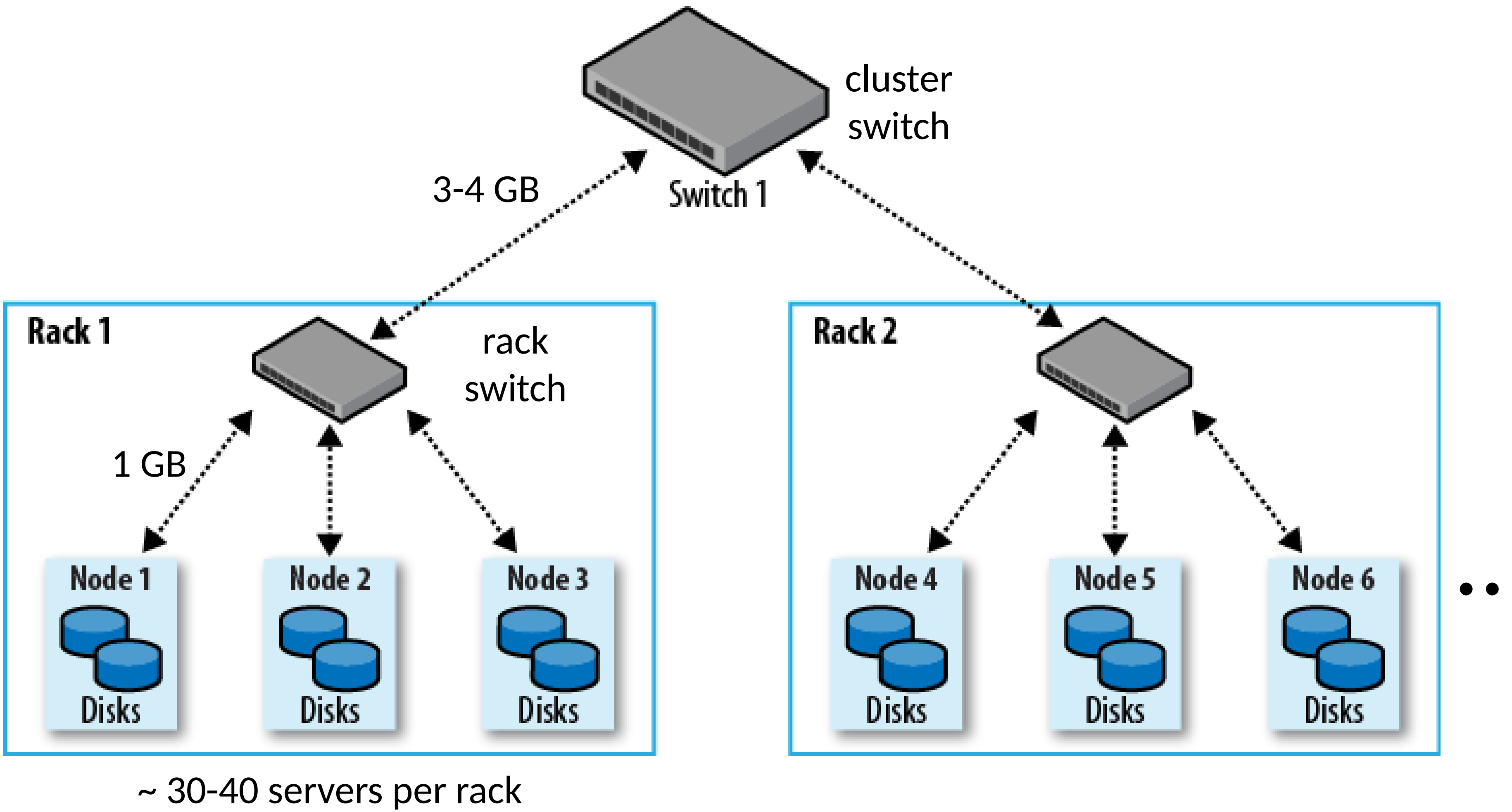
The Hadoop Family

MapReduce	**Distributed computation framework (data processing model and execution environment)**
HDFS	**Distributed file system**
HBase	Distributed, column-oriented database
Hive	Distributed data warehouse
Pig	Higher-level data flow language and parallel execution framework
ZooKeeper	Distributed coordination service
Avro	Data serialization system (RPC and persistent data storage)
Sqoop	Tool for bulk data transfer between structured data stores (e.g., RDBMS) and HDFS
Oozie	Complex job workflow service
Chukwa	System for collecting management data
