Big Data
- Hadoop Architecture

Dr. Matt Zhang
ITU

Outline

- Motivation
- Hadoop Cluster
- Components
- Comparison with Traditional MPP Databases
- Features Supported
- Architecture

Limitations of Traditional Large-Scale Computation

- Traditionally, computation is processor-bound
  - Relatively small amount of data
  - Significant amount of complex processing performed on that data
- For decades, the primary push was to increase the computing power of a single machine
  - Faster processor, more RAM

Limitations of Distributed Systems

- Programming for traditional distributed systems is complex
  - Data exchange requires synchronization
  - Limited bandwidth is available
  - Temporal dependencies are complicated
  - It is difficult to deal with partial failures of the system
- Data is stored on a SAN
  - At compute time, data is copied to the compute nodes
  - Acceptable for relatively limited amounts of data

Motivation

- Amount of data is increasing rapidly but the rate of data that can be read from disk drives have not kept up:
  - twenty years ago (1990) drive stored 1.37GB and the IO rate was around 4.4 MB/sec (read the drive in 5 minutes).
  - Today drive is around 1TB and IO rate is around 100 MB/sec (read the drive in 2.5 hrs)!
- With parallel processing the probability of errors/failures increase

Motivation (cont.)

- Most of the time in the analysis process, we will need to combine data from different disk drives
- MapReduce provides a programming model that abstracts the problem from R/W to disk to computation over sets of Keys and Values (K/V pairs).
  - Move computation to data!

Motivation (cont.)

- Hadoop provides a reliable shared storage and analysis platform, storage is provided by HDFS and analysis by MapReduce.

Requirements

- Partial Failure Support
  - Failure of a component should result in a graceful degradation of application performance
    - Not complete failure of the entire system
- Data Recoverability
  - If a component of the system fails, its workload should be assumed by working units in the system
    - Failure should not result in the loss of any data

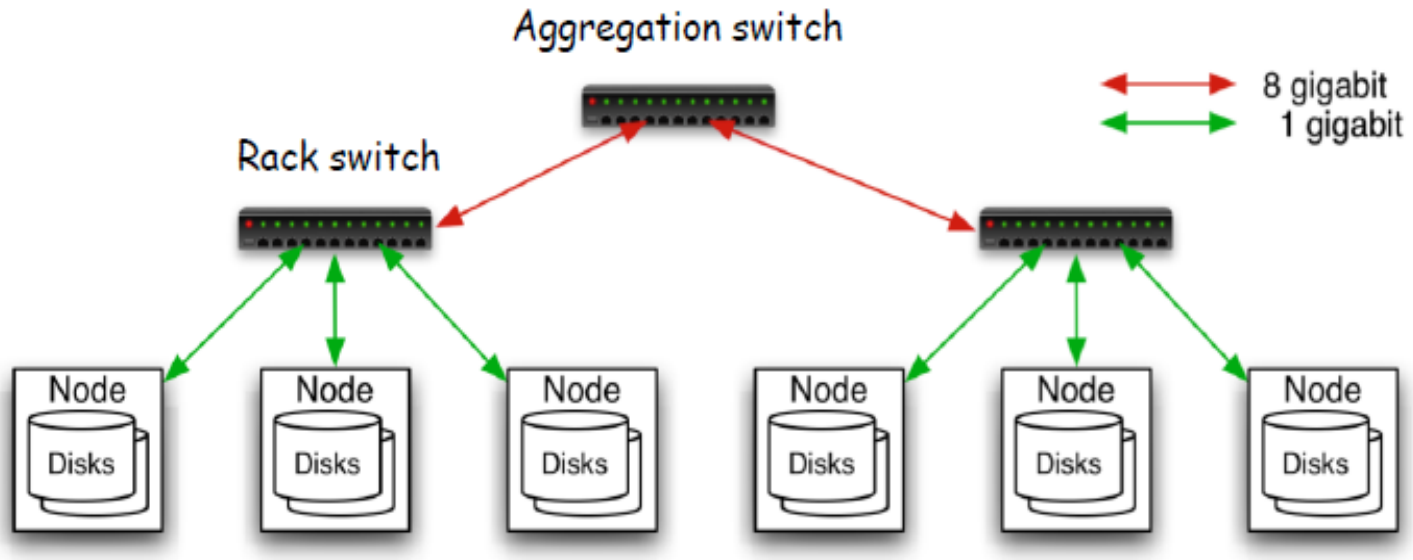
Requirements (cont'd)

