Administrator Training for Apache Hadoop

The Case for Apache Hadoop

The Data Deluge (1)

- **We are generating more data than ever**
  - Financial transactions
  - Sensor networks
  - Server logs
  - Analytics
  - e-mail and text messages
  - Social media

The Data Deluge (2)

- **And we are generating data faster than ever**
  - Automation
  - Ubiquitous internet connectivity
  - User-generated content
- **For example, every day**
  - Twitter processes 500 million messages
  - Amazon S3 storage adds more than one billion objects
  - Facebook users generate 4.5 billion comments and “Likes”

Data is Value

- **This data has many valuable applications**
  - Marketing analysis
  - Product recommendations
  - Demand forecasting
  - Fraud detection
  - And many, many more…
- **We must process it to extract that value**

Data Processing Scalability

- **How can we process all that information?**
- **There are actually two problems**
  - Large-scale data storage
  - Large-scale data analysis

Disk Capacity and Price

- **We are generating more data than ever before**
- **Fortunately, the size and cost of storage has kept pace**
  - Capacity has increased while price has decreased

Year	Capacity (GB)	Cost per GB (USD)
1997	2.1	$157
2004	200	$1.05
2015	3,000	$0.029

Disk Capacity and Performance

- **Disk performance has also increased in the last 15 years**
- **Unfortunately, transfer rates have not kept pace with capacity**

Year	Capacity (GB)	Transfer Rate (MB/s)	Disk Read Time
1997	2.1	16.6	126 seconds
2004	200	56.5	59 minutes
2015	3,000	210	3 hours, 58 minutes

Data Access is the Bottleneck

- **Although we can process data more quickly, *accessing* it is slow**
  - This is true for both reads and writes
- **For example, reading a single 3TB disk takes almost four hours**
  - We cannot process the data until we have read it
  - We are limited by the speed of a single disk
- **We will see Hadoop's solution in a few moments**
  - But first we will examine how we *process* large amounts of data

Monolithic Computing

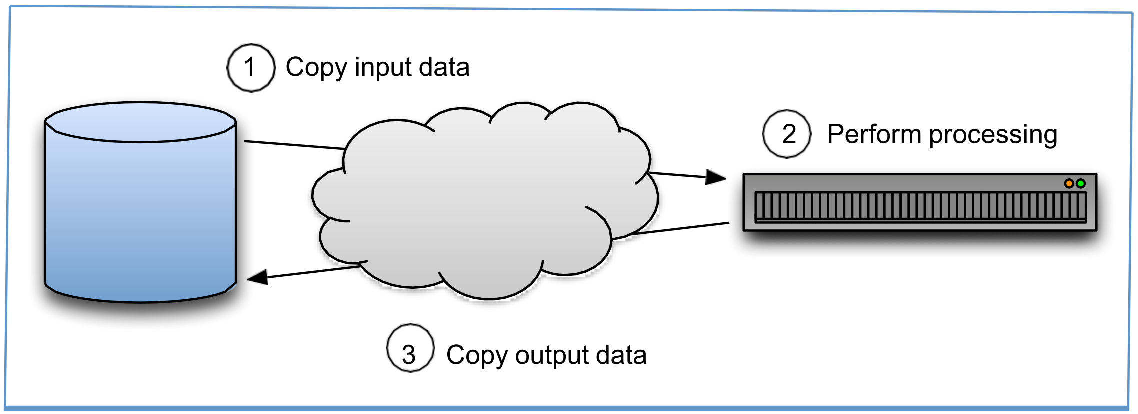
- **Traditionally, computation has been processor-bound**
  - Intense processing on small amounts of data
- **For decades, the goal was a bigger, more powerful machine**
  - Faster processor, more RAM
- **This approach has limitations**
  - High cost
  - Limited scalability

