

Course Guide Volume 2

IBM BigInsights Foundation v4.0

Course code DW613 ERC 1.0



This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

IBM Training

June 2015 edition**NOTICES**

This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America*

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

TRADEMARKS

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

© Copyright International Business Machines Corporation 2015.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Contents

IBM OPEN PLATFORM (IOP) WITH APACHE HADOOP	II
Unit 6 Introduction to IBM Open Platform with Apache Hadoop.....	6-1
Unit objectives	6-3
Agenda.....	6-4
ODP: Open Data Platform initiative summary.....	6-5
The Open Data Platform goals	6-6
The Hadoop marked is evolving and favors IBM's strengths	6-7
Use data to know your business and make better decisions.....	6-8
Three things to know about IBM and IBM BigInsights v4	6-10
Enabling personas with capabilities	6-11
New capabilities in IBM BigInsights v4	6-12
Package structure	6-13
IBM BigInsights for Apache Hadoop offering suite.....	6-15
Pricing and licensing	6-16
Components	6-17
Maintain open source currency for interoperability	6-18
IBM Open Platform v4.0 with Apache Hadoop.....	6-20
Open source currency	6-21
Components of the Open Data Platform.....	6-22
Unit summary	6-23
Exercise 1: Additional exploration of the lab environment.....	6-24
Unit 7 Apache Ambari.....	7-1
Unit objectives	7-3
Agenda.....	7-4
Functionality of Apache Ambari.....	7-5
Ambari architecture	7-6
Sign in to the Ambari web interface	7-9
Navigating the Ambari web interface	7-10
The Ambari web dashboard	7-11
Metric details on the Ambari dashboard	7-12
Metric details for time-based cluster components.....	7-13
Service Actions/Alert and Health Checks	7-14
Add Service Wizard	7-15
Background service check from the admin menu	7-16
Host metrics: example of one host (pseudo-distributed)	7-17
Non-functioning/failed services: example of HBase	7-18
Managing Ambari	7-19

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Creating users and groups	7-20
Standard Ambari Service users and groups	7-21
Managing hosts in a cluster.....	7-23
Running Ambari from the command line.....	7-24
Ambari terminology	7-26
Unit summary	7-28
Exercise 1: Managing Hadoop clusters with Apache Ambari	7-29
Unit 8 Hadoop and HDFS.....	8-1
Unit objectives	8-3
Agenda.....	8-4
The importance of Hadoop	8-5
Hardware improvements through the years.....	8-6
What hardware is not used for Hadoop	8-8
Parallel data processing is the answer	8-9
What is Hadoop?	8-10
Hadoop open source projects.....	8-12
Advantages and disadvantages of Hadoop	8-14
Time line for Hadoop	8-15
Hadoop: major components	8-17
Brief introduction to HDFS and MapReduce.....	8-18
Hadoop Distributed File System (HDFS) principles	8-19
HDFS architecture.....	8-20
HDFS blocks	8-22
HDFS replication of blocks	8-24
Setting rack network topology (Rack Awareness).....	8-26
Compression of files.....	8-29
Which compression format should I use?	8-31
NameNode startup	8-32
NameNode files (as stored in HDFS).....	8-33
Adding a file to HDFS: replication pipelining	8-34
Managing the cluster	8-37
HDFS-2 NameNode HA (High Availability)	8-38
Secondary NameNode	8-39
Possible FileSystem setup approaches	8-40
Federated NameNode (HDFS2)	8-41
fs - file system shell	8-43
Unit summary	8-48
Exercise 1: File access and basic commands with HDFS	8-49

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit 9 MapReduce and YARN	9-1
Unit objectives	9-3
Topic: Introduction to MapReduce processing based on MR1	9-4
Agenda.....	9-5
MapReduce: the Distributed File System.....	9-6
MapReduce v1 explained	9-7
MapReduce v1 engine.....	9-8
The MapReduce programming model	9-9
The MapReduce execution environments	9-10
Agenda.....	9-11
MapReduce 1 overview	9-12
MapReduce: Map Phase	9-13
MapReduce: Shuffle Phase.....	9-14
MapReduce: Reduce Phase.....	9-15
MapReduce: Combiner (Optional)	9-16
Agenda.....	9-17
WordCount example.....	9-18
Map Task	9-19
Shuffle	9-21
Reduce.....	9-22
Optional: Combiner	9-23
Source code for WordCount.java.....	9-24
How does Hadoop run MapReduce jobs?	9-28
Classes	9-30
Splits	9-31
RecordReader	9-32
InputFormat.....	9-33
Fault tolerance.....	9-34
Topic: Issues with and limitations of Hadoop v1 and MapReduce v1	9-36
Issues with the original MapReduce paradigm	9-37
Limitations of classic MapReduce (MRv1)	9-38
Scalability in MRv1: Busy JobTracker.....	9-39
YARN overhauls MRv1.....	9-40
Hadoop 1 and 2 architectures compared.....	9-42

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

YARN features	9-43
YARN features: scalability	9-44
YARN features: multi-tenancy	9-45
YARN features: compatibility	9-47
YARN features: higher cluster utilization	9-48
YARN features: reliability and availability	9-49
YARN major features summarized	9-50
Topic: The architecture of YARN	9-51
Hadoop v1 to Hadoop v2	9-52
Architecture of MRv1	9-53
YARN architecture	9-54
Terminology changes from MRv1 to YARN	9-56
YARN in BigInsights	9-57
YARN high level architecture	9-58
Running an application in YARN	9-59
How YARN runs an application	9-66
Container JAVA command line	9-67
Provisioning, management, and monitoring	9-68
Spark in Hadoop 2+	9-69
Unit summary	9-70
Exercise 1: Running MapReduce and YARN jobs	9-71
Exercise 2: Creating and coding a simple MapReduce job	9-82
Unit 10 Apache Spark	10-1
Unit objectives	10-3
Big data and Spark	10-4
Ease of use	10-6
Who uses Spark and why?	10-7
Spark unified stack	10-9
Brief history of Spark	10-11
Resilient Distributed Datasets (RDD)	10-12
Spark jobs and shell	10-14
Brief overview of Scala	10-15
Scala: anonymous functions, aka Lambda functions	10-17
Computing wordcount using Lambda functions	10-18
Resilient Distributed Dataset (RDD)	10-19
Creating an RDD	10-21
Spark's Scala and Python shells	10-23
RDD basic operations	10-24
What happens when an action is executed?	10-26

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

RDD operations: transformations	10-34
RDD operations: actions.....	10-36
RDD persistence	10-37
Best practices for which storage level to choose	10-39
Shared variables and key-value pairs.....	10-41
Programming with key-value pairs.....	10-43
Programming with Spark	10-44
SparkContext	10-45
Linking with Spark: Scala	10-46
Initializing Spark: Scala	10-47
Linking with Spark: Python	10-48
Initializing Spark: Python	10-49
Linking with Spark: Java.....	10-50
Initializing Spark: Java.....	10-51
Passing functions to Spark	10-52
Programming the business logic.....	10-54
Running Spark examples	10-55
Create Spark standalone applications: Scala	10-57
Run standalone applications	10-58
Spark libraries	10-59
Spark SQL.....	10-60
Spark streaming	10-61
Spark Streaming - Internals	10-62
MLib	10-64
GraphX.....	10-65
Spark cluster overview	10-66
Spark monitoring	10-68
Unit summary	10-70
Exercise 1: Working with a Spark RDD with Scala	10-71
Unit 11 Coordination, Management, and Governance.....	11-1
Unit objectives	11-3
Topic: ZooKeeper.....	11-4
ZooKeeper	11-5
Distributed systems	11-7
What is ZooKeeper?.....	11-8
ZooKeeper service: replicated mode	11-10
ZooKeeper service: Standalone mode.....	11-12
Consistency guarantees	11-13
What ZooKeeper does not guarantee.....	11-14
ZooKeeper structure: data model	11-15

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

ZNode operations	11-16
ZNode watches	11-17
ZNode reads and writes	11-18
ZooKeeper generic use cases	11-19
ZooKeeper's role in the Hadoop infrastructure	11-20
Real world use cases for ZooKeeper.....	11-21
Topic 2: Slider	11-22
Slider	11-23
Apache Slider: enable long running services on YARN	11-25
Role of Slider in the Hadoop ecosystem.....	11-26
Topic: Knox	11-28
Knox.....	11-29
Knox security.....	11-31
Apache Knox is just one ring of an overall security system.....	11-32
Unit summary	11-33
Exercise 1: Explore Zookeeper.....	11-34