- Component Recovery
  - If a component of the system fails and then recovers, it should be able to rejoin the system
    - Without requiring a full restart of the entire system
- Consistency
  - Component failures during execution of a job should not affect the outcome of the job

Requirements(cont'd)

- Scalability
  - Adding load to the system should result in a graceful decline in performance of individual jobs
    - Not failure of the system
- Increasing resources should support a proportional increase in load capacity

Typical Hadoop Cluster



- 40 nodes/rack, 1000-4000 nodes in a cluster
- 1 Gbps bandwidth within rack, 8 Gbps out of rack
- Node: 8 x 2GHZ cores, 8 GB RAM, 4 disks (4 TB)

Hadoop Components

HDFS – Distributed File System:

- Combines cluster's local storage into a single name space
- All data is replicated X3 (default) to multiple machines
- Provides locality information to clients

MapReduce:

- Batch computation framework
- Tasks re-executed on failure
- Optimizes for data locality of input

Comparison with Traditional MPP Databases

Opeartion	MPP Databases	MapReduce
Data size	GB - TB	PetaBytes
Access	Interactive and Batch	Batch
Update	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Structure Enforcement	Schema on write	Schema on read (Hive)
Integrity	High	Low
Scaling	Nonlinear	Linear

Features Supported by Different Hadoop Releases

Feature	1.x	0.22	2.x
Secure authentication	Yes	No	Yes
Old configiration names	Yes	Deprecated	Deprecated
New configuration names	No	Yes	Yes
Old M/R API	Yes	Yes	Yes
New M/R API	Yes (with some missing libs	Yes	Yes
M/R 1 runtime (Classic)	Yes	Yes	No
M/R 2 runtime (YARN)	No	No	Yes
HDFS federation	No	No	Yes
HDFS High-availability	No	No	Yes

Configuration Changes

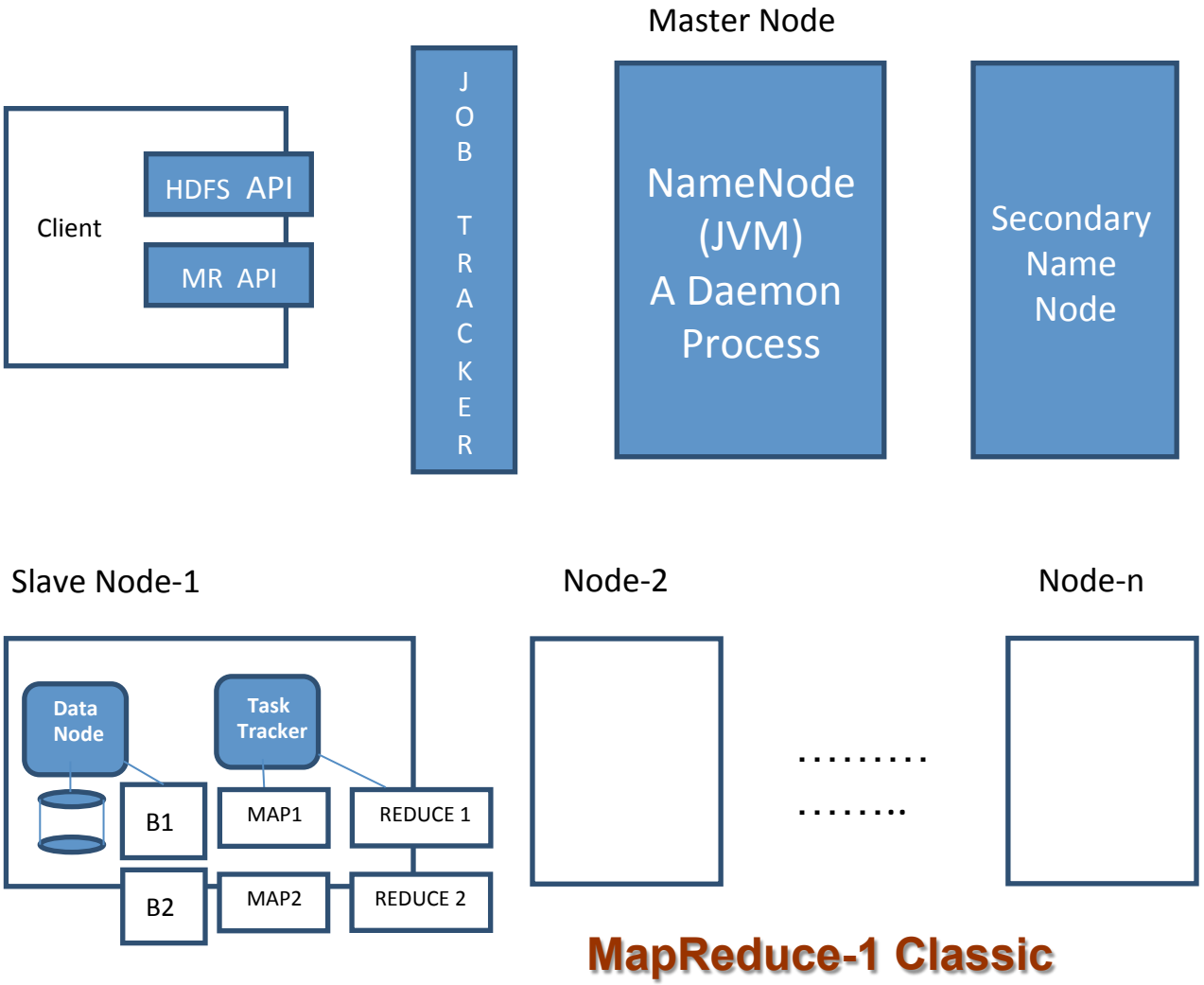
- Configuration property names have changed after 1.x release:
  - dfs.namenode prefix: dfs.name.dir ➔ dfs.namenode.name.dir
  - mapreduce prefix: mapred.job.name ➔ mapreduce.job.name
- MapReduce APIs:
  - there are two Java MapReduce APIs: Classic (API v1) and YARN (API v2)