Distributed Computing

- **Modern large-scale processing is distributed across machines**
  - Often hundreds or thousands of nodes
  - Common frameworks include MPI, PVM and Condor
- **Focuses on distributing the processing workload**
  - Powerful compute nodes
  - Separate systems for data storage
  - Fast network connections to connect them

Distributed Computing Processing Pattern

- **Typical processing pattern**
  - Step 1: Copy input data from storage to compute node
  - Step 2: Perform necessary processing
  - Step 3: Copy output data back to storage



- **This works fine with relatively small amounts of data**
  - That is, where step 2 dominates overall runtime

Data Processing Bottleneck

- **That pattern does not scale with large amounts of data**
  - More time spent copying data than actually processing it
  - Getting data to the processors is the bottleneck
- **Grows worse as more compute nodes are added**
  - They are competing for the same bandwidth
  - Compute nodes become starved for data

Complexity of Distributed Computing

- **Distributed systems pay for scalability by adding complexity**
- **Much of this complexity involves**
  - Availability
  - Data consistency
  - Event synchronization
  - Bandwidth limitations
  - Partial failure
  - Cascading failures
- **These are often more difficult than the original problem**
  - Error handling often accounts for the majority of the code

System Requirements: Failure Handling

- **Failure is inevitable**
  - We should strive to handle it well
- **An ideal solution should have (at least) these properties**

Failure-Handling Properties of an Ideal Distributed System	
Automatic	Job can still complete without manual intervention
Transparent	Tasks assigned to a failed component are picked up by others
Graceful	Failure results only in a proportional loss of load capacity
Recoverable	That capacity is reclaimed when the component is later replaced
Consistent	Failure does not produce corruption or invalid results

More System Requirements

- **Linear horizontal scalability**
  - Adding new nodes should add proportional load capacity
  - Avoid contention by using a “shared nothing” architecture
  - Must be able to expand cluster at a reasonable cost
- **Jobs run in relative isolation**
  - Results must be independent of other jobs running concurrently
  - Although performance can be affected by other jobs
- **Simple programming model**
  - Should support a widely-used language
  - The API must be relatively easy to learn
- **Hadoop addresses these requirements**

Hadoop: A Radical Solution

- **Traditional distributed computing frequently involves**
  - Complex programming requiring explicit synchronization
  - Expensive, specialized fault-tolerant hardware
  - High-performance storage systems with built-in redundancy
- **Hadoop takes a radically different approach**
  - Inspired by Google’s GFS and MapReduce architecture
  - This new approach addresses the problems described earlier

Hadoop Scalability

- **Hadoop aims for linear horizontal scalability**
  - Cross-communication among nodes is minimal
  - Just add nodes to increase cluster capacity and performance
- **Clusters are built from industry-standard hardware**
  - Widely-available and relatively inexpensive servers
  - You can “scale out” later when the need arises

Solution: Data Access Bottleneck

- **Recap: separate storage and compute systems create bottleneck**
  - Can spend more time copying data than processing it
- **Solution: store and process data on the same machines**
  - This is why adding nodes increases capacity and performance
- **Optimization: Use intelligent job scheduling (data locality)**
  - Hadoop tries to process data on the same machine that stores it
  - This improves performance and conserves bandwidth
  - "Bring the computation to the data"

Solution: Disk Performance Bottleneck

- **Recap: a single disk has great capacity but poor performance**
- **Solution: use multiple disks in parallel**
  - The transfer rate of one disk might be 210 *megabytes*/second
    - Almost four hours to read 3 TB of data
  - 1000 such disks in parallel can transfer 210 *gigabytes*/second
    - Less than 15 seconds to read 3TB of data
- **Colocated storage and processing makes this solution feasible**
  - 100-node cluster with 10 disks per node = 1000 disks