Unit 12 Data Movement..... 12-1

Unit objectives	12-3
Load scenarios	12-4
Data at rest.....	12-5
Data in motion	12-6
Load solution using Flume.....	12-7
Solution if data is from a data warehouse or RDBMS	12-8
Data from a web server	12-9
Topic: Sqoop	12-10
Sqoop.....	12-11
Sqoop connection.....	12-12
Sqoop import.....	12-13
Sqoop import examples.....	12-14
Sqoop exports	12-15
Sqoop export examples.....	12-16
Additional export information	12-17
Topic: Flume	12-18
Working with Flume	12-19
Flume	12-20
How Flume works.....	12-21
Consolidation	12-24
Replicating and multiplexing	12-25
Configuration considerations of Flume	12-26
Configuring Flume	12-27

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Configuration example.....	12-28
Flume sources.....	12-29
Interceptors	12-31
Flume sinks	12-32
Flume channels	12-33
Flume channel selectors.....	12-34
Configuration details: components	12-35
Configuration details: properties	12-36
Configuration details: bindings.....	12-37
Flume example configuration file	12-38
Working with an agent.....	12-39
Unit summary	12-40
Exercise 1: Moving data into HDFS with Sqoop	12-41
Unit 13 Storing and Accessing Data.....	13-1
Unit objectives	13-3
Introduction to data.....	13-4
Gathering and cleaning/munging/wrangling data.....	13-5
Flat files/text files.....	13-7
CSV and various forms of delimited files	13-8
Avro/SequenceFile	13-9
JSON format - Java Script Object Notation.....	13-11
XML(eXtensible Markup Language)	13-14
RC/ORD file formats.....	13-15
Parquet.....	13-16
NoSQL	13-18
Origins of the NoSQL products.....	13-19
HBase	13-20
What is HBase.....	13-21
Why NoSQL?	13-22
Why HBase in particular?	13-24
HBase and ACID properties	13-26
HBase data model.....	13-28
Think "Map": this is not a spreadsheet model	13-29
HBase Data Model: logical view	13-30
Column family.....	13-31
HBase vs traditional RDBMS.....	13-32
Example of a classic RDBMS table	13-33
Example of HBase logical view ("records")	13-34
Example of the physical view ("cell")	13-35
More on the HBase data model.....	13-36

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Indexes in HBase	13-37
Recap of the Hadoop v1 processing environment	13-38
Processing environment of Hadoop v2	13-39
Open source programming languages: Pig and Hive.....	13-40
Pig.....	13-41
Pig vs. SQL	13-42
Characteristics of the Pig language	13-43
Example of a Pig script.....	13-44
What is Hive?	13-45
SQL for Hadoop	13-47
Java vs. Hive: the wordcount algorithm	13-48
Hive and wordcount.....	13-49
Hive components.....	13-50
Starting Hive: the Hive shell	13-51
Data types and models.....	13-52
Data model partitions.....	13-53
Data model external table.....	13-54
Physical layout	13-55
Creating a table	13-56
Hive and HBase	13-57
HBase table mapping	13-58
Languages used by data scientists: R and Python	13-59
Quick overview of R	13-60
R clients	13-61
Simple example.....	13-62
R is noted for its ability to produce graphical output.....	13-63
Quick overview of Python	13-64
Python wordcount program	13-65
Unit summary	13-66
Exercise 1: Using Hive to access Hadoop/HBase data.....	13-67

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit 14 Advanced topics	14-1
Unit objectives	14-3
Topic: Oozie	14-4
Introduction to Oozie	14-5
Oozie Workflow Scheduler for Hadoop.....	14-6
Elements in a Workflow XML.....	14-7
Example of Oozie workflow XML	14-8
How to run an example Oozie application	14-9
Topic: Apache Solr	14-10
Rationale for searching.....	14-11
Solr and other available search technologies	14-12
Apache Solr.....	14-13
Solr usage	14-14
Search procedure	14-15
Overall Solr/Lucene architecture	14-16
Solr web console	14-17
Unit summary	14-18

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

IBM OPEN PLATFORM (IOP) WITH APACHE HADOOP

IBM Training



IBM Open Platform (IOP) with Apache Hadoop

- IBM Open Platform with Apache Hadoop
- Apache Ambari
- Hadoop Distributed File System
- MapReduce and Yarn
- Apache Spark
- Coordination Management and Governance
- Data Movement
- Storing and Accessing Data
- Advanced Topics

© Copyright IBM Corporation 2015

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit 6 Introduction to IBM Open Platform with Apache Hadoop

The slide features a blue header bar with 'IBM Training' on the left and the 'IBM' logo on the right. The main content area has a light gray background with a subtle diagonal striped pattern. The title 'Introduction to IBM Open Platform with Apache Hadoop' is centered in large, bold, dark blue font. Below it, the text 'IBM BigInsights v4.0' is displayed in a smaller, regular dark blue font. At the bottom of the slide, a copyright notice reads: '© Copyright IBM Corporation 2015' and 'Course materials may not be reproduced in whole or in part without the written permission of IBM.'

**Introduction to IBM Open
Platform with Apache Hadoop**

IBM BigInsights v4.0

© Copyright IBM Corporation 2015
Course materials may not be reproduced in whole or in part without the written permission of IBM.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit objectives

- List the major components of the IBM Open Data Platform
- List the common components of the Apache Hadoop stack
- Describe the functionality of each open source component

Unit objectives

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Agenda

- Open Data Platform initiative
 - membership and goals
- IBM's strategy
- IBM BigInsights packaging
- IBM BigInsights personas and add-ins
- Components of a generic Open Data Platform



The Open Data Platform Initiative (ODP) is a shared industry effort focused on promoting and advancing the state of Apache Hadoop® and big data technologies for the enterprise

ODP: Open Data Platform initiative summary

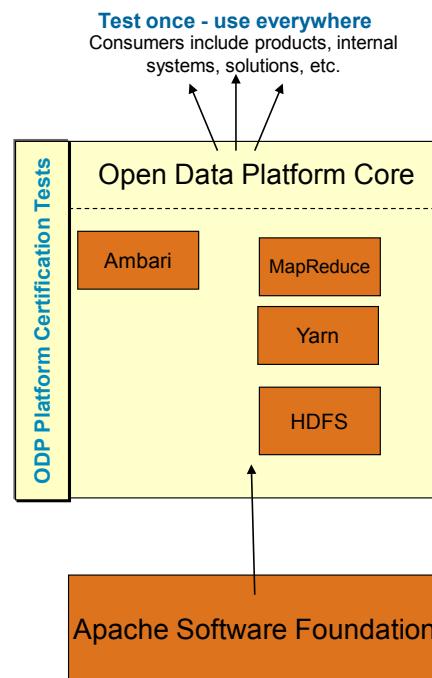


On February 17, 2015, the intention to found the Open Data Platform Initiative was announced and press releases were issued.

By providing the Open Data Platform Core, developers can focus their energies on innovation, instead of on basic integration and testing.

The ODP Association is currently defining the ODP Bylaws, recruiting more members, etc. to prepare for the ODP launch later in 2015.