Mahout	Machine learning and data mining library
BigTop	Packaging and testing

Hadoop: Architectural Design Principles

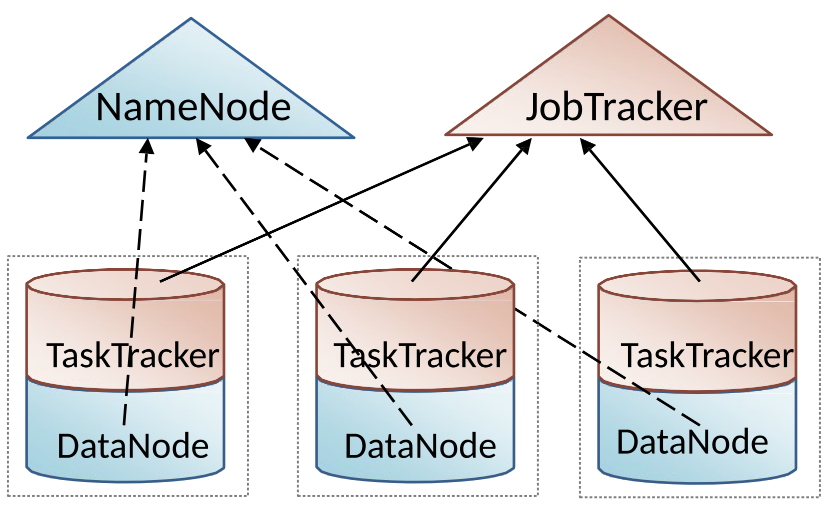
- Linear scalability
  - More nodes can do more work within the same time
  - Linear on data size, linear on compute resources
- Move computation to data
  - Minimize expensive data transfers
  - Data is large, programs are small
- Reliability and Availability: Failures are common
- Simple computational model (MapReduce)
  - Hides complexity in efficient execution framework
- Streaming data access (avoid random reads)
  - More efficient than seek-based data access