Solution: Complex Processing Code

- **Recap: Distributed programming is very difficult**
  - Often done in C or FORTRAN using complex libraries
- **Solution: Use a popular language and a high-level API**
  - MapReduce code is typically written in Java (like Hadoop itself)
    - It is possible to write MapReduce in nearly any language
- **The MapReduce programming model simplifies processing**
  - Deal with one record (key-value pair) at a time
  - Complex details are abstracted away
    - No file I/O
    - No networking code
    - No synchronization

Solution: Fault Tolerance

- **Recap: Distributed systems often use expensive components**
  - In order to minimize the *possibility* of failure
- **Solution: Realize that failure is inevitable**
  - And instead try to minimize the *effect* of failure
  - Hadoop satisfies all the requirements we discussed earlier
- **Machine failure is a regular occurrence**
  - A server might have a mean time between failures (MTBF) of 5 years (~1825 days)
  - Equates about one failure per day in a 2,000 node cluster

Core Hadoop Components

- **Hadoop is a system for large-scale data processing**
- **Hadoop provides**
  - HDFS for data storage
  - The extensible YARN framework
    - For application scheduling and resource management
    - Includes MapReduce version 2 for data processing
- **Plus the infrastructure needed to make them work, including**
  - Filesystem utilities
  - Application scheduling and monitoring
  - Web UI

The Hadoop Ecosystem

- **Many related tools integrate with Hadoop**
  - Data processing: Spark
  - Data analysis: Hive, Pig, and Impala
  - Data discovery: Solr (Cloudera Search)
  - Machine learning: MLlib, Mahout, and others
  - Data ingestion: Sqoop, Flume, Kapa
  - Coordination: ZooKeeper
  - User experience: Hue
  - Workflow management: Oozie
  - Cluster management: Cloudera Manager
- **These are not considered "core Hadoop"**
  - Rather, they are part of the "Hadoop ecosystem"
  - Many are also open source Apache projects
  - We will learn about several of these later in the course

Hadoop Cluster Installation

Hadoop Cluster Overview

- **Hadoop daemons run on a cluster of machines**
- **The Hadoop Distributed File System (HDFS) is used to distribute data amongst the nodes**
- **Computational frameworks such as MapReduce, Spark, and Impala bring the computing to the data**
- **To realize the benefits of Hadoop you must deploy Hadoop daemons across a sizable cluster of machines**
  - Many organizations maintain multiple clusters, each with hundreds or thousands of nodes

Deploying on Multiple Machines

- **When commissioning multiple machines, use an automated operating system deployment tool**
  - Red Hat's Kickstart
  - Debian Fully Automatic Installation
  - Dell Crowbar
  - …
- **You might optionally use a tool to manage the underlying operating system**
  - Puppet
  - Chef
  - …
- **Use Cloudera Manager to install Hadoop and manage the Hadoop cluster**

Hadoop Requirements (1)