<http://opendataplatform.org/>
http://opendataplatform.org/OpenDataPlatformInitiative_FAQ.pdf



ODP: Open Data Platform initiative summary

Introduction to IBM Open Platform with Apache Hadoop

© Copyright IBM Corporation 2015

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

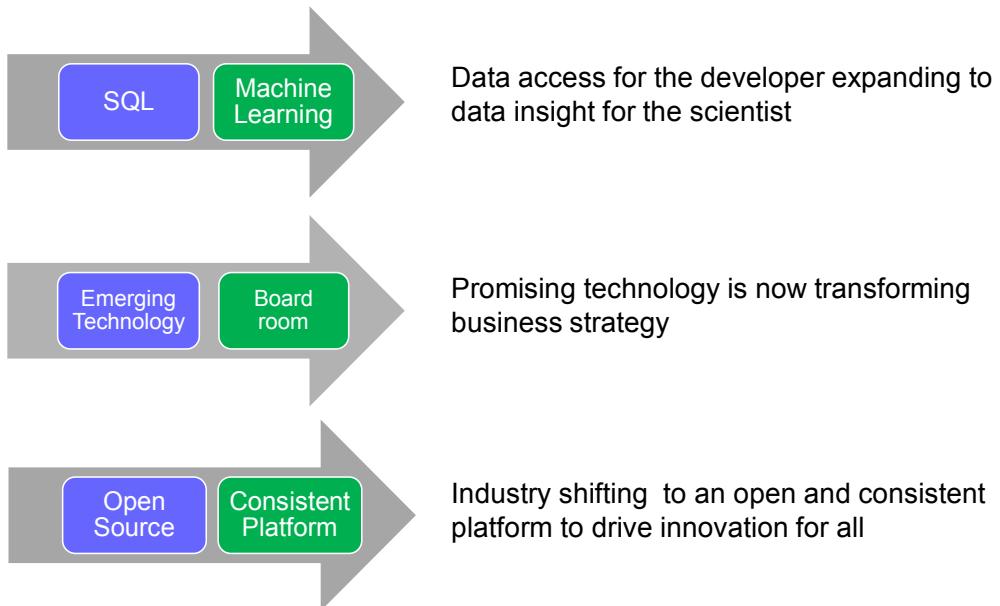
The Open Data Platform goals

1. Accelerate the delivery of big data solutions by providing a well-defined core platform to target
2. Define, integrate, test, and certify a standard "ODP Core" of compatible versions of select big data open source projects
3. Provide a stable base against which big data solutions providers can qualify solutions
4. Produce a set of tools and methods that enable members to create and test differentiated offerings based on the ODP Core
5. Reinforce the role of the Apache Software Foundation (ASF) in the development and governance of upstream projects
6. Contribute to ASF projects in accordance with ASF processes and Intellectual Property guidelines
7. Support community development and outreach activities that accelerate the rollout of modern data architectures that leverage Apache Hadoop
8. Will help minimize the fragmentation and duplication of effort within the industry

The Open Data Platform goals

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

The Hadoop market is evolving and favors IBM's strengths



The Hadoop market is evolving and favors IBM's strengths

Use data to know your business and make better decisions

- Modernize traditional data warehouse environments
- Eliminate fraud and threats by finding hidden patterns
- Better predict customer demands and understand sentiment
- Proactively predict maintenance and quality

Use data to know your business and make better decisions

These are the types of business problems that organizations can leverage Hadoop to address:

- ensure investment protection with a 100% standard Hadoop distribution
- reduce time to value with tools to accelerate application development and deployment
- simplify the environment with productivity tools for business users
- boost performance and efficiency with enhanced implementations of Hadoop components
- ensure seamless interoperability with existing tools and data sources for maximum investment protection

And take actions so that:

- nothing will ever be out of stock because companies will be able to better predict what we want and where we want to buy it
- cars, trucks and equipment will not breakdown as often because predictive maintenance will tell you when and where to get things fixed before they break
- roads will be free from potholes because sensors will know where they are and will tell crews to fix them
- the common flu will not spread because healthcare workers will be able to track outbreaks and treat them on the spot

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Three things to know about IBM and IBM BigInsights v4

- More flexible packaging
 - IBM InfoSphere BigInsights is now IBM BigInsights for Apache Hadoop
 - Data Science positioning resonating with market
 - Open Source components are available independently from IBM's value-added capabilities
- Features data scientists and analysts will love
 - Machine learning using Big R for data scientists
 - Big SQL enhancements for analysts and developers
 - Current Open Source Apache packages
- IBM is committed to open source
 - IBM named platinum sponsor of Open Data Platform initiative
 - Goal is to drive both open source and standards to accelerate innovation

Three things to know about IBM and IBM BigInsights v4

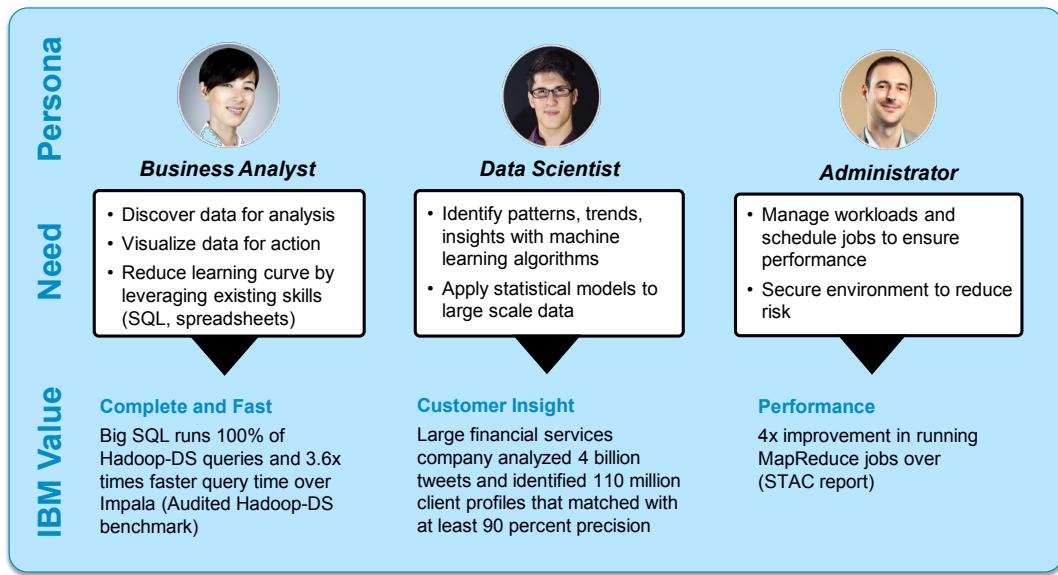
Every organization sees Hadoop as providing an open-source, rapidly-evolving platform that is capable of collecting and economically storing a very large corpus of highly variable types of data and making it available.

And yet many organizations are not yet fully realizing the value of Hadoop due to the lack of skills data scientists and developers to extract the valuable insight, or because of the complexity to scale the Hadoop environment.

In order to drive Hadoop adoption, organizations require advances:

- to have the most powerful analytics in their hands
- to distill experience and build skills to drive time to value faster
- to easily incorporate Hadoop into a broader data architecture

Enabling personas with capabilities



Introduction to IBM Open Platform with Apache Hadoop

© Copyright IBM Corporation 2015

Enabling personas with capabilities

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

New capabilities in IBM BigInsights v4

1

BigInsights Data Scientist module:

Accelerate data science teams with advanced analytics to extract valuable insights from Hadoop

- **Big R:** Statistical analysis and distributed frames using entire Hadoop cluster
- **Machine Learning:** Machine Learning algorithms in R optimized for Hadoop
- **Text Analytics:** Text extraction via business web tooling

2

BigInsights Analyst module:

Leverage existing skills to find and visualize data across all sources including Hadoop

- **Big SQL:** Hbase support, High Availability
- **BigSheets:** Geospatial support and Big SQL Integration

3

BigInsights Enterprise Management module:

Ensure scalability, performance and security of Hadoop clusters

- Multi-tenant scheduling
- Multi-instance support with data isolation

All built upon:

4

IBM Open Platform with Apache Hadoop:

Free (with optional paid support) product use version of 100% Open Source Apache Hadoop distribution

- **Apache Ambari:** Hadoop cluster administration GUI / Apache YARN and MapReduce 2
- **Apache Spark:** High-performance and flexible big data processing framework
- **Open Source:** Currency updates, including Hadoop 2.6