Configuration Changes

- Compatibility
  - When moving from one release to another , you need to consider the upgrade steps needed.
  - There are several aspects of compatibility
    - API compatibility (between user code and published Hadoop APIs),
    - data compatibility (related to persistent data and metadata format), and
    - wire compatibility (protocol between client and server (RPC and HTTP)

Hadoop Architecture Overview

- **NameNode**: keep the mapping of HDFS file to blocks where each block (64MB) is mapped to a physical file at the OS level.
- **Each file** has default 3 copies
- **NameNode** knows proximity of client to a given replica of a given block in a file
- **Job Tracker** function as resource manager + manage an application life cycle + manages cluster resources
- **Task Tracker** + Per-node agent + Manage tasks



NameNode

- Maintains metadata (dfs.name.dir property)
- Consists of FsImage (initial copy of the metadata from last checkpoint) in memory, and
- EditLog (record every change to the metadata) on disk.

NameNode

- On checkpoint or reboot (which forces checkpoint), the name node flush metadata from memory to disk "FsImage" + merge the EditLog with the "fsimage."
- On reboot the primary node, the NameNode will start with the merged metadata (FsImage) and with clean EditLog.
- Problem: if you do not do checkpoint or reboot for a while and the NameNode dies, you will lose data.

Secondary NameNode

- Secondary NameNode is not HA solution
- Rather it takes periodically (every 1 hr) the responsibility to merge edit log with the latest “fsimage” file and re-upload the new image to the primary name node.
- On primary node reboot it will have latest fsimage file.

