Indian Institute of Science
Bangalore, India
भारतीय विज्ञान संस्थान
बंगलौर, भारत

SE256:Jan16 (2:1)

L5-6:Runtime Platforms
Hadoop and HDFS

Yogesh Simmhan

03/10 Feb, 2016

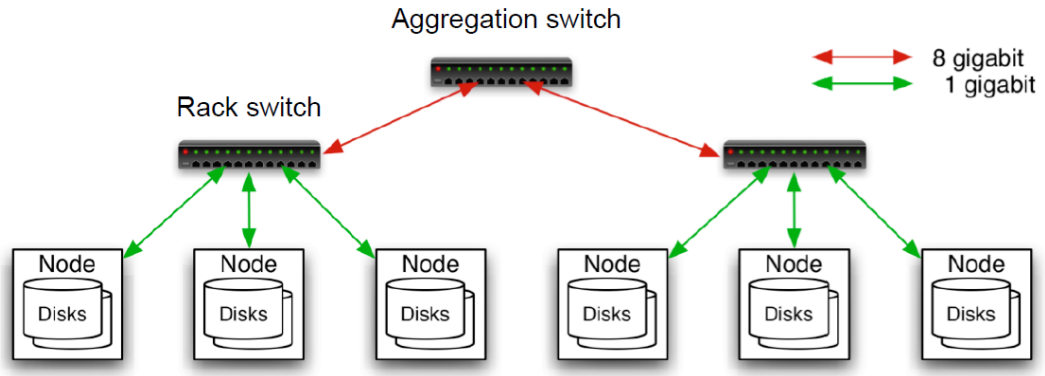
©Yogesh Simmhan & Partha Talukdar, 2016
This work is licensed under a Creative Commons Attribution 4.0 International License
Copyright for external content used with attribution is retained by their original authors

Learning Objectives

1. *How* does **HDFS** work? *Why* is it effective?
2. *How* does Hadoop **MapReduce** work? *Why* is it effective?
3. Optimizations for performance and reliability in Hadoop MR

Data Centre Architecture *Recap*

- Commodity hardware
  - 1000's machines of medium performance and reliability
  - Failure is a given. Design to withstand failure.
- Network bottlenecks
  - Hierarchical network design
  - Push compute to data



Data Centre Architecture *Recap*

- I/O bottlenecks & failure
  - Multiple disks for cumulative bandwidth
  - Data redundancy: Hot/Hot
- Example: How long to read 150TB of CC data?
  - Say you can store 150GB in a SATA disk
    - SATA Disk bandwidth is 3Gbps → 111hrs
  - Say you have 50 disks of 3Gbps, each with 3TB
    - I/O controller in node handles 10Gbps → 33hrs to read
  - Say cluster with 12 nodes, 4 disks of 3TB each
    - (150/12)TB *1000*8bits/10Gbps = 2.7hrs to read
  - Say cluster with 12 nodes, dual Ethernet, reading over network
    - (150/12)TB*1000*8/2Gbps = 13.9hrs to read
- *Time to read across network is not very different from time to read from stressed disk*

GrayWulf, Scalable Clustered Architecture for Data Intensive Computing, Szalay, HICSS, 2008

E.g. Open Cloud Server

- High density: 24 blades / chassis, 96 blades / rack
- Compute blades
  - Dual socket, 4 HDD, 4 SSD
  - 16-32 CPU cores
  - 4-16TB HDD/SSD
- JBOD Blade
  - 10 to 80 HDDs, 6G or 12G SAS
  - 40-160TB HDD

http://www.opencompute.org/wiki/Motherboard/SpecsAndDesigns

Class Cluster

- Nodes
  - 8 core AMD Opteron 3380, 2.6GHz
  - 32GB DDR3
  - 2TB HDD
  - 1Gbps LAN
- 12 nodes, 3U
- 1 Gigabit within switch, 10Gbps across switches

http://www.supermicro.com/Aplus/system/3U/3012/AS-3012MA-H12TRF.cfm

Cisco’s Data Center in Texas

Google's Data Center in Georgia

Microsoft's Data Center in Ireland

NSA’s Data Center in Utah

Who is this?

Doug Cutting and *Hadoop* the elephant

Hadoop was created by Doug Cutting (Yahoo) and Mike Cafarella (UW) in 2006.
Cutting's son, then 2, was just beginning to talk and called his beloved stuffed yellow elephant "Hadoop" (with the stress on the first syllable).

http://www.cnbc.com/id/100769719
https://en.wikipedia.org/wiki/Apache_Hadoop#History

Hadoop: Big Picture Interactions

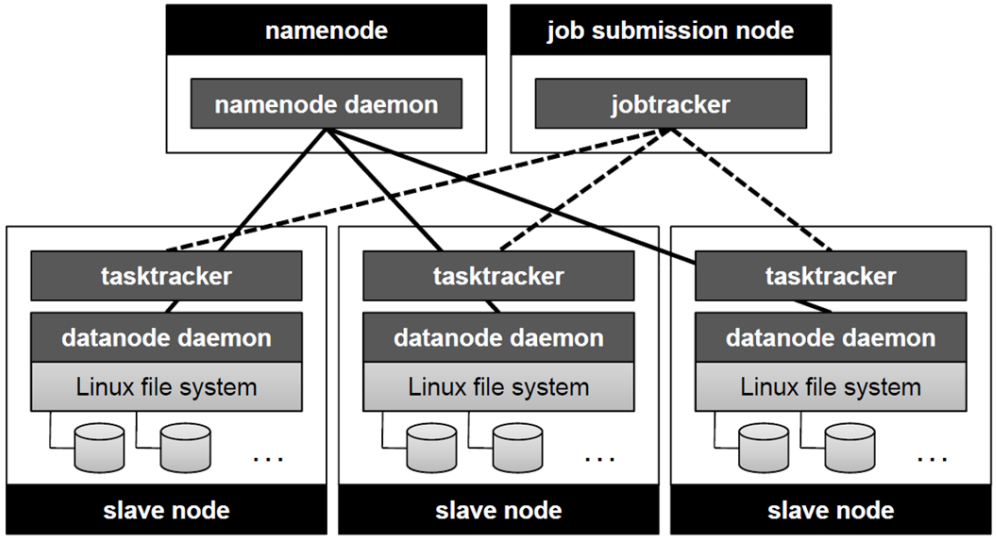


Figure 2.6: Architecture of a complete Hadoop cluster, which consists of three separate components: the HDFS master (called the namenode), the job submission node (called the jobtracker), and many slave nodes (three shown here). Each of the slave nodes runs a tasktracker for executing map and reduce tasks and a datanode daemon for serving HDFS data.

Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer, 2010

Hadoop: Big Picture Interactions

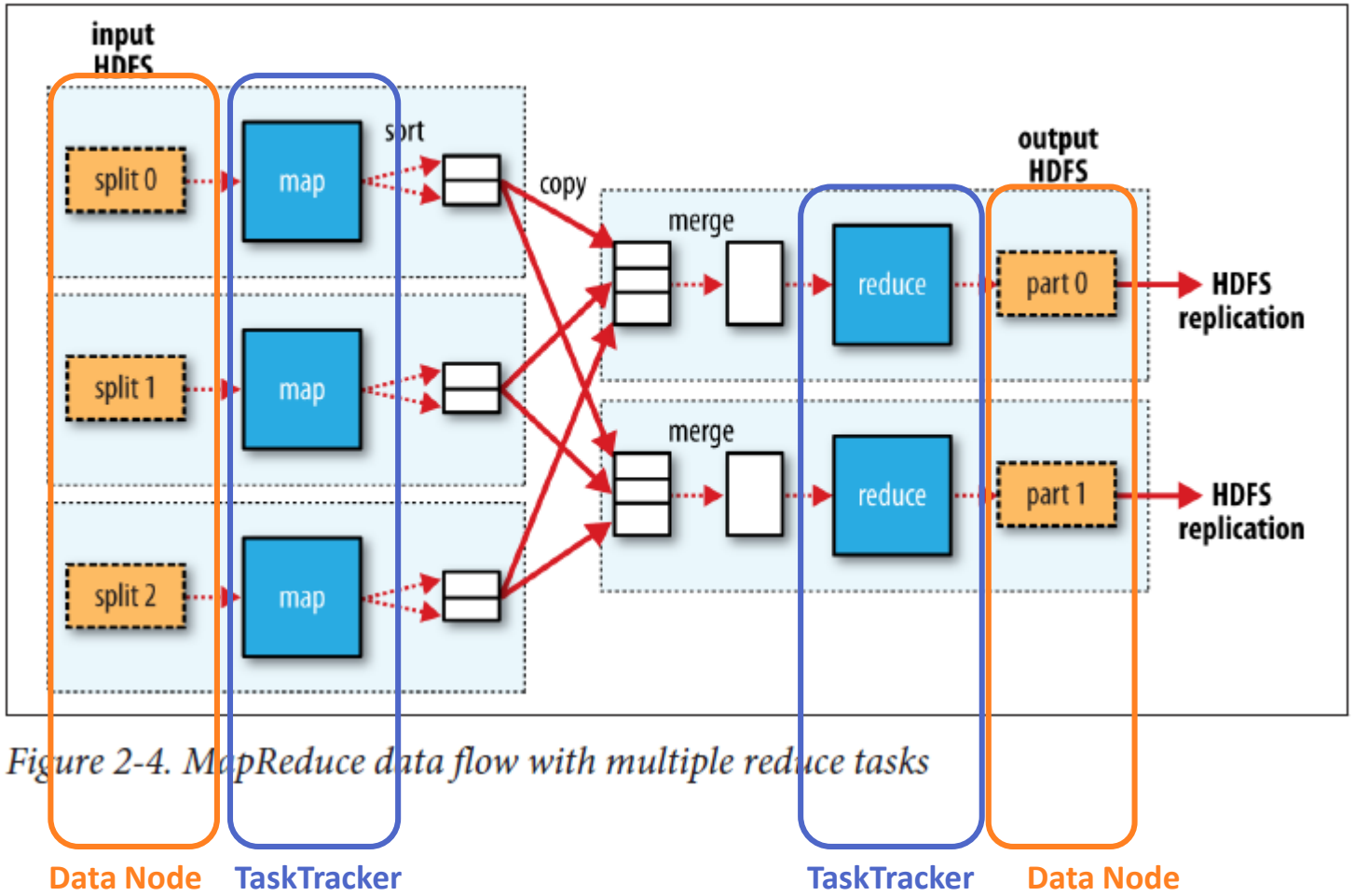


Figure 2-4. MapReduce data flow with multiple reduce tasks

Hadoop: The Definitive Guide, **4th Edition**, 2015

Hadoop Distributed File System

2016-02-10

Hadoop Distributed File System (HDFS)

- Based on Google File System (GFS)
- Optimized for huge files
- Write once, read many
  - Create new data. Never update-in-place, only append.
  - No write locks needed. Initial cost of writes is amortized.
- Optimized for sequential reads
  - Typically, start at a point and read to completion
- Throughput (cumulative bandwidth) favoured over low latency
  - Low total time for all data than time per small files
- Survive high disk/node failures
  - Both persistence, availability

The Google File System, Sanjay Ghemawat, et al, SOSP, 2003

HDFS File Distribution

- Files are split into *blocks* of equal size
  - Unit of data that can be read or written
  - Block sizes are large, e.g. 128MB
  - Blocks themselves are persisted on local disks, e.g. using POSIX file system
  - Blocks are replicated
- Advantages
  - Larger reads/writes (throughput)
  - Files can be larger than single disk
  - Eases distributed management
    - Same size, opaque content, complexity pushed up.
    - Unit of recovery, replication

HDFS Design

- Master-slave architecture
  - Master manages namespace, directory/file names/tree structure, metadata, block ids, permissions
  - Slave manages blocks containing data

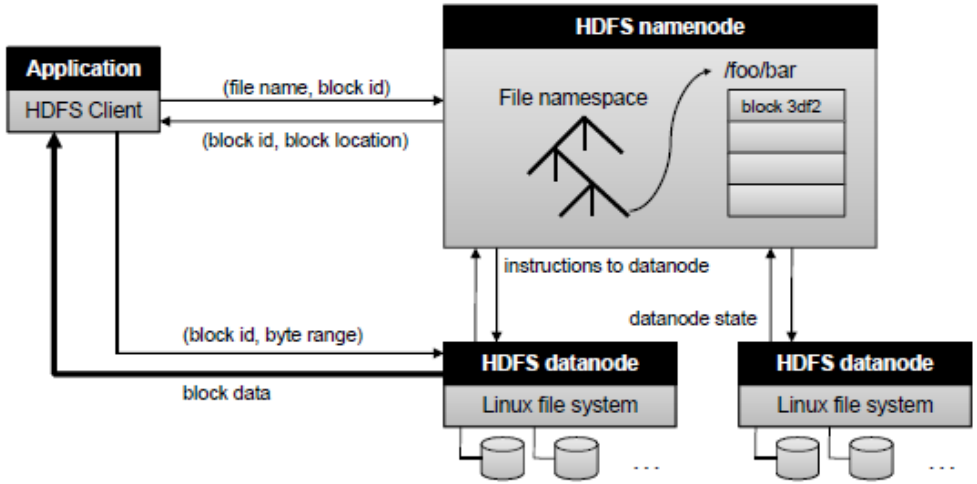


Figure 2.5: The architecture of HDFS. The namenode (master) is responsible for maintaining the file namespace and directing clients to datanodes (slaves) that actually hold data blocks containing user data.

Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer, 2010

Master: Name Node

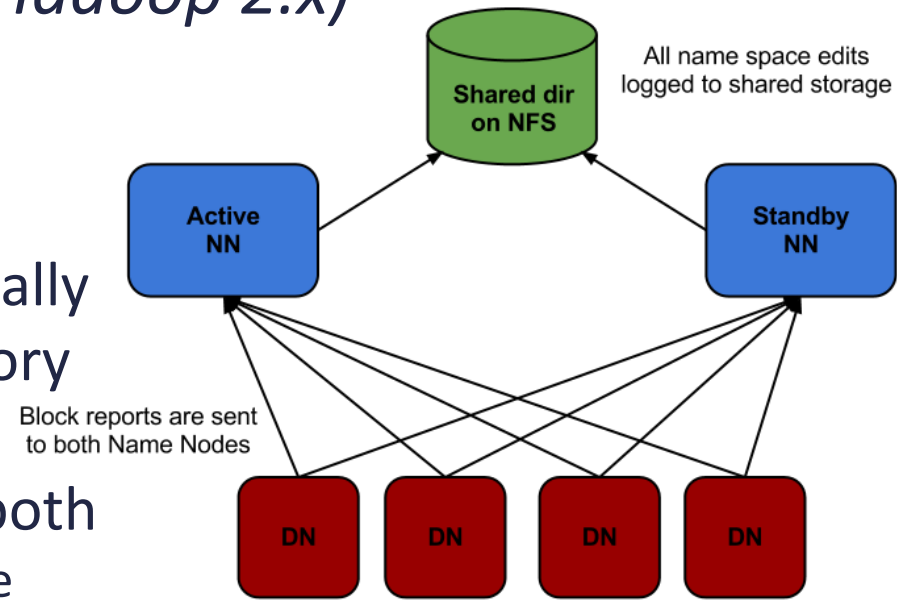
- Persists names, trees, metadata, permissions on disk
  - Namespace image (fsimage)
  - Edit log of deltas (rename, permission, create)
  - Mapping from files to list of blocks
- Security is not a priority
- Block location not persistent, kept in-memory
  - Mapping from blocks to locations is *dynamic*
    - *Why?*
  - *Reconstructs* location of blocks from data nodes