Introduction to IBM Open Platform with Apache Hadoop

© Copyright IBM Corporation 2015

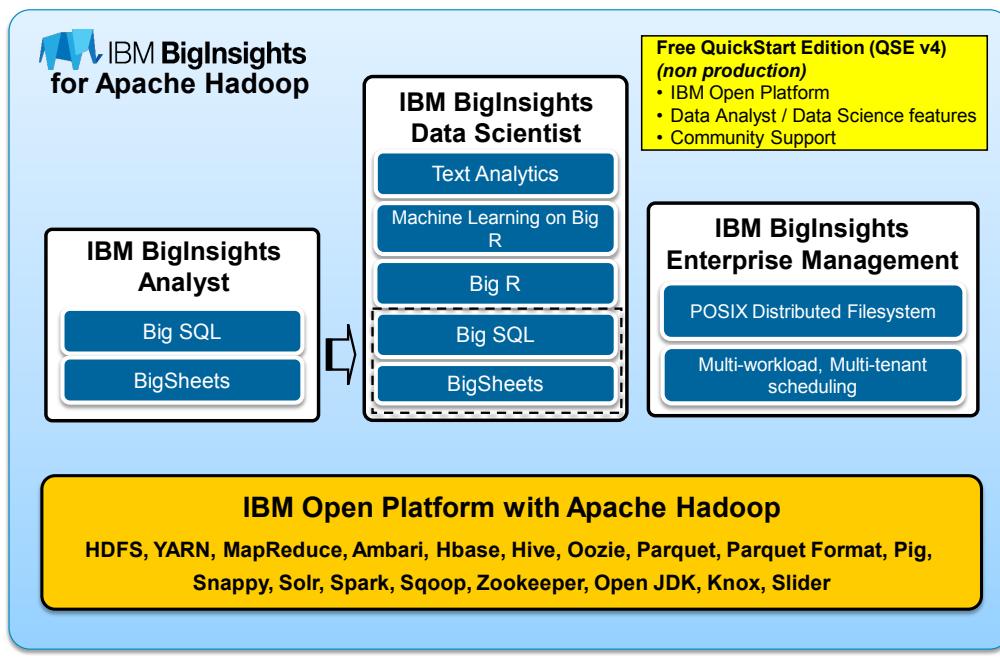
New capabilities in IBM BigInsights v4

This section of this course deals with the **IBM Open Platform with Apache Hadoop** exclusively.

This is the open-source base onto which the IBM BigInsights add-ins are installed.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Package structure



Introduction to IBM Open Platform with Apache Hadoop

© Copyright IBM Corporation 2015

Package structure

The IBM Open Platform with Apache Hadoop has the listed open-source components.

The following components are included with the various IBM BigInsights add-ins:

- Big SQL: SQL engine on Hadoop
- BigSheets: Data discovery and visualization
- Big R: Statistical analysis and Hadoop scale
- System ML: Machine learning through Big R
- Text Analytics: Text extraction with business tooling

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

New features in IBM BigInsights v4, open source and/or BigInsights add-ins:

- **R distributed:** Statistical analysis and distributed frames using entire Hadoop cluster
- **Machine Learning:** Machine learning algorithms optimized for Hadoop
- **Text Analytics:** Text extraction w/ business web tooling
- **Scalability:** Multi-tenant scheduling, Multi-cluster support, Multi-workload, Dynamic Runtime Elasticity (Elastic 4.1 FPO using Hadoop Connector and Symphony 7.1)
- **Apache Spark:** High-performance and flexible big data processing framework
- **Apache Ambari:** Hadoop cluster administration GUI
- **Additional capabilities and enhancements:** Spatiotemporal toolkit; ESS 4.1 improvements (GPFS, aka Spectrum Scale); encryption, geo-replication; data recovery tools; Big SQL features (HBase, performance improvements, additional federation sources, High Availability)

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

IBM BigInsights for Apache Hadoop offering suite

IBM BigInsights v4	IBM BigInsights Quick Start Edition	IBM Open Platform with Apache Hadoop	Elite Support for IBM Open Platform with Apache Hadoop	BigInsights Analyst Module	BigInsights Data Scientist Module	BigInsights Enterprise Mgmt Module	BigInsights for Apache Hadoop
Apache Hadoop Stack: Ambari, HDFS, YARN, MapReduce, Hbase, Hive, Oozie, Parquet, Parquet Format, Pig, Snappy, Solr, Spark, Sqoop, Zookeeper, Open JDK, Knox, Slider	✓	✓	✓	*	*	*	✓
Big SQL: 100% ANSI compliant, high performant, secure SQL engine	✓			✓	✓		✓
BigSheets: spreadsheet-like interface for discovery and visualization	✓			✓	✓		✓
Big R: advanced statistical and data mining	✓				✓		✓
Machine Learning with Big R: machine learning algorithms apply to Hadoop data set	✓				✓		✓
Advanced Text Analytics: visual tooling to annotate automated text extraction	✓				✓		✓
Enterprise Mgmt: Enhanced cluster and resource management and POSIX-compliant FS						✓	✓
Governance Catalog				✓	✓		✓
Cognos BI, InfoSphere Streams, Watson Explorer, Data Click							✓

Introduction to IBM Open Platform with Apache Hadoop

© Copyright IBM Corporation 2015

IBM BigInsights for Apache Hadoop offering suite

This shows the components of the various offering suites.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Pricing and licensing

Products	BigInsights Quick Start	IBM Open Platform with Apache Hadoop	Elite Support for IBM Open Platform with Apache Hadoop	BigInsights Analyst Module	BigInsights Data Scientist Module	BigInsights Enterprise Management Module	BigInsights for Apache Hadoop
Pricing Terms	Free	Free	Yearly Subscription Only	Perpetual or Monthly License			
Support provided	Community	Community	IBM 24x7 support				
Usage License	Non-production, five node cap	Production Usage					
Pricing Model	Free	Free	Node based pricing				
Access through	ibm.com/hadoop		Passport Advantage				

Community support, where noted, is provided via Hadoop Dev:

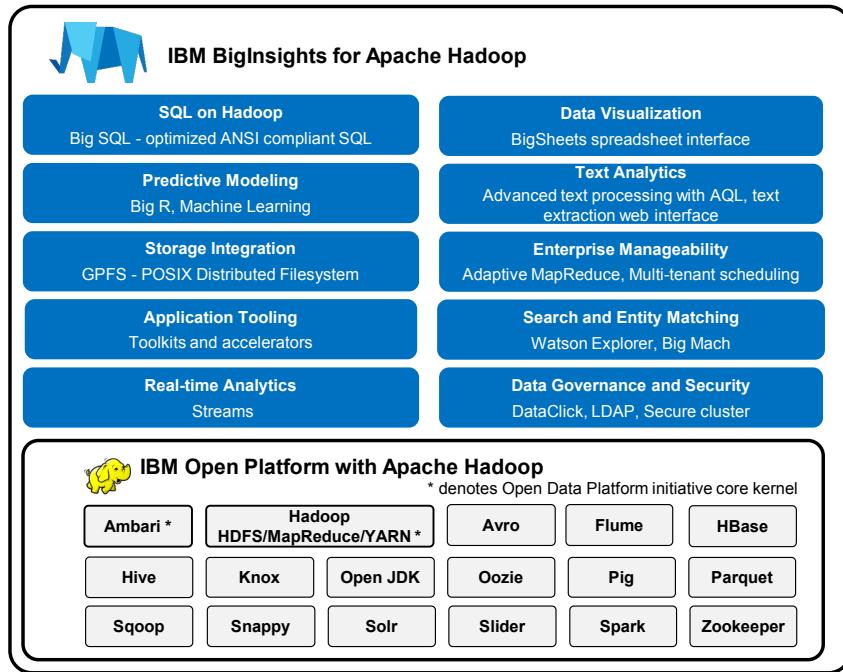
- over 100,000 visitors since inception
- modeled after StackOverflow, the most popular developer Q&A site on web

Pricing and licensing

This is a pricing overview for the various offerings. Consult IBM sales for pricing details.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Components



Introduction to IBM Open Platform with Apache Hadoop

© Copyright IBM Corporation 2015

Components

Currently the components of the Open Data Platform are:

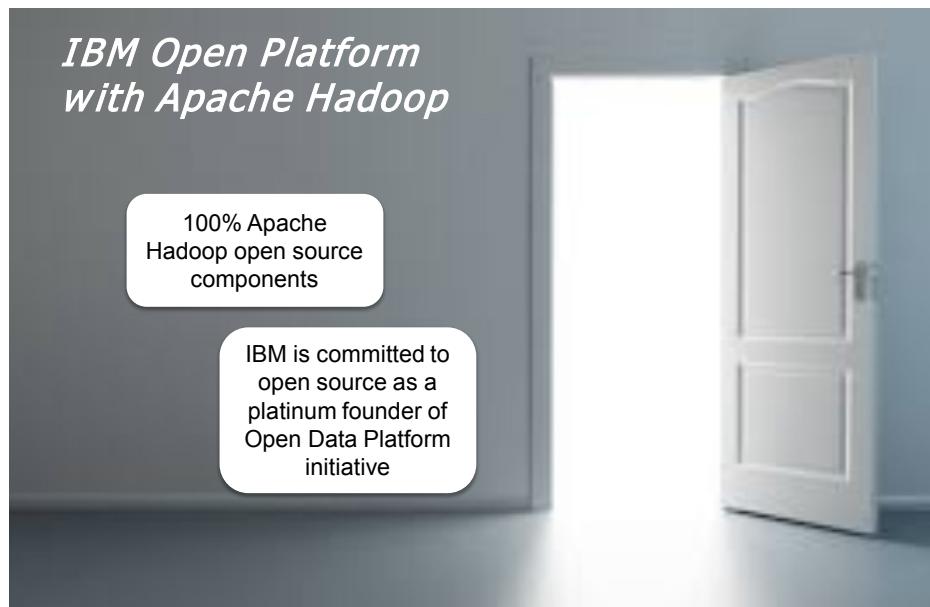
- Ambari
- HDFS
- MapReduce 2
- YARN

These are marked with an asterisk in the chart above.