A Typical Hadoop Cluster Architecture

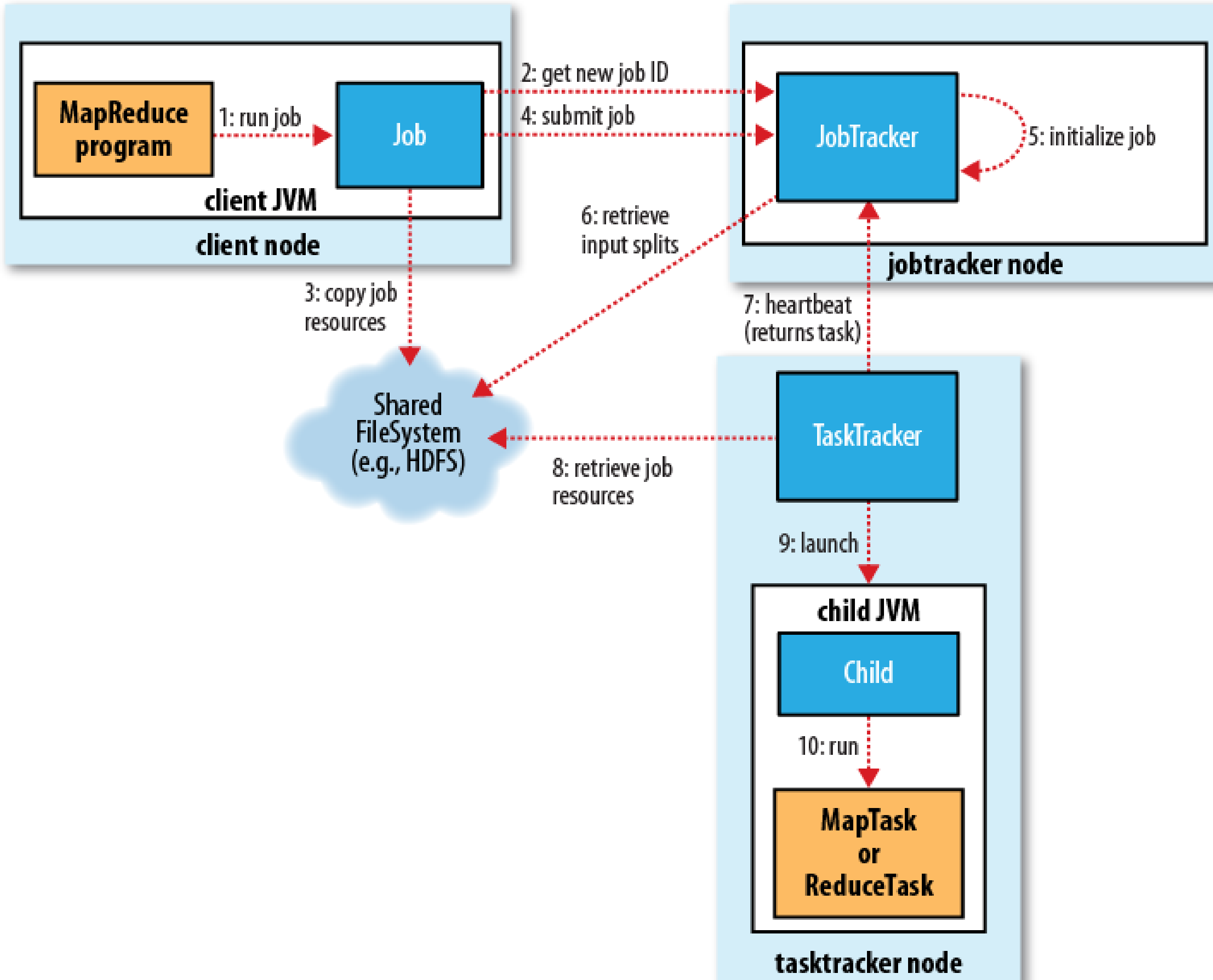


Hadoop Main Cluster Components

- HDFS daemons
  - **NameNode**: namespace and block management (~ master in GFS)
  - **DataNodes**: block replica container (~ chunkserver in GFS)
- MapReduce daemons
  - **JobTracker**: client communication, job scheduling, resource management, lifecycle coordination (~ master in Google MR)
  - **TaskTrackers**: task execution module (~ worker in Google MR)



MapReduce Job Execution in Hadoop



Job Submission (1-4)

- Client submits MapReduce job through Job.submit() call
- Job submission process
  - Get new job ID from JobTracker
  - Determine input splits for job
  - Copy job resources (job JAR file, configuration file, computed input splits) to HDFS into directory named after the job ID
  - Inform JobTracker that job is ready for execution

Job Initialization (5-6)

- JobTracker puts ready job into internal queue
- Job scheduler picks job from queue
  - Initializes it by creating job object
  - Creates list of tasks
    - One map task for each input split
    - Number of reduce tasks determined by mapred.reduce.tasks property in Job, which is set by setNumReduceTasks()
- Tasks need to be assigned to worker nodes

Task Assignment (7)