Master: Name Node

- Detects health of FS
  - Is data node alive?
  - Is data block under-replicated?
  - Rebalancing block allocation across data nodes, improved disk utilization
- Coordinates file operations
  - Directs application clients to datanodes for reads
  - Allocates blocks on datanodes for writes

Master: Name Node

- Single Point of failure! *(Hadoop 1.x)*
  - Upgrades → Downtime
  - Failure→ Data loss *(file names, file:block ID mapping)*
  - Secondary NameNode (Stale backup), multiple FS
- NameNode High Availability *(Hadoop 2.x)*
  - Cold standby can take time
    - 10mins to load, 1hr for block list
  - Active acks after write to NFS
  - Hot standby refreshed periodically
  - Loads current status into memory
  - Shared NFS should be reliable!
  - DataNodes send heartbeat to both
    - But ops received only from active



http://blog.cloudera.com/blog/2012/03/high-availability-for-the-hadoop-distributed-file-system-hdfs/

Slave/Worker: Data Node

- Store & retrieve blocks
- Respond to client and master requests for block operations
- Sends heartbeat every 3 secs for liveliness
- Periodically sends list of block IDs and location on that node
  - Piggyback on heartbeat message
  - e.g., send block list every hour

File Reads

- Clients get list of data nodes locations for each block from NameNode, sorted by distance
- Blocks read in order
  - Connection opened and closed to nearest DataNode for each block
  - Tries alternate on network failure, checksum failure

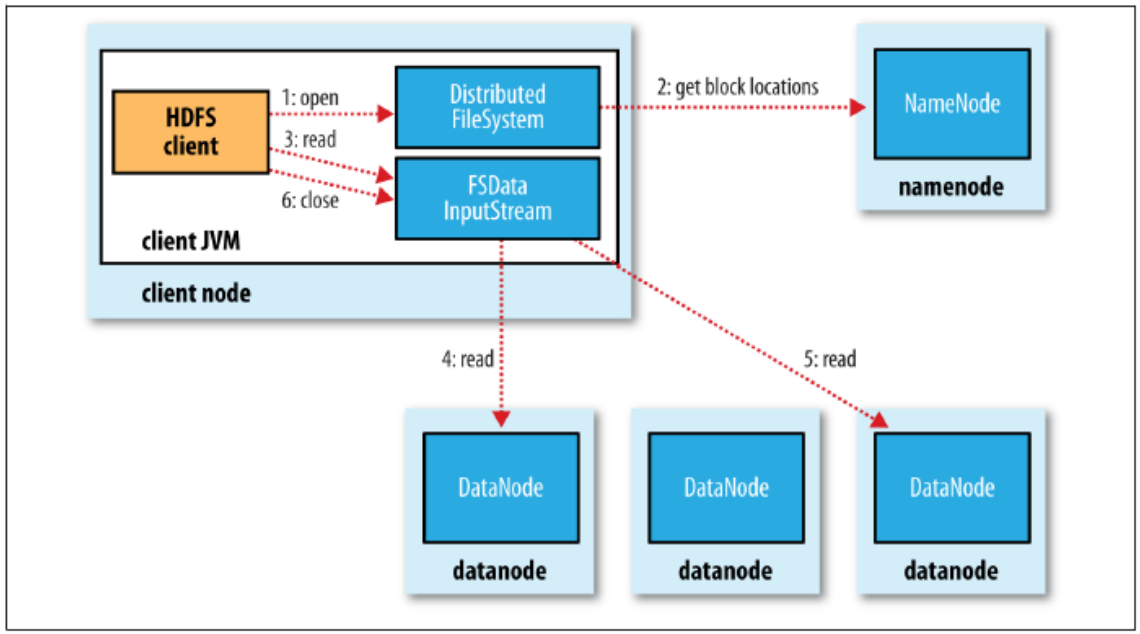
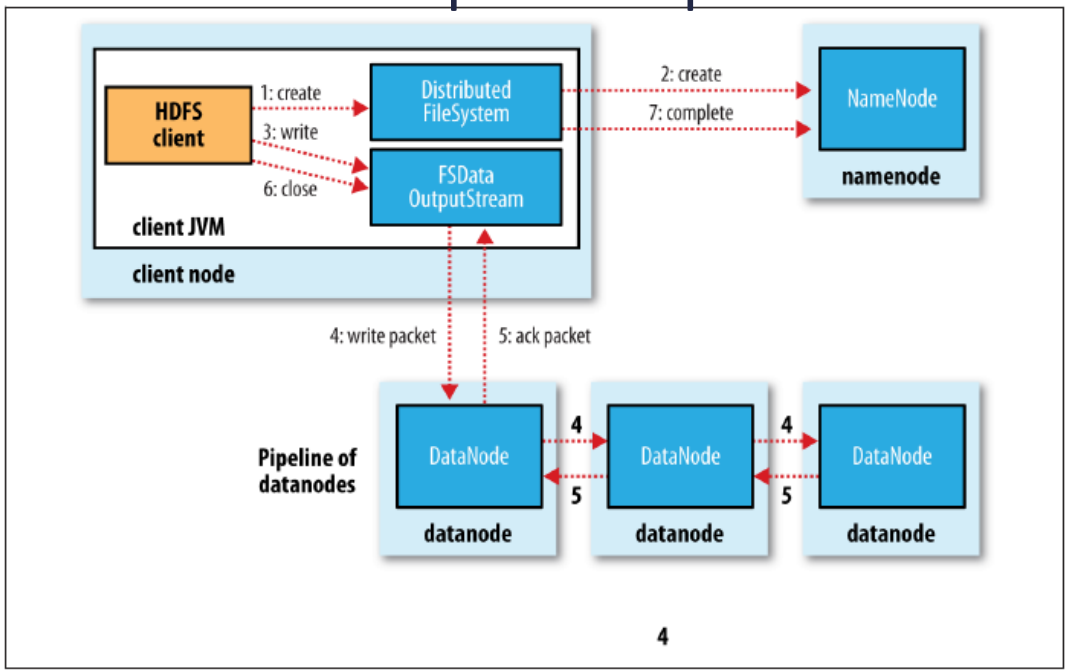


Figure 3-2. A client reading data from HDFS

Hadoop: The Definitive Guide, 4th Edition, 2015

File Writes

- Clients get list of data nodes to store a block's replica
  - First copy on same data node as client, or random.
  - Second is off-rack. Third on same rack as second.
- Blocks written in order. Forwarded in a pipeline. Acks from all replicas expected before next block written.



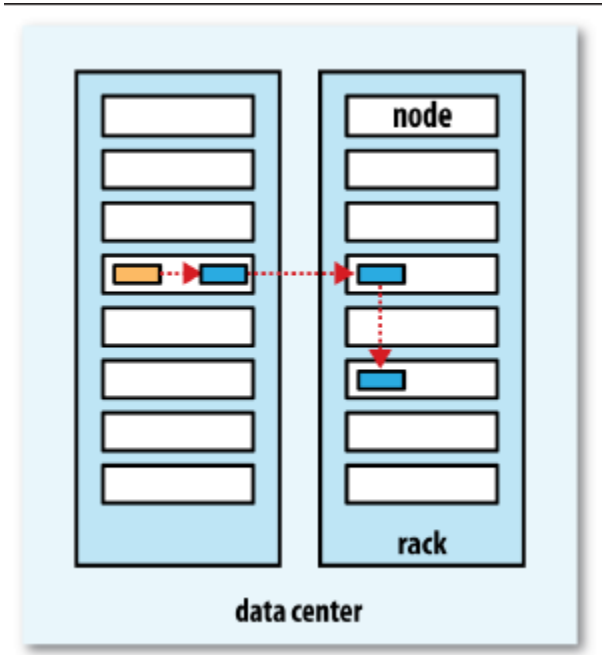


Figure 3-4. A client writing data to HDFS

Hadoop YARN

Yet Another Resource Negotiator

2016-02-10

MapReduce v1 → MapReduce v2 (YARN)