Other candidate components from the Hadoop open source ecosystem are shown in grey (without the asterisk) and many of these are likely to be formally added in the future to the Open Data Platform consortium's standard base.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Maintain open source currency for interoperability



Introduction to IBM Open Platform with Apache Hadoop

© Copyright IBM Corporation 2015

Maintain open source currency for interoperability

All existing components from IBM InfoSphere BigInsights v3.0 have been updated to the latest versions of each of the respective components.

Major changes include:

- native support for rolling upgrades for Hadoop services
 - completed or in-process application work restarts at point of restart, not beginning
- support for long-running applications within YARN for enhanced reliability and security
- heterogeneous storage in HDFS for in-memory, SSD in addition to HDD
 - optimizes applications based on data access speed required

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

New open source additions:

- Spark 1.2.1: in-memory distributed compute engine
 - dramatic performance increases over MapReduce
 - emerging capabilities for streaming, SQL, machine learning and graph processing
- simplifies developer experience, leveraging Java, Python and Scala languages
- key capability for advanced Hadoop and Data Scientist users
- Ambari 1.7: operational framework for provisioning, managing and monitoring Apache Hadoop clusters
 - installation of individual components increases speed to value

Why is IBM involved in the ODP?

- the development of a common core will accelerate innovation
- IBM has committed to open source currency in future releases
- smooth application compatibility across vendor offerings will result

ODP vs. ASF

- ODP supports the ASF mission
- ASF provides a governance model around individual projects within open source software development
- ODP aims to provide a vendor-led consistent packaging model for multiple Apache components

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

IBM Open Platform v4.0 with Apache Hadoop

- All existing components from BigInsights v3.0 have been updated to their latest versions:
 - Native support for rolling upgrades for Hadoop services
 - Completed or in process application work restarts at point of restart, not beginning
 - Support for long-running applications within YARN for enhanced reliability and security
 - Heterogeneous storage in HDFS for in-memory, SSD in addition to HDD
 - Optimize applications based on data access speed required
- New Open Source Additions:
 - Spark 1.2.1 - In-memory distributed compute engine
 - Dramatic performance increases over MapReduce
 - Emerging capabilities for streaming, SQL, machine learning and graph processing
 - Simplifies developer experience, leveraging Java, Python and Scala languages
 - Ambari 1.7 - Operational framework for provisioning, managing and monitoring Apache Hadoop clusters
 - Installation of individual components increases speed to value
 - Key capability for advanced Hadoop and Data Scientist users

IBM Open Platform v4.0 with Apache Hadoop

For organizations upgrading from BigInsights v3 or earlier:

- For in-place migration: Shut down BigInsights, then layer new v4.0 services on top. One script to shut down. Extract from current all configuration files, then lay down Ambari, then launch and install new services. Recipes that you run when installing Ambari, can also run via rest APIs. Instead of using the wizard, use the rest APIs install. As much as possible is automated. Then data migration from HDFS from 2.4 to 2.6; most open source stuff there are ways, hive/HBase. Our catalog will be most complex, export from old to new.
- In-place migration does not have a single push button. It is a step-and-check process.

Open source currency

- IBM is as current as competitors in every package shipped
- IBM uses the Apache Ambari installer in anticipation of the Open Data Platform Initiative certifying a future version of Apache Ambari

No more massive images to download:
download a small package and then
download only the components needed

Component Name	Version
Ambari	1.7.0
Avro	1.7.7
Flume	1.5.2
Hadoop	2.6
HBase	0.98.8
Hive	0.14.0
Knox	0.5.0
Oozie	4.0.1
Pig	0.14.0
Parquet (hadoop)	1.5.0
Parquet (format)	2.1.0
Spark	1.2.1
Snappy	1.0.5
Sqoop	1.4.5
Solr	4.10.3
Slider	0.6.0
Zookeeper	3.4.5

Open source currency

The versions of the component software are shown as of mid-June 2015. You should consult with IBM to find the current versions of the components as this information now changes frequently.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Components of the Open Data Platform



Introduction to IBM Open Platform with Apache Hadoop

© Copyright IBM Corporation 2015

Components of the Open Data Platform

The ODP Core will initially focus on Apache Hadoop (inclusive of HDFS, YARN, and MapReduce) and Apache Ambari.

Once the ODP members and processes are well established, the scope of the ODP Core is expected to expand to include other open source projects; some of these projects are listed above.

All of the topics listed above will be discussed in this section of this course.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit summary

- List the major components of the IBM Open Data Platform
- List the common components of the Apache Hadoop stack
- Describe the functionality of each open source component

Exercise 1

Additional exploration of the lab environment

Exercise 1: Additional exploration of the lab environment

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1: Additional exploration of the lab environment

Purpose:

This is an additional exploratory exercise of the lab environment. You will access your lab environment and use a command line window. You will be exploring the directory structure of the Linux system that you will be using.

VM Hostname: <http://ibmclass.localdomain>

User/Password: **biadmin / biadmin**
root/dalvm3

Task 1. Login to your image and perform some basic setup.

You will need verify that your hostname and IP address are setup correctly, and will make changes if they are needed.

Note: If you powered your VMware image from an earlier session, this verification of hostname and IP address should be repeated.

1. Connect to and login to your VMware image with the user as **biadmin** and the password **biadmin**.
The desktop of your Linux system is displayed (desktop of the user biadmin).
To avoid having to constantly login, you can change your screen saver settings.
2. At the top left border of your Linux window, choose **System > Preferences > Screensaver**, uncheck the box **Lock screen when screen saver is active**, and then click the **Close** button.
3. To open a terminal window, right-click the desktop and select **Open in Terminal**.

This terminal window can be resized. You may find it useful to stretch the right side to make it wider, as some commands are long and some responses will be wide; any text that is longer than the default line width will be word-wrapped to the next line.

You may want to have more than one window open. When you do this, place multiple windows so that they do not overlap completely.

You can also have more than one terminal window, and have the choice of several, simultaneous desktops that you can alternate between:



Here the desktop on the left is currently selected, and the second arrow points to an alternate/extension desktop for the same login session.

4. In your terminal window, type `ifconfig`.

Take note of the IP address (inet addr) for your environment and your Ethernet adapter (eth0, or eth1, etc., or eth4 as in this case).

5. To see the hostname, type the Linux command `hostname` in your terminal window.

The results appear as follows:

```
[biadmin@ibmclass ~]$ hostname
ibmclass.localdomain
```

6. Check your /etc/hosts file by using `cat /etc/hosts` to verify whether the IP address and hostname are recorded correctly.

The results appear as follows:

```
[biadmin@ibmclass ~]$ cat /etc/hosts
10.0.0.68 ibmclass.localdomain ibmclass
127.0.0.1 localhost.localdomain localhost
```

This listing shows that the hostname and IP address are correctly set in the /etc/hosts file. If you find that the values in steps 4 and 5 are not the same as what you find in step 6, you will have to edit the /etc/hosts file. Since /etc/hosts is a system file, you will need to edit that file as an administrator (root).

If your settings in the /etc/hosts file are correct, proceed to step 8.

7. If you need to edit /etc/hosts, open the file **/etc/hosts** as an administrator (root):
 - a. Either: **sudo gedit /etc/hosts**
or:

```
[biadmin@ibmclass ~]$ su -
Password:
[root@ibmclass ~]# gedit /etc/hosts
```

 For sudo, use your biadmin password. For substitute user (su -), use the root password.
 - b. Edit the line that shows the current system to match the information you found in steps 4 and 5.
 - c. Save and close.
 - d. If you used su, exit from the root login with **Ctrl-D** or **exit**.
8. If you have connectivity problems later in this exercise or in another exercise, perform steps 5 and 6 and edit the /etc/hosts file again, if necessary.
Repeating these steps will not normally be necessary unless your lab environment is restarted, either deliberately or overnight.