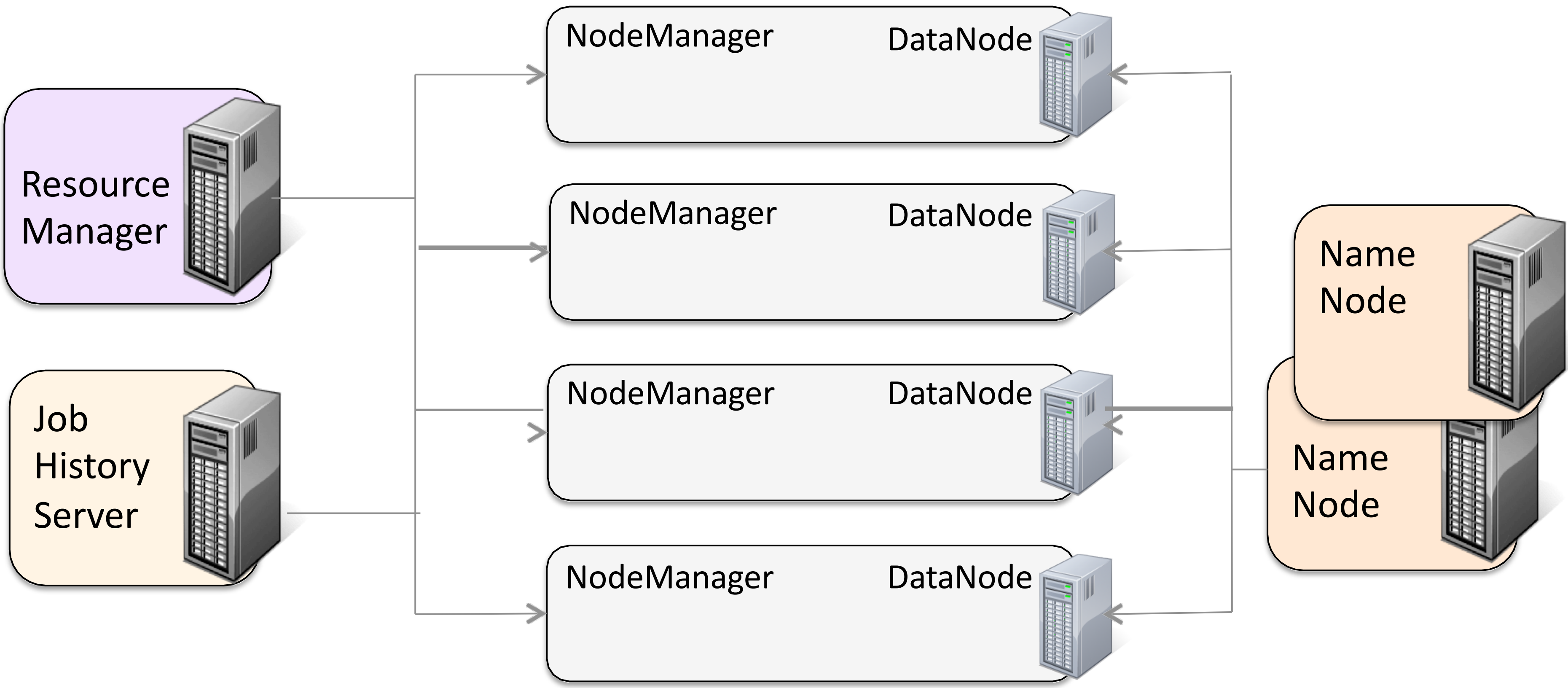
- **Supported Operating Systems (all 64-bit)**
  - Red Hat Enterprise Linux/Centos 5.7, 6.4, 6.5, 6.6
  - Oracle Enterprise Linux 5.6, 6.4, 6.5, 6.6
  - SUSE Linux Enterprise Server 11 Service Pack 2 or later
  - Debian 7.0, 7.1
  - Ubuntu 12.04, 14.04
- **Supported Browsers**
  - Internet Explorer 9 or later
  - Google Chrome
  - Safari 5 or later
  - Firefox 24 or 31

Hadoop Requirements (2)

- **Supported JDKs**
  - Oracle JDK 1.7.0_55, 1.7.0_67 or higher, 1.8.0_40 or higher
- **Supported databases**
  - MySQL 5.5 and 5.6
  - Oracle 11g Release 2
  - PostgreSQL 8.4, 9.2, and 9.3

Basic Hadoop Cluster Installation: HDFS and YARN (MR2 Included)

- **YARN – Resource Manager, Job History Server, and many NodeManagers**
- **HDFS – Name Node(s) and many DataNodes**



Distribute the Daemons

- **Not all daemons run on each machine**
  - NameNode, ResourceManager, JobHistoryServer (‘master’ daemons)
    - One per cluster, unless running in an HA configuration
  - Secondary NameNode
    - One per cluster in a non-HA environment
  - DataNodes, NodeManagers
    - On each data node in the cluster
  - Exception: for small clusters (less than 10 - 20 nodes), it is acceptable for more than one of the master daemons to run on the same physical node

Lab : Launching Cluster HDFS