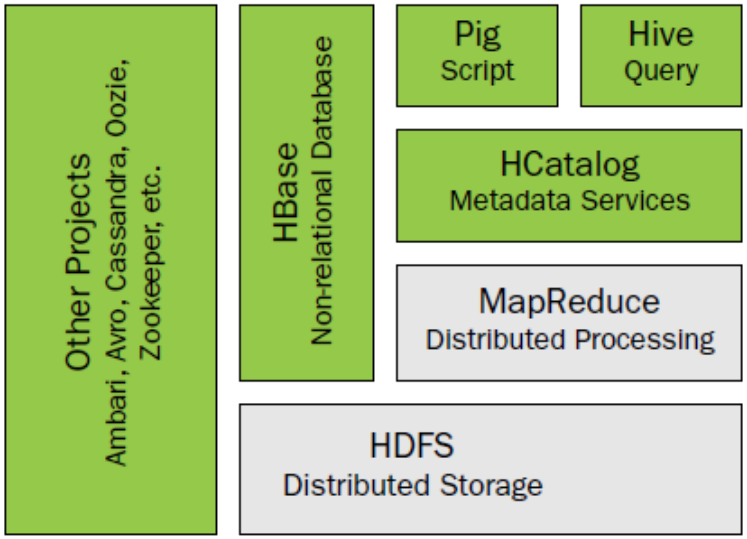


Figure 3.1 The Hadoop 1.0 ecosystem. MapReduce and HDFS are the core components, while other components are built around the core.

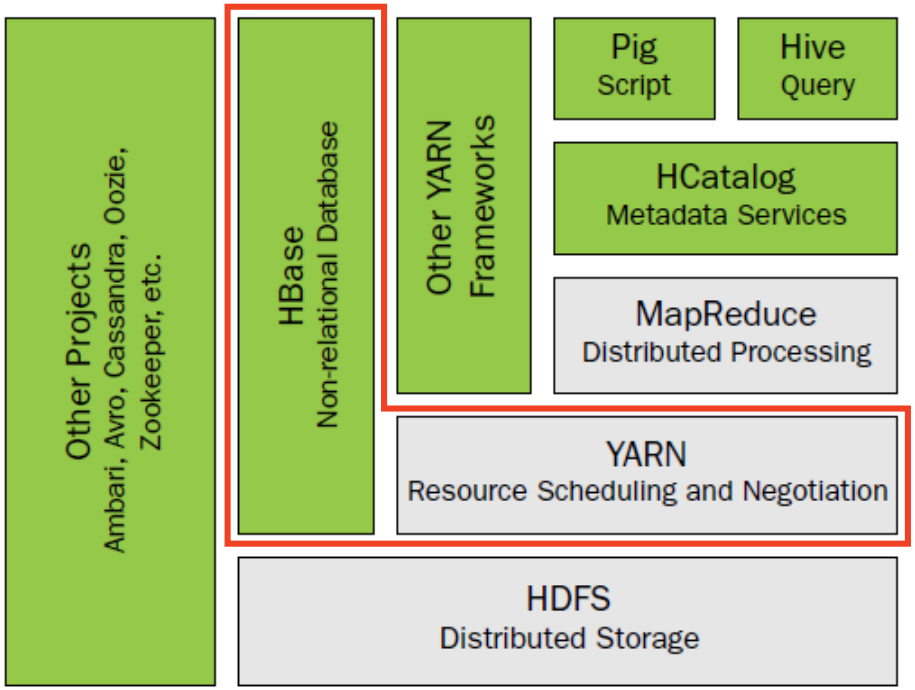


Figure 3.2 YARN adds a more general interface to run non-MapReduce jobs within the Hadoop framework

Apache Hadoop YARN, Arun C. Murthy, et al, HortonWorks, Addison Wesley, 2014

YARN

- Designed for scalability
  - 10k nodes, 400k tasks
- Designed for availability
  - Separate application management from resource management
- Improve utilization
  - Flexible slot allocation. Slots not bound to Map or Reduce types.
- Go beyond MapReduce

YARN

- ResourceManager for cluster
  - Keeps track of nodes, capacities, allocations
  - Failure and recovery (heartbeats)
- Coordinates scheduling of jobs on the cluster
  - Decides which node to allocate to a job
  - Ensures load balancing
- Used by programming frameworks to schedule distributed applications
  - MapReduce, Spark, etc.
- NodeManager
  - Offers slots with given capacity on a host to schedule tasks
  - Container maps to one or more slots

YARN Application Lifecycle

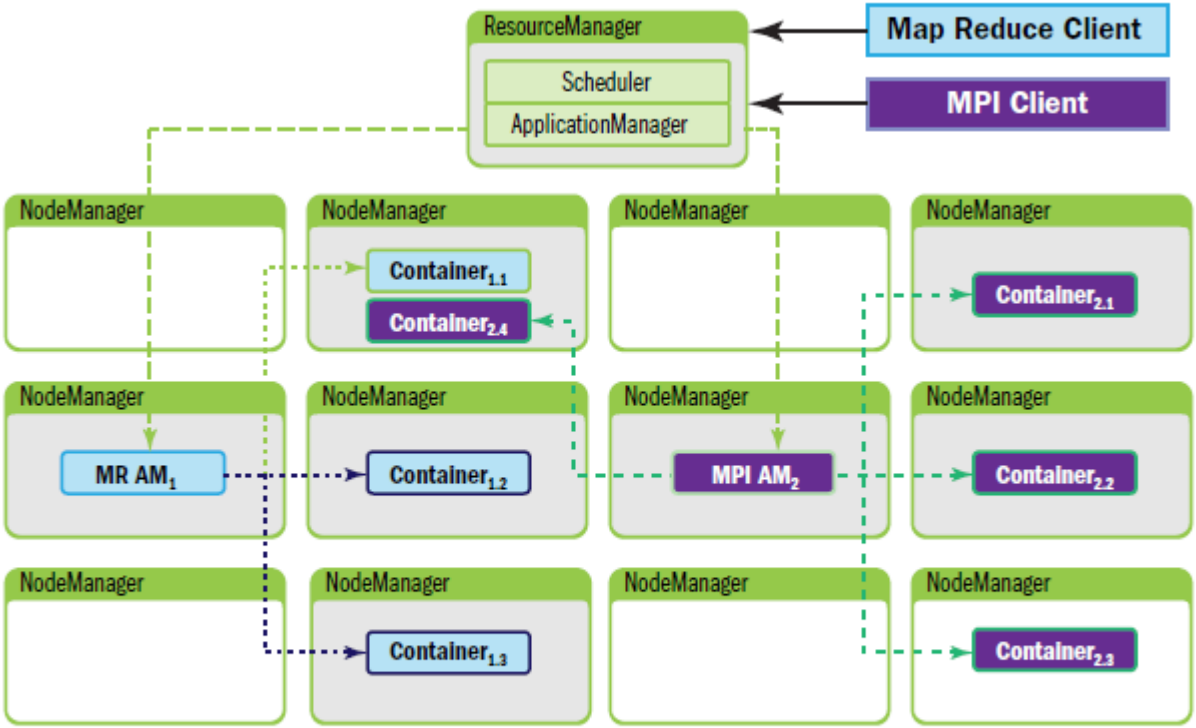


Figure 4.1 YARN architecture with two clients (MapReduce and MPI). The client MPI AM_2 is running an MPI application and the client MR AM_1 is running a MapReduce application.