- TaskTrackers send heartbeats to JobTracker
  - Indicate if ready to run new tasks
  - Number of “slots” for tasks depends on number of cores and memory size
- JobTracker replies with new task
  - Chooses task from first job in priority-queue
    - Chooses map tasks before reduce tasks
    - Chooses map task whose input split location is closest to machine running the TaskTracker instance (data-local < rack-local < off-rack; data locality optimization)
  - Could also use other scheduling policy

Task Execution (8-10)

- TaskTracker copies job JAR and other configuration data from HDFS to local disk
- Creates local working directory
- Creates TaskRunner instance
- TaskRunner launches new JVM (or reuses one from another task) to execute the JAR

Monitoring Job Progress

- Tasks report progress to TaskTracker
- TaskTracker includes task progress in heartbeat message to JobTracker
- JobTracker computes global status of job progress
- JobClient polls JobTracker regularly for status
- Visible on console and web UI

Handling Task Failures

- Error reported to TaskTracker and logged
- Hanging task detected through timeout
- JobTracker will automatically re-schedule failed tasks
  - Tries up to mapred.map.max.attempts many times (similar for reduce)
  - Job is aborted when task failure rate exceeds mapred.max.map.failures.percent (similar for reduce)

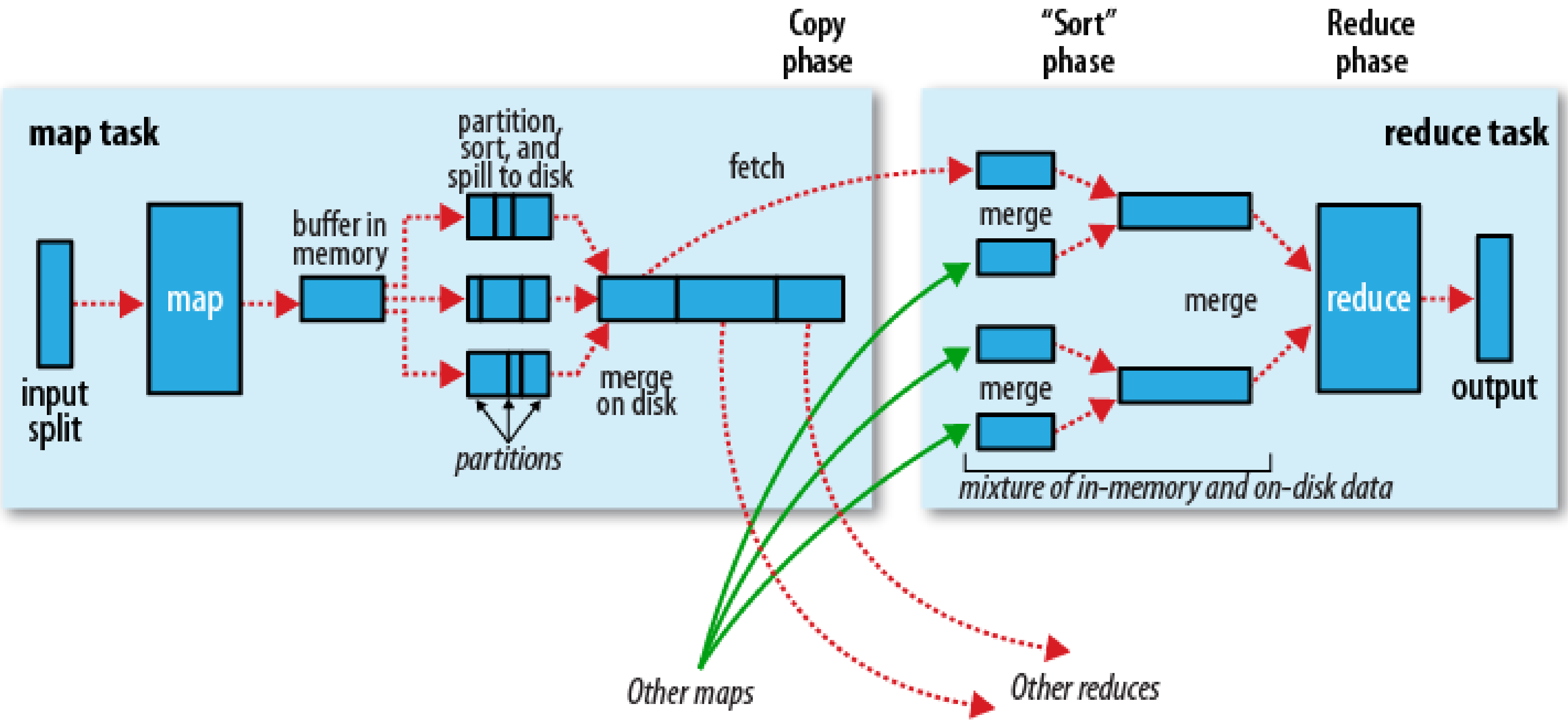
Handling TaskTracker & JobTracker Failures