Secondary NameNode

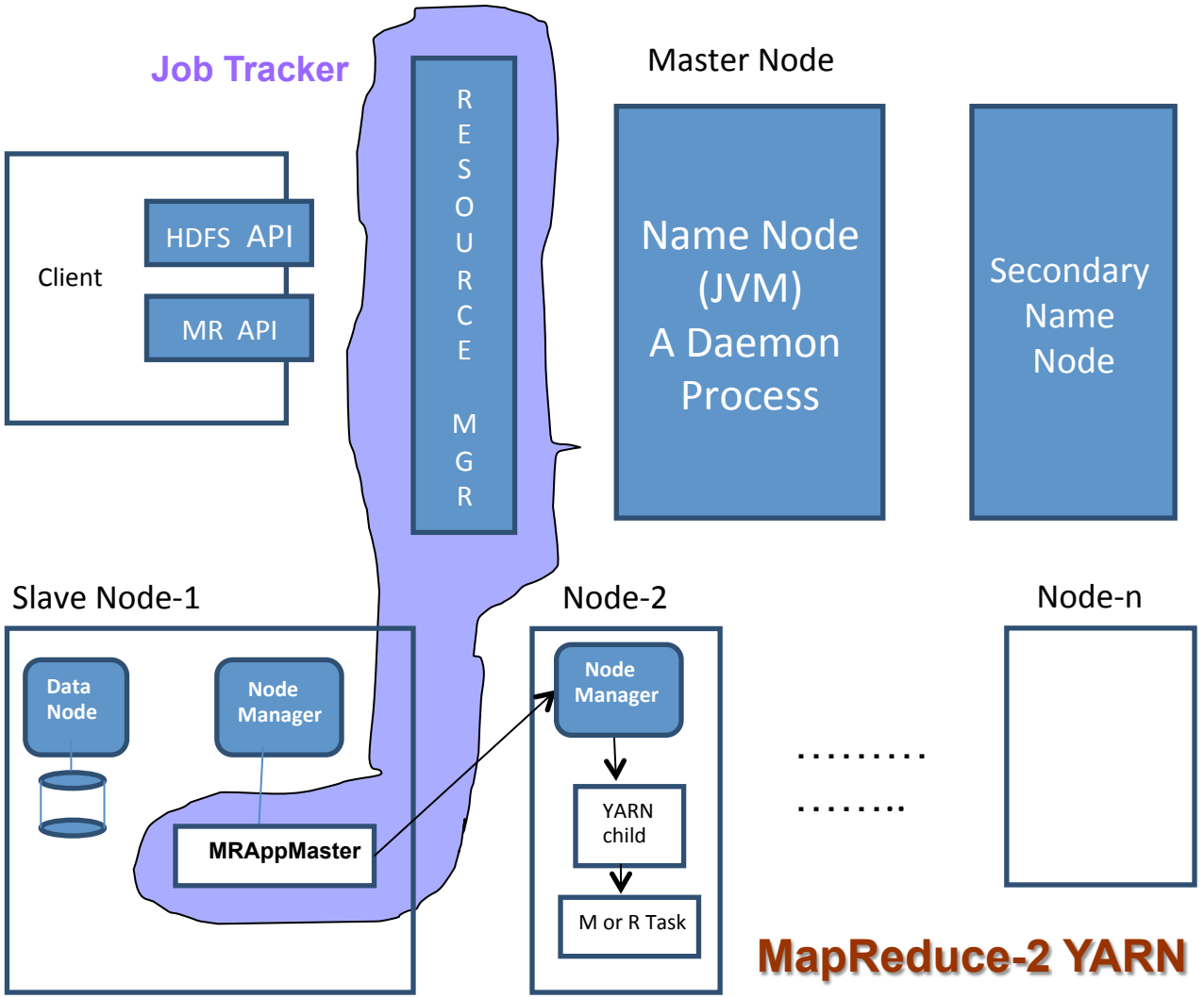
- Namenode does not communicate with any other nodes; instead data node, secondary node, and clients will communicate with the primary NameNode.
- Data nodes send hearbeat every 3 seconds to the NameNode.
- If NameNode do not receive heartbeat from a specific data node in 10 minutes it is assumed down and start creating replacement data node.

Slave Node(DataNode)

- Actual contents of the files are stored as blocks on the slave nodes
- Blocks are simply files on the slave nodes' underlying filesystem
  - Named blk_xxxxxxx
  - Nothing on the slave node provides information about what underlying file the block is a part of
    - That information is only stored in the NameNode's metadata
- Each block is stored on multiple different nodes for redundancy
  - Default is three replicas
- Each slave node runs a DataNode daemon
  - Controls access to the blocks
  - Communicates with the NameNode

Hadoop Architecture Overview

- **YARN** (Yet Another Resource Negotiator): is intended to support multiple app paradigms in addition to M/R.
- **Job Tracker** is being replaced with: Resource Mgr (manage global resources) + MRAppMaster (manage the Application life cycle).



Hadoop Scheduling

- Default Hadoop Scheduler:
  - FIFO queueing of MR jobs
  - Each job gets entire cluster
  - Hadoop will select # of mappers, # of reducers, or you can select (over 100 parameters!)

Hadoop Scheduling

- Fair scheduler / Capacity scheduler:
  - Jobs can run concurrently
  - Provide short response times to small jobs
  - Improve utilization and throughput

More on Fair Scheduler Basics

- Group jobs into “pools”:
  - Guaranteed minimum map slots (# of mappers)
  - Guaranteed minimum reduce slots (# of reducers)
  - Shared across jobs
- Limits on # of running jobs:
  - Per user
  - Per pool

Summary

- We covered an overview on Hadoop
- We also discussed the main components in Hadoop cluster
- We explained the architecture and features with Hadoop technologies