v1 vs. v2 Application Lifecycle

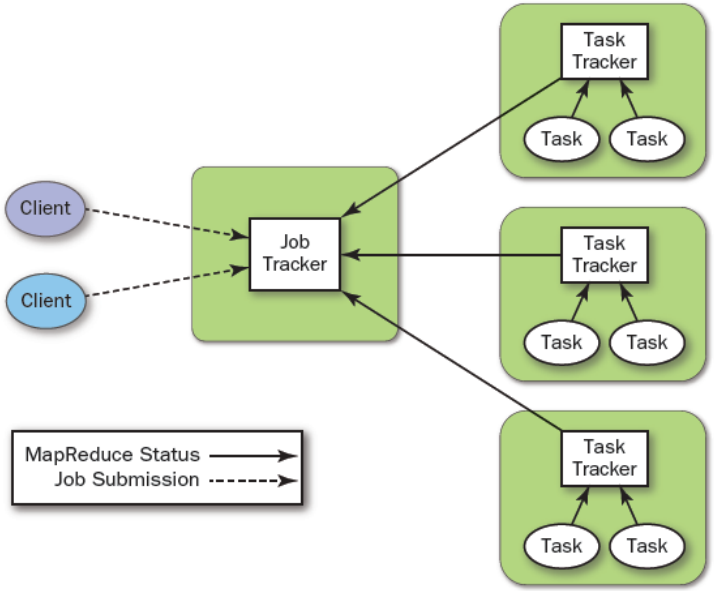


Figure 3.3 Current Hadoop MapReduce control elements

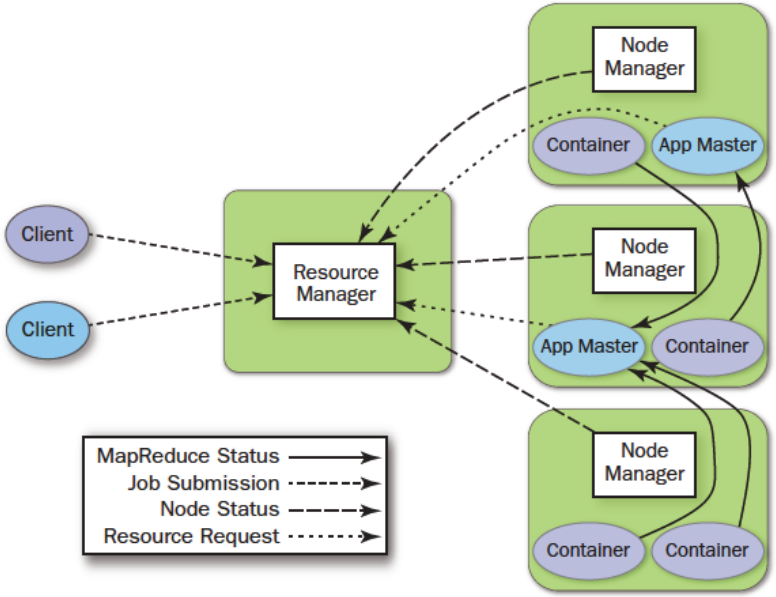


Figure 3.4 New YARN control elements

Apache Hadoop YARN, Arun C. Murthy, et al, HortonWorks, Addison Wesley, 2014

MapReduce Job Lifecycle

- Container requests are aware of block locality

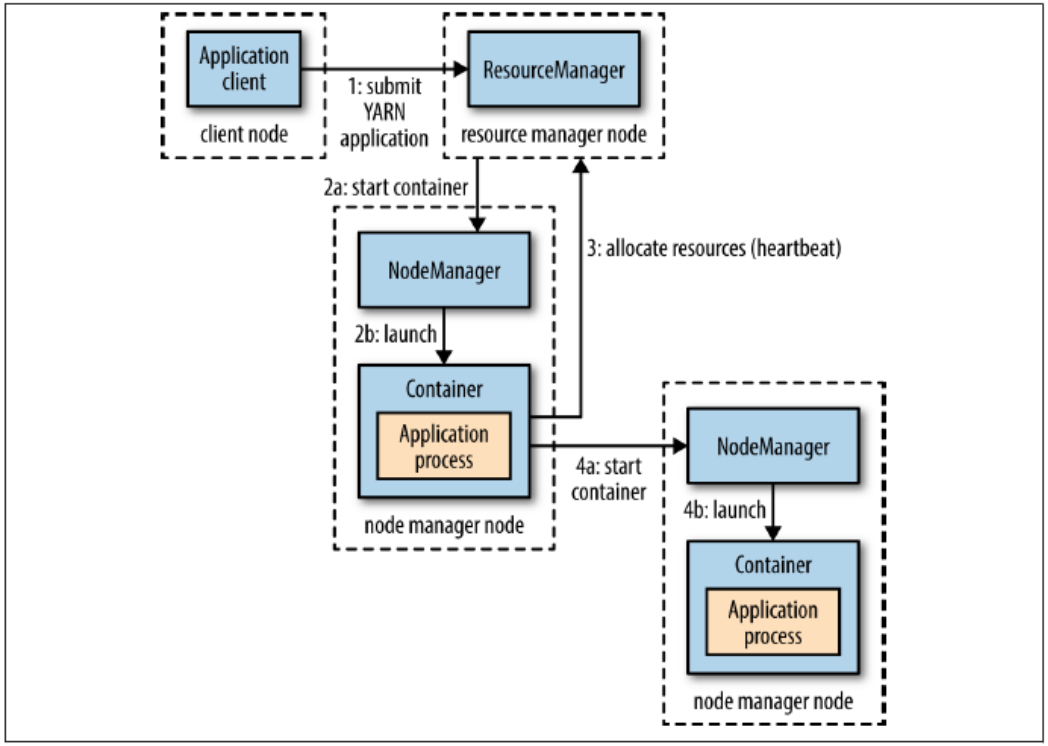


Figure 4-2. How YARN runs an application

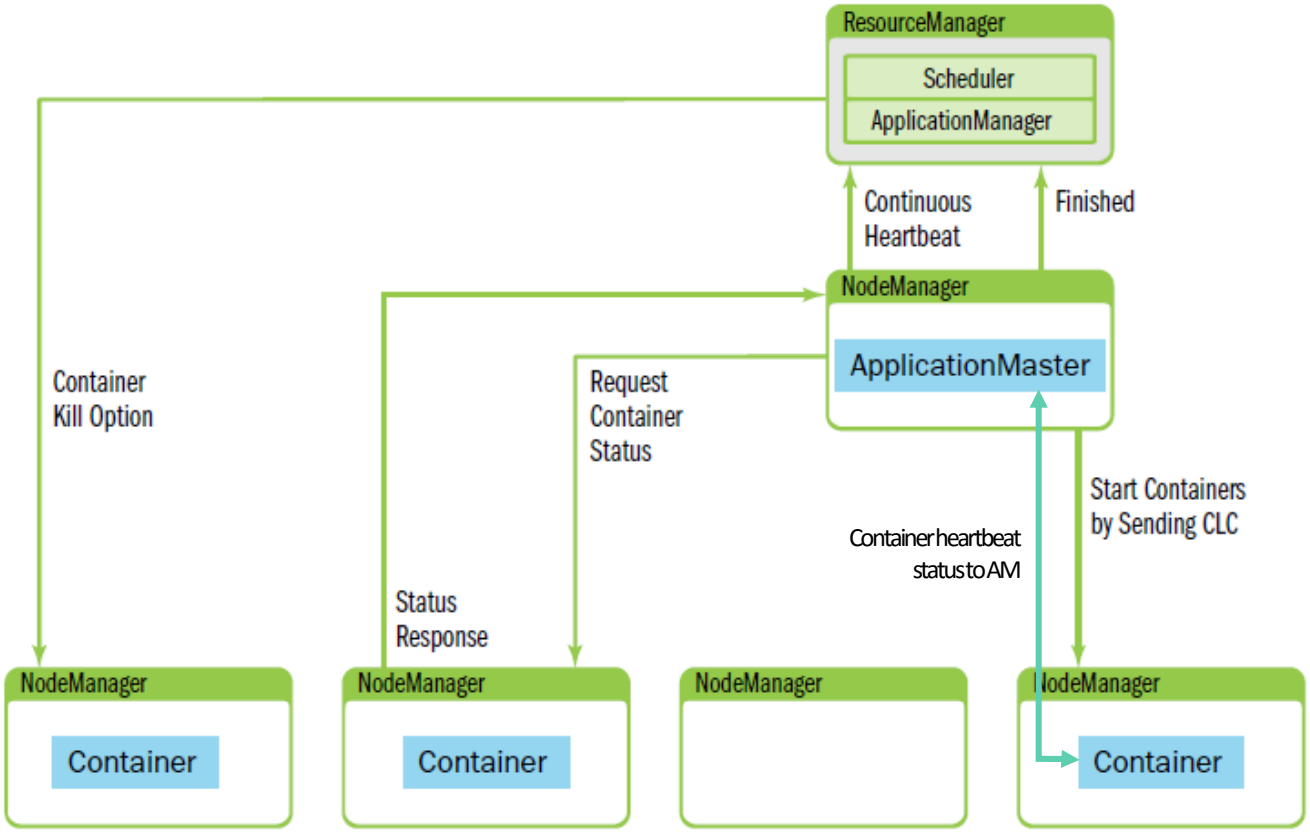


Figure 4.3 ApplicationMaster NodeManager interaction.

Scheduling in YARN

- Scheduler has narrow mandate
- FIFO, as soon as resource available
- Capacity
  - using different queues, min capacity per queue
  - Allocate excess resource to more loaded
- Fair
  - Give all available