Task 2. Start Ambari and the Hadoop processing environment.

In order to work with your Hadoop cluster environment, you need to start the Ambari server and use Ambari to initialize the cluster.

The Ambari server takes some time to go through startup and initialization. While it is starting, you can move on to other activities.

1. Launch **Firefox**, and then if necessary, navigate to the **Ambari** login page, <http://localhost:8080>.
You are running a single-host Hadoop cluster (pseudo-distributed mode) and thus you can connect to the Ambari server on port 8080.
2. Log in to the **Ambari** console as **admin/admin**.
On the left side of the browser are the statuses of all the services. If they are currently yellow, wait a couple of minutes for them to become red before you start them up.

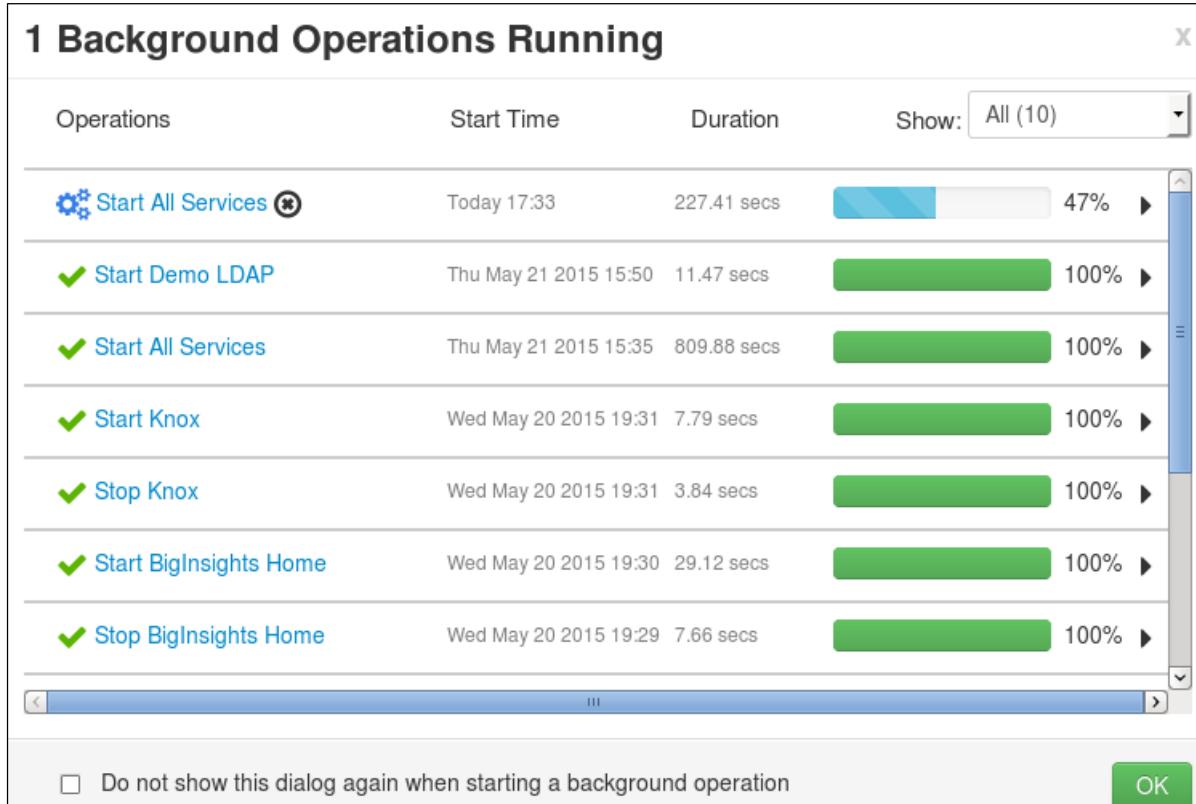
This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

3. Once all the statuses are red, at the bottom of the left side, click **Actions** and then click **Start All** to start the services.



This will take several minutes to complete.

This will bring up a window showing that Start All Services is running in the background (running tasks are colored blue). Completed tasks are colored green, and failed tasks are colored red.



- When the services have started successfully, click **OK**, and then close Firefox. Starting all services may take as much as 10 minutes. You will be revisiting the Ambari Server in its own unit and will look at the various pages in more detail at that time. You will now explore your Hadoop cluster environment.

Task 3. Explore the placement of files in the Linux host-system environment.

You will review the placement of files in the Linux host-system environment so that you can become familiar with where the open-source ODP binaries, all configuration files, and the BigInsights software are installed. You need this knowledge to be able to work best with a Hadoop cluster.

This Hadoop cluster is unique however, as it is a single-node cluster. This is called pseudo-distributed mode since it simulates a multi-node cluster.

The Open Data Platform software is found in the /usr/iop directory. This is the Apache open source software that forms the basis for the Open Data Platform Initiative. (IOP = IBM Open Platform).

1. In a new terminal window, type `cd` to change to your home directory.
2. Type `pwd` to verify that you are in biadmin's home directory.
3. To change the directory to /usr/iop, type `cd /usr/iop`.

The results appear as follows:

```
[biadmin@ibmclass Desktop]$ cd
[biadmin@ibmclass ~]$ pwd
/home/biadmin
[biadmin@ibmclass ~]$ cd /usr/iop
[biadmin@ibmclass iop]$ █
```

4. To review the files and directories in the /usr/iop directory, type `ls -l`.

```
[biadmin@ibmclass iop]$ ls -l
total 8
drwxr-xr-x. 19 root root 4096 Apr 15 13:07 4.0.0.0
drwxr-xr-x.  2 root root 4096 Jun 11 11:32 current
```

The first of these subdirectories (4.0.0.0) is the release level of the ODP software that you are working with; the version in your lab environment may differ if the software has been updated.

5. To display the contents of the 4.0.0.0 directory, type `ls -l 4*`.
The results appear similar to the following:

```
[biadmin@ibmclass iop]$ ls -l 4*
total 68
drwxr-xr-x. 18 root root 4096 Apr 15 13:07 etc
drwxr-xr-x. 7 root root 4096 Apr 15 12:45 flume
drwxr-xr-x. 9 root root 4096 Apr 15 12:51 hadoop
drwxr-xr-x. 6 root root 4096 Apr 15 12:44 hadoop-hdfs
drwxr-xr-x. 5 root root 4096 Apr 15 12:43 hadoop-mapreduce
drwxr-xr-x. 6 root root 4096 Apr 15 12:43 hadoop-yarn
drwxr-xr-x. 8 root root 4096 Apr 15 12:50 hbase
drwxr-xr-x. 8 root root 4096 Apr 15 12:51 hive
drwxrwxr-x. 7 hive hive 4096 Apr 15 12:52 hive-hcatalog
drwxr-xr-x. 8 root root 4096 Jun 11 11:33 knox
drwxr-xr-x. 11 oozie root 4096 Apr 15 13:19 oozie
drwxr-xr-x. 7 root root 4096 Apr 15 13:01 pig
drwxr-xr-x. 6 root root 4096 Apr 15 13:02 slider
drwxr-xr-x. 8 root root 4096 Apr 15 13:04 solr
drwxr-xr-x. 8 root root 4096 Apr 15 13:06 spark
drwxr-xr-x. 6 root root 4096 Apr 15 13:07 sqoop
drwxr-xr-x. 7 root root 4096 Apr 15 12:43 zookeeper
```

Take note of the user IDs associated with the various directories. Some have a user name that is same as the software held in the directory; some are owned by root. You will be looking at the standard users in the Apache Ambari unit when you explore details of the Ambari Server and work with the management of your cluster.