- TaskTracker failure detected by JobTracker from missing heartbeat messages
  - JobTracker re-schedules map tasks and not completed reduce tasks from that TaskTracker
- Hadoop cannot deal with JobTracker failure
  - Could use Google's proposed JobTracker take-over idea, using ZooKeeper to make sure there is at most one JobTracker
  - Improvements in progress in newer releases…

Moving Data from Mappers to Reducers

- “Shuffle & Sort” phase
  - synchronization barrier between map and reduce phase
  - one of the most expensive parts of a MapReduce execution
- Mappers need to separate output intended for different reducers
- Reducers need to collect their data from all mappers and group it by key
  - keys at each reducer are processed in order

Shuffle & Sort Overview



Hadoop Assessment

- Very I/O intensive
  - write intermediate results to disk
  - great for fault tolerance, but poor performance
- Idea: Keep intermediate results in memory
  - Resilient Data Sets: lineage of how to recompute
  - Key idea of Spark (UC Berkeley, AMP Lab)
  - Much better performance, okay fault tolerance
- Many other wrinkles in Hadoop implementation
  - expect 10x performance with RDBMS

Combiner Functions

- Pre-reduces mapper output before transfer to reducers (to minimize data transferred)
- Does not change program semantics
- Usually same as reduce function, but has to have same output type as Map
- Works only for certain types of reduce functions (commutative and associative (a.k.a. distributive))
  - E.g.: max(5, 4, 1, 2) = max(max(5, 1), max(4, 2))

Partitioner Functions

- Partitioner determines which keys are assigned to which reduce task
- Default HashPartitioner essentially assigns keys randomly
- Create custom partitioner by implementing your own getPartition() method of Partitioner in org.apache.hadoop.mapreduce

MapReduce Development Steps

- Write Map and Reduce functions
  - Create unit tests
- Write driver program to run a job
  - Can run from IDE with small data subset for testing
  - If test fails, use IDE for debugging
  - Update unit tests and Map/Reduce if necessary
- Once program works on small test set, run it on full data set
  - If there are problems, update tests and code accordingly
- Fine-tune code, do some profiling

Local (Standalone) Mode

- Runs same MapReduce user program as cluster version, but does it sequentially on a single machine
- Does not use any of the Hadoop daemons
- Works directly with local file system
  - No HDFS, hence no need to copy data to/from HDFS
- Great for development, testing, initial debugging

Pseudo-Distributed Mode

- Still runs on a single machine, but simulating a real Hadoop cluster
  - Simulates multiple nodes
  - Runs all daemons
  - Uses HDFS
- For more advanced testing and debugging
- You can also set this up on your laptop

Programming Language Support

- Java API (native)
- Hadoop Streaming API
  – allows writing map and reduce functions in any programming language that can read from standard input and write to standard output
  – Examples: Ruby, Python
- Hadoop Pipes API
  – allows map and reduce functions written in C++ using sockets to communicate with Hadoop's TaskTrackers