  - Redistribute as jobs arrive

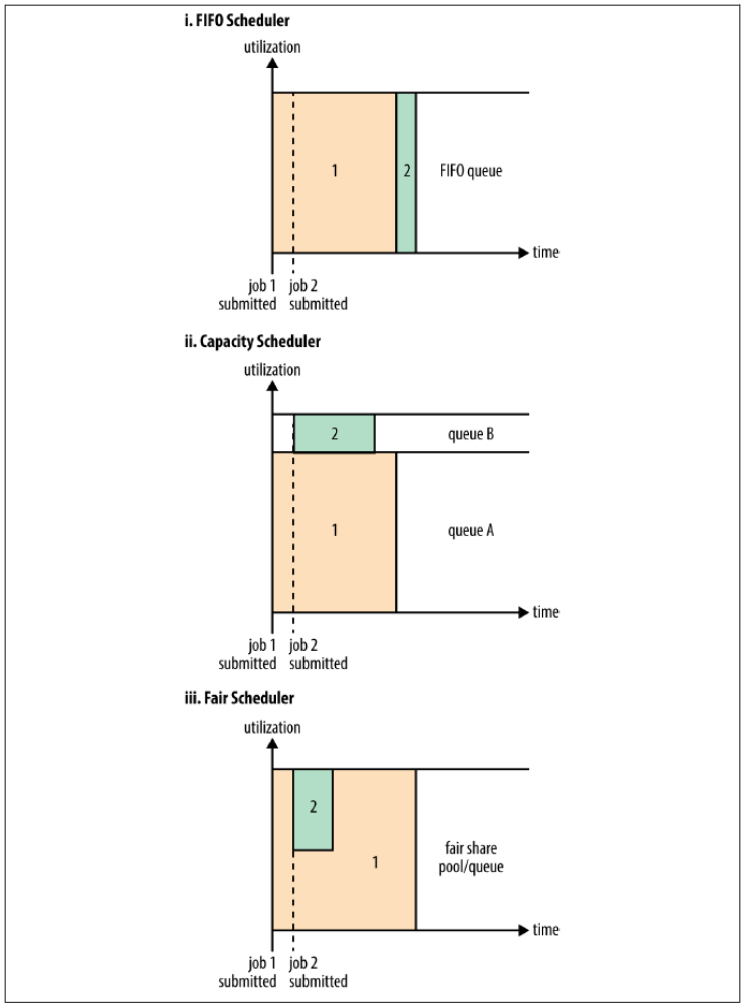


Figure 4-3. Cluster utilization over time when running a large job and a small job under the FIFO Scheduler (i), Capacity Scheduler (ii), and Fair Scheduler (iii)

Hadoop: The Definitive Guide, **4th Edition**, 2015

Hadoop MapReduce

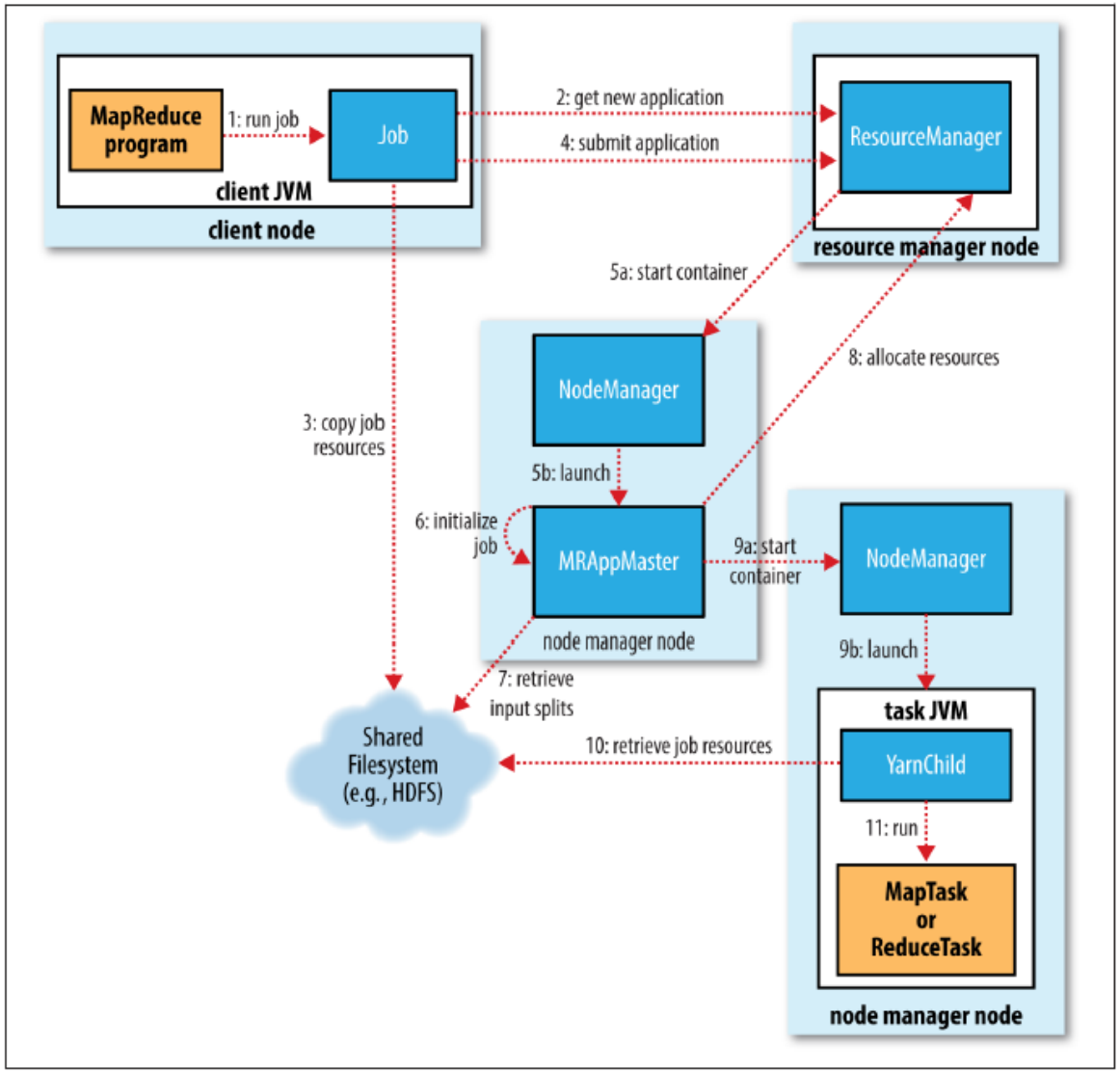


Figure 7-1. How Hadoop runs a MapReduce job

Mapping tasks to blocks

- Data blocks converted to one or more "*splits*"
  - Typically, 1 split per block … reduce task creation overhead vs. overwhelm single task
- Each split handled by a single Mapper task
- *Records* read from each split, forms Key-Value pair input to Map function
- Compute tasks are moved to data *block* location, if possible
  - Enhances data locality