6. To view the **current** subdirectory which has a set of links that point back to files and subdirectories in the 4.*.*.* directory, type `ls -l current`.

```
[biadmin@ibmclass iop]$ ls -l current
total 0
lrwxrwxrwx 1 root 25 Jun 2 18:02 accumulo-client -> /usr/iop/4.0.0.0/accumulo
lrwxrwxrwx 1 root root 25 Jun 2 18:02 accumulo-gc -> /usr/iop/4.0.0.0/accumulo
lrwxrwxrwx 1 root root 25 Jun 2 18:02 accumulo-master -> /usr/iop/4.0.0.0/accumulo
lrwxrwxrwx 1 root root 25 Jun 2 18:02 accumulo-monitor -> /usr/iop/4.0.0.0/accumulo
lrwxrwxrwx 1 root root 25 Jun 2 18:02 accumulo-tablet -> /usr/iop/4.0.0.0/accumulo
lrwxrwxrwx 1 root root 25 Jun 2 18:02 accumulo-tracer -> /usr/iop/4.0.0.0/accumulo
lrwxrwxrwx 1 root root 23 Jun 2 18:02 falcon-client -> /usr/iop/4.0.0.0/falcon
lrwxrwxrwx 1 root root 23 Jun 2 18:02 falcon-server -> /usr/iop/4.0.0.0/falcon
lrwxrwxrwx 1 root root 22 Jun 2 18:02 flume-server -> /usr/iop/4.0.0.0/flume
lrwxrwxrwx 1 root root 23 Jun 2 18:02 hadoop-client -> /usr/iop/4.0.0.0/hadoop
lrwxrwxrwx 1 root root 28 Jun 2 18:02 hadoop-hdfs-client -> /usr/iop/4.0.0.0/hadoop-
hdfs
lrwxrwxrwx 1 root root 28 Jun 2 18:02 hadoop-hdfs-datanode -> /usr/iop/4.0.0.0/hadoop-
hdfs
lrwxrwxrwx 1 root root 28 Jun 2 18:02 hadoop-hdfs-journalnode ->
/usr/iop/4.0.0.0/hadoop-hdfs
lrwxrwxrwx 1 root root 28 Jun 2 18:02 hadoop-hdfs-namenode -> /usr/iop/4.0.0.0/hadoop-
hdfs
lrwxrwxrwx 1 root root 28 Jun 2 18:02 hadoop-hdfs-nfs3 -> /usr/iop/4.0.0.0/hadoop-hdfs
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

```

lrwxrwxrwx 1 root root 28 Jun  2 18:02 hadoop-hdfs-portmap -> /usr/iop/4.0.0.0/hadoop-
hdfs
lrwxrwxrwx 1 root root 28 Jun  2 18:02 hadoop-hdfs-secondarynamenode ->
/usr/iop/4.0.0.0/hadoop-hdfs
lrwxrwxrwx 1 root root 33 Jun  2 18:02 hadoop-mapreduce-client ->
/usr/iop/4.0.0.0/hadoop-mapreduce
lrwxrwxrwx 1 root root 33 Jun  2 18:02 hadoop-mapreduce-historyserver ->
/usr/iop/4.0.0.0/hadoop-mapreduce
lrwxrwxrwx 1 root root 28 Jun  2 18:02 hadoop-yarn-client -> /usr/iop/4.0.0.0/hadoop-
yarn
lrwxrwxrwx 1 root root 28 Jun  2 18:02 hadoop-yarn-nodemanager ->
/usr/iop/4.0.0.0/hadoop-yarn
lrwxrwxrwx 1 root root 28 Jun  2 18:02 hadoop-yarn-resourcemanager ->
/usr/iop/4.0.0.0/hadoop-yarn
lrwxrwxrwx 1 root root 28 Jun  2 18:02 hadoop-yarn-timelineserver ->
/usr/iop/4.0.0.0/hadoop-yarn
lrwxrwxrwx 1 root root 22 Jun  2 18:02 hbase-client -> /usr/iop/4.0.0.0/hbase
lrwxrwxrwx 1 root root 22 Jun  2 18:02 hbase-master -> /usr/iop/4.0.0.0/hbase
lrwxrwxrwx 1 root root 22 Jun  2 18:02 hbase-regionserver -> /usr/iop/4.0.0.0/hbase
lrwxrwxrwx 1 root root 21 Jun  2 18:02 hive-client -> /usr/iop/4.0.0.0/hive
lrwxrwxrwx 1 root root 21 Jun  2 18:02 hive-metastore -> /usr/iop/4.0.0.0/hive
lrwxrwxrwx 1 root root 21 Jun  2 18:02 hive-server2 -> /usr/iop/4.0.0.0/hive
lrwxrwxrwx 1 root root 30 Jun  2 18:02 hive-webhcat -> /usr/iop/4.0.0.0/hive-hcatalog
lrwxrwxrwx 1 root root 22 Jun  2 18:02 kafka-broker -> /usr/iop/4.0.0.0/kafka
lrwxrwxrwx 1 root root 21 Jun  2 18:02 knox-server -> /usr/iop/4.0.0.0/knox
lrwxrwxrwx 1 root root 23 Jun  2 18:02 mahout-client -> /usr/iop/4.0.0.0/mahout
lrwxrwxrwx 1 root root 22 Jun  2 18:02 oozie-client -> /usr/iop/4.0.0.0/oozie
lrwxrwxrwx 1 root root 22 Jun  2 18:02 oozie-server -> /usr/iop/4.0.0.0/oozie
lrwxrwxrwx 1 root root 24 Jun  2 18:02 phoenix-client -> /usr/iop/4.0.0.0/phoenix
lrwxrwxrwx 1 root root 20 Jun  2 18:02 pig-client -> /usr/iop/4.0.0.0/pig
lrwxrwxrwx 1 root root 29 Jun  2 18:02 ranger-admin -> /usr/iop/4.0.0.0/ranger-admin
lrwxrwxrwx 1 root root 32 Jun  2 18:02 ranger-usersync -> /usr/iop/4.0.0.0/ranger-
usersync
lrwxrwxrwx 1 root root 23 Jun  2 18:02 slider-client -> /usr/iop/4.0.0.0/slider
lrwxrwxrwx 1 root root 22 Jun  2 18:02 spark-client -> /usr/iop/4.0.0.0/spark
lrwxrwxrwx 1 root root 22 Jun  2 18:02 spark-historyserver -> /usr/iop/4.0.0.0/spark
lrwxrwxrwx 1 root root 22 Jun  2 18:02 spark-thriftserver -> /usr/iop/4.0.0.0/spark
lrwxrwxrwx 1 root root 22 Jun  2 18:02 sqoop-client -> /usr/iop/4.0.0.0/sqoop
lrwxrwxrwx 1 root root 22 Jun  2 18:02 sqoop-server -> /usr/iop/4.0.0.0/sqoop
lrwxrwxrwx 1 root root 22 Jun  2 18:02 storm-client -> /usr/iop/4.0.0.0/storm
lrwxrwxrwx 1 root root 22 Jun  2 18:02 storm-nimbus -> /usr/iop/4.0.0.0/storm
lrwxrwxrwx 1 root root 36 Jun  2 18:02 storm-slider-client -> /usr/iop/4.0.0.0/storm-
slider-client
lrwxrwxrwx 1 root root 22 Jun  2 18:02 storm-supervisor -> /usr/iop/4.0.0.0/storm
lrwxrwxrwx 1 root root 20 Jun  2 18:02 tez-client -> /usr/iop/4.0.0.0/tez
lrwxrwxrwx 1 root root 26 Jun  2 18:02 zookeeper-client -> /usr/iop/4.0.0.0/zookeeper
lrwxrwxrwx 1 root root 26 Jun  2 18:02 zookeeper-server -> /usr/iop/4.0.0.0/zookeeper
[biadmin@ibmclass iop]$
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

There are sometimes multiple links back to the one directory.

The software that will be executed at this time (current) is determined by these links. Since you can have multiple versions of software installed (4.0.0,0, 4.1.0.0, etc.), it is possible to install the software for an upcoming upgrade and then later have Ambari install a rolling upgrade that will systematically change the links as appropriate to files in the new, upgraded software directories.

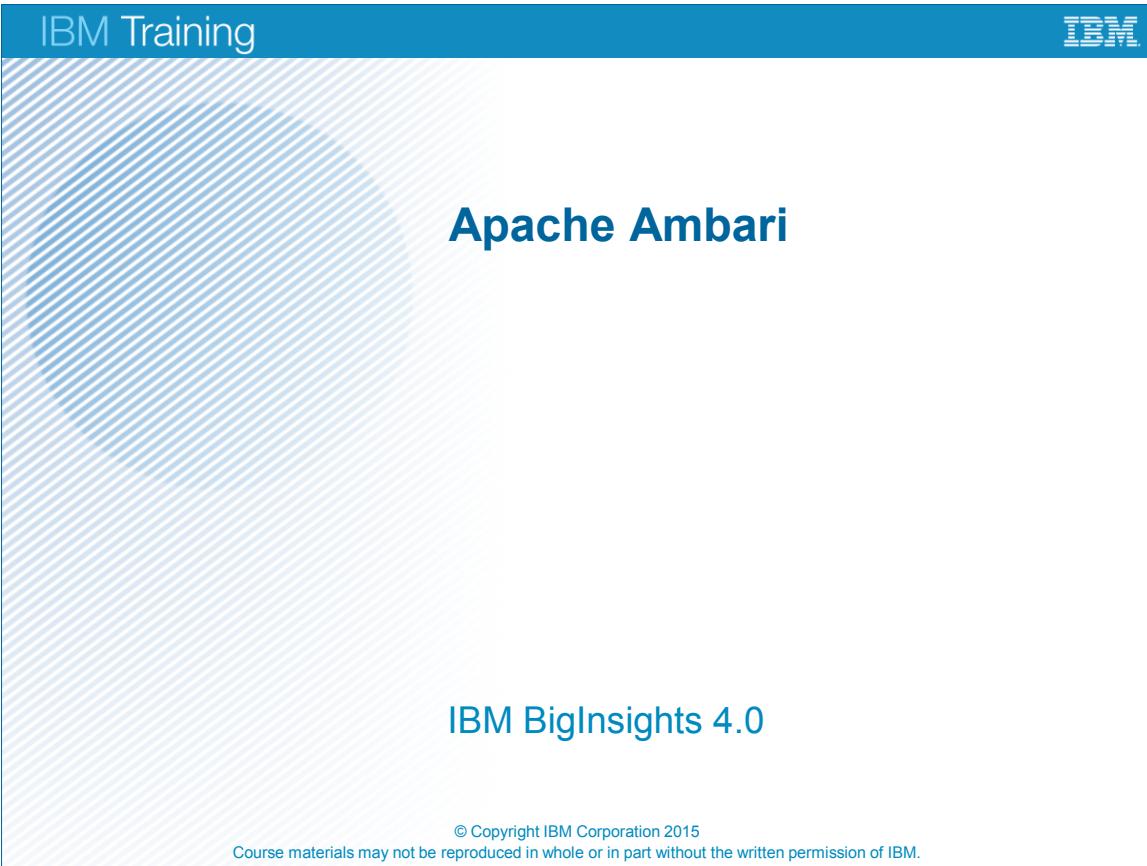
7. Close all open windows.

Results:

You explored the lab environment, and the command line window. You also explored the directory structure of the Linux system.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit 7 Apache Ambari



The slide features a blue header bar with 'IBM Training' on the left and the IBM logo on the right. The main content area has a light blue diagonal striped background. The title 'Apache Ambari' is centered in large blue text. Below it, 'IBM BigInsights 4.0' is also centered in blue text. At the bottom of the slide, there is a copyright notice: '© Copyright IBM Corporation 2015' and 'Course materials may not be reproduced in whole or in part without the written permission of IBM.'

IBM Training

IBM

Apache Ambari

IBM BigInsights 4.0

© Copyright IBM Corporation 2015
Course materials may not be reproduced in whole or in part without the written permission of IBM.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit objectives

- Understand the purpose of Apache Ambari in the Open Data Platform
- Understand the overall architecture of Ambari, and Ambari's relation to other services and components of a Hadoop cluster
- List the functions of the main components of Ambari
- Explain how to start and stop services from the Ambari Web Console

Agenda

- The functionality offered by the Apache Ambari server
- The web interface and dashboard
- Service metrics
- Managing Ambari, including adding, starting, and configuring services
- Command line interface to the Ambari server
- Ambari terminology



Apache Ambari

© Copyright IBM Corporation 2015

Agenda

Origin of the word *Ambari*:

Ambari is a locality in Guwahati, India. Located North West of Guwahati, it is a site for important archaeological excavations related to ancient Assam. Guwahati is a major commercial and educational hub of Assam and Northeast India, and is home to premier institutions such as the Indian Institute of Technology Guwahati, Gauhati University, and Cotton College. The city is a major center for cultural activities and sporting events, as well as a center for administrative and political activities of Assam, and important regional hub for transportation.

Apache Ambari is one of the foundation software components of the Open Data Platform initiative as it is the mechanism for adding, starting, and configuring services.

The IBM add-ins that are part of IBM BigInsights are installed from here.

Ambari replaces the BigInsights Console of prior releases of IBM InfoSphere BigInsights.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Functionality of Apache Ambari

- Ambari enables System Administrators to:
 - provision a Hadoop cluster
 - Ambari provides a step-by-step wizard for installing Hadoop services across any number of hosts
 - Ambari handles configuration of Hadoop services for the cluster
 - manage a Hadoop cluster
 - Ambari provides central management for starting, stopping, and reconfiguring Hadoop services across the entire cluster
 - monitor a Hadoop cluster
 - Ambari provides a dashboard for monitoring health and status of the Hadoop cluster
 - Ambari leverages **Ganglia** for metrics collection
 - Ambari leverages **Nagios** for system alerting and will send emails when your attention is needed (for example, a node goes down, remaining disk space is low, etc.)
- Ambari enables application developers and system integrators to:
 - easily integrate Hadoop provisioning, management, and monitoring capabilities to their own applications with the Ambari REST APIs

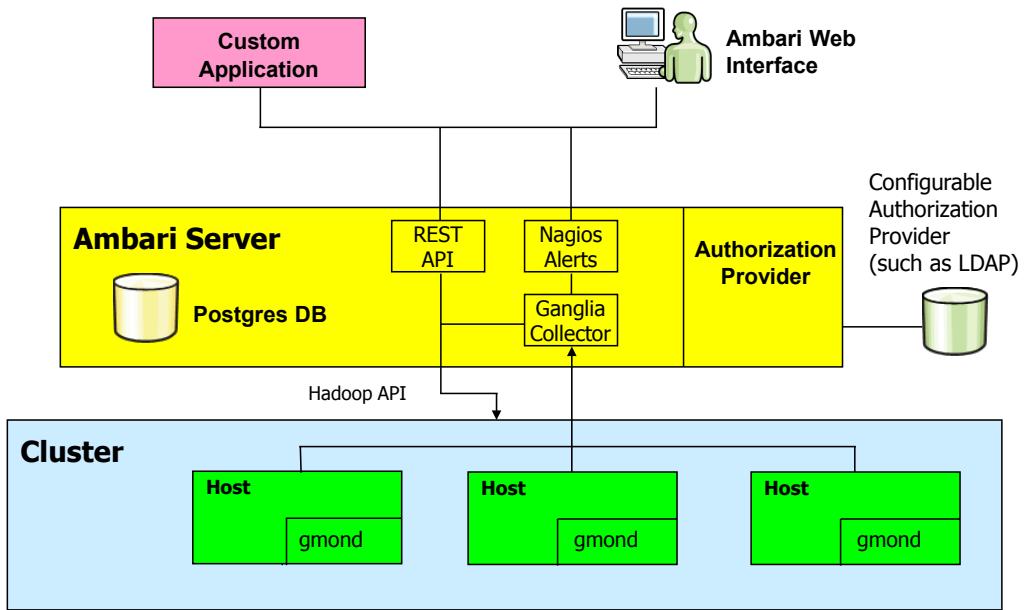
Functionality of Apache Ambari

The Apache Ambari project is aimed at making Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters. Ambari provides an intuitive, easy-to-use Hadoop management web user interface (UI) backed by its RESTful APIs.

Additional information is available on the Ambari wiki:

- <https://cwiki.apache.org/confluence/display/AMBARI/Ambari>

Ambari architecture



Apache Ambari

© Copyright IBM Corporation 2015

Ambari architecture

The main architectural components of **Ambari** are:

- **Ambari Agents:** the **Ganglia Monitors** (gmond) running on the individual hosts in the cluster collect metrics
- **Ambari Server** with the following components:
 - **Ganglia Collector** responsible for collecting information received from each host through Ganglia monitors running on the hosts
 - **Postgres RDBMS** stores the cluster configurations
 - **Authorization Provider** integrates with an organization's authentication/authorization provider such as the LDAP service
 - **Nagios** (optional) supports alerts and notifications
 - **REST API** integrates with the web-based front-end Ambari Web - this REST API