Mapping tasks to blocks

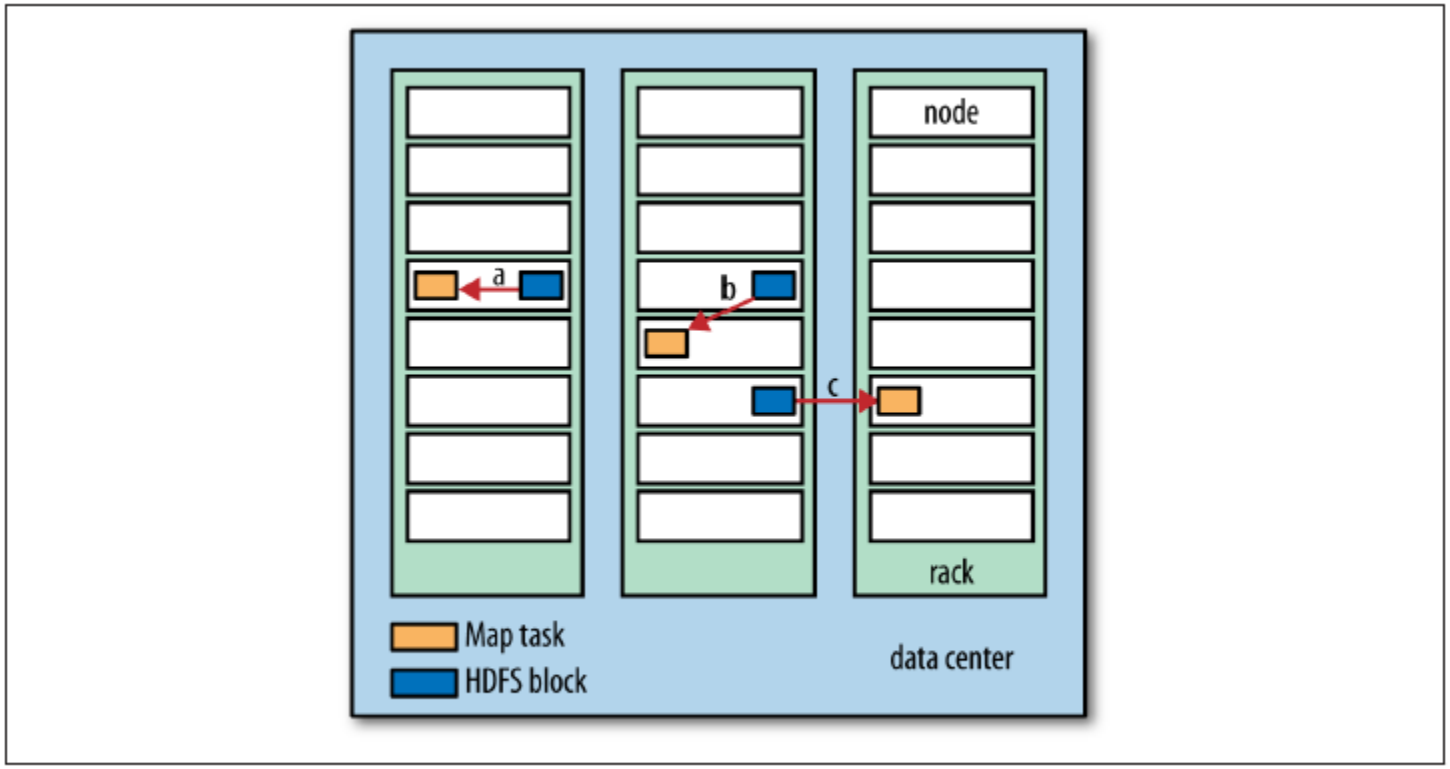


Figure 2-2. Data-local (a), rack-local (b), and off-rack (c) map tasks

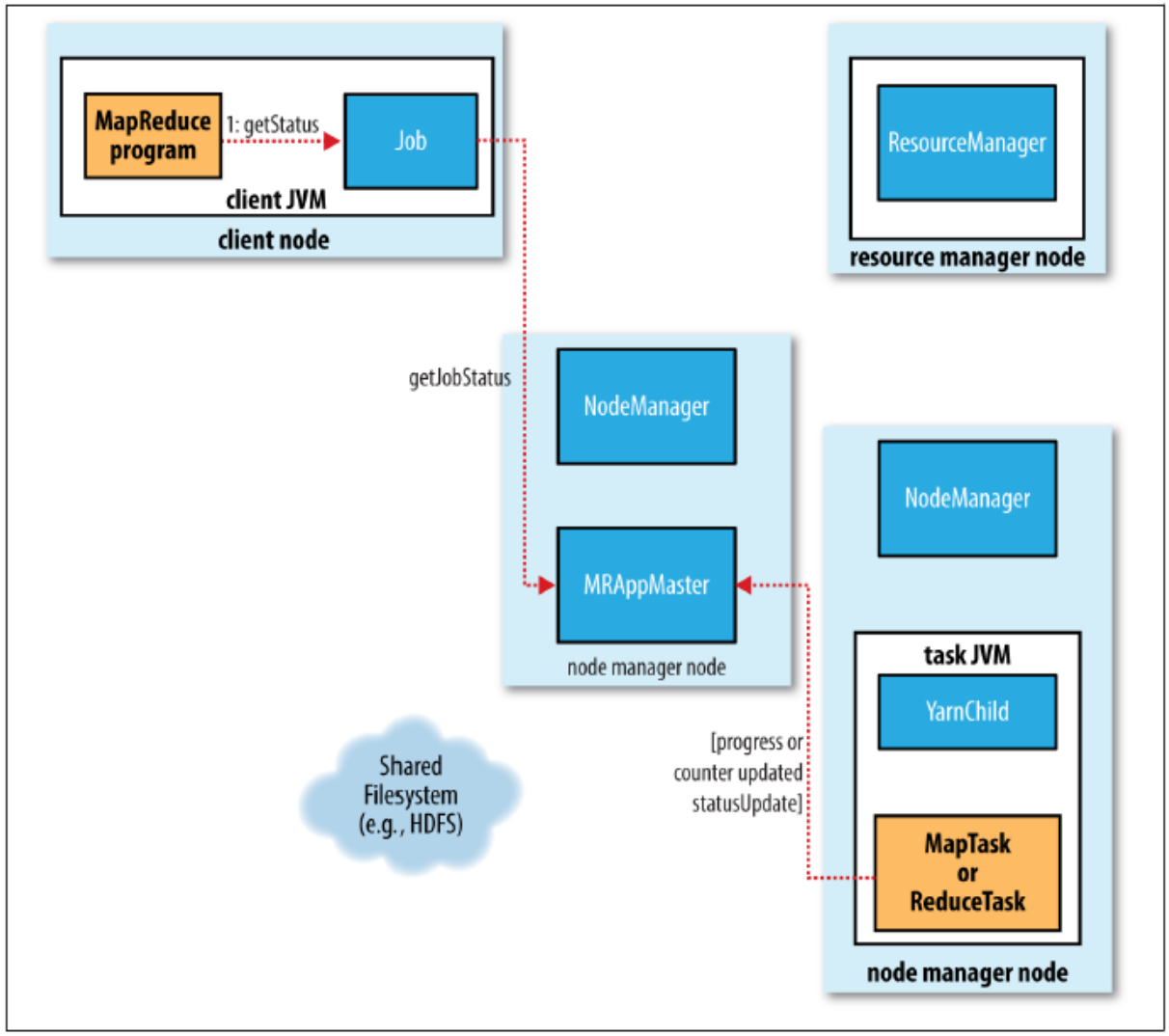


Figure 7-3. How status updates are propagated through the MapReduce system

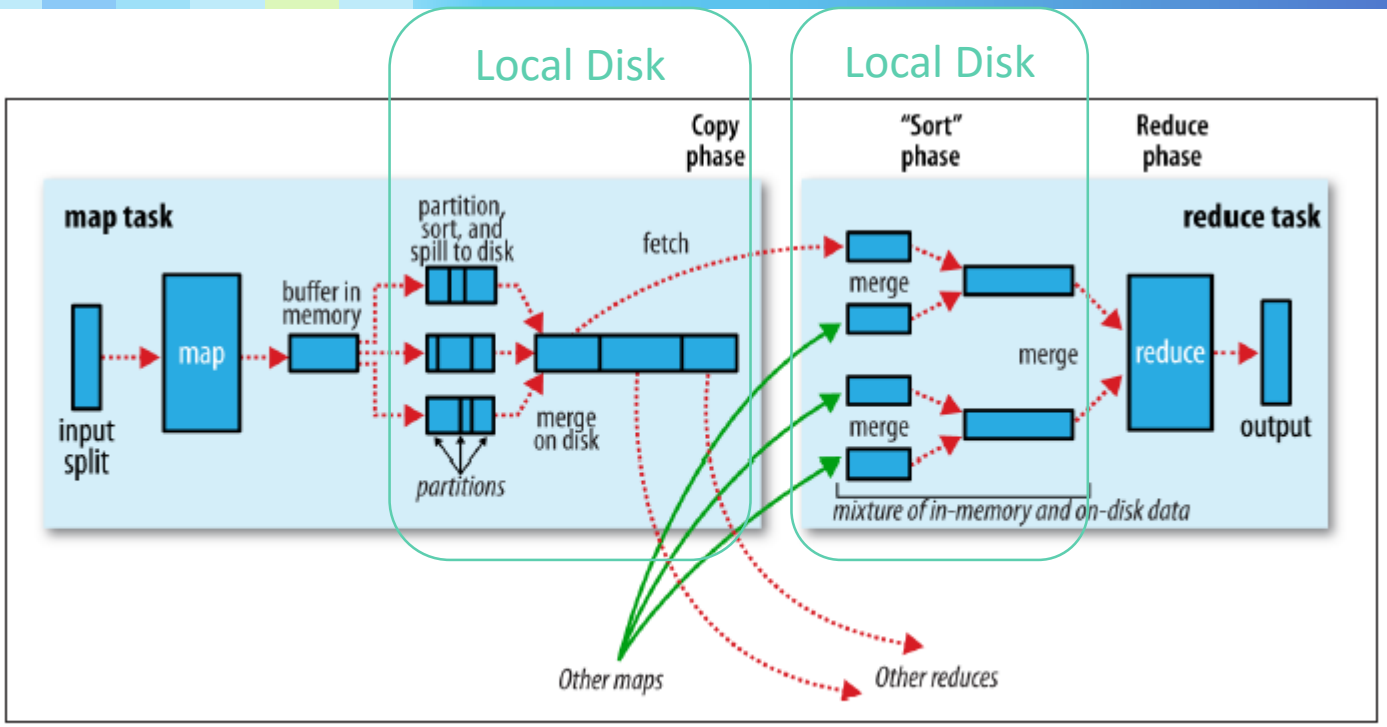


Figure 7-4. Shuffle and sort in MapReduce

- Spills when memory buffer full
- Divides the data into partitions for each reducer
- performs an in-memory sort by key, per partition
- Runs combiner if present on sorted output and writes
- If >3 spill files, runs combiner again.
- Outputs merged, partitioned and sorted into single file on disk

Hadoop: The Definitive Guide, 4th Edition, 2015

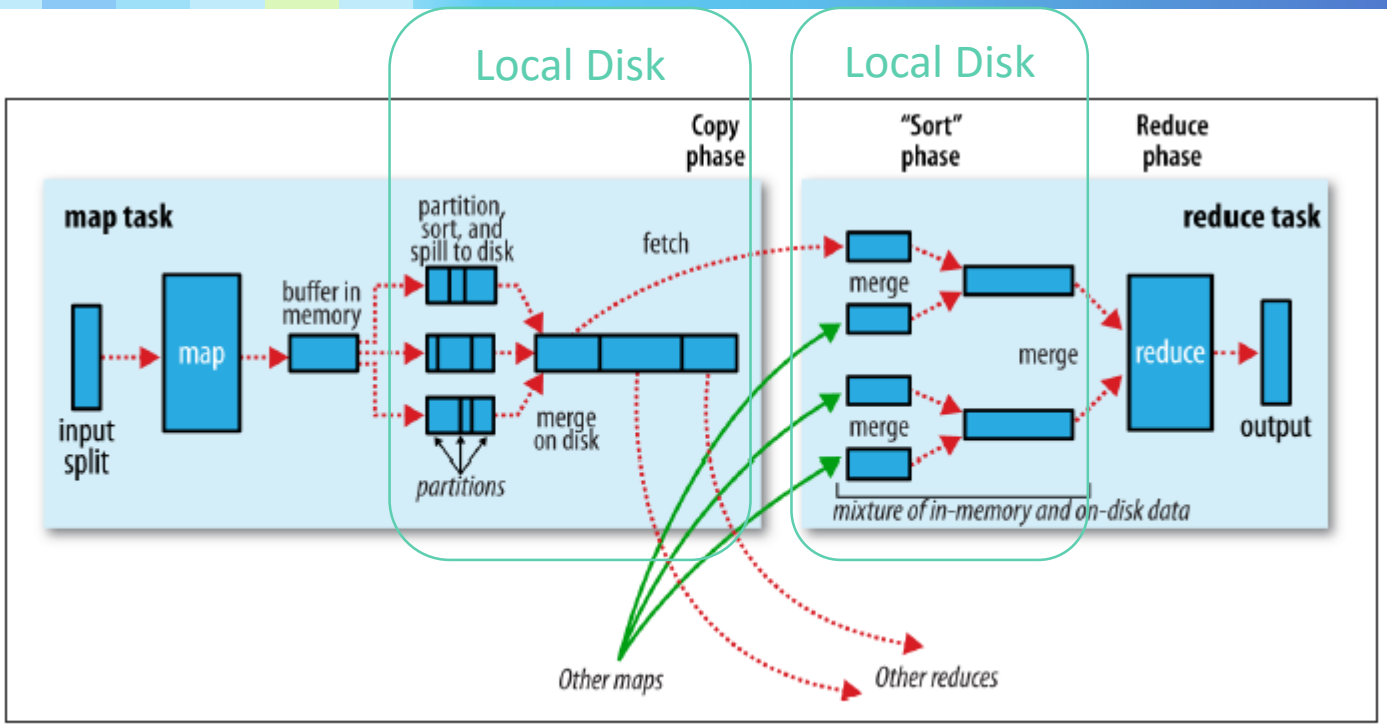


Figure 7-4. Shuffle and sort in MapReduce

- Reducer copies as soon as map output available
  - Copied to reducer memory if small, then merged/spilled to disk on overflow
- Incremental merge sort takes place in background
- Full merge/sort takes place before reduce method called

Fault Tolerance

- Idempotent, “side-effect free”
- Save data to local disk before reduce
- Task crash & recover
- Node crash & recover
- Skipping bad records

Re-execution

- Redundant execution
- Improve Utilization
- Speculative execution
- Locating stragglers

Reading

- Hadoop: The Definitive Guide, **4th Edition**, 2015
  - **Chapters 3, 4, 7**

Additional Resources

- Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop, 2015
  - **Chapters 1, 3, 4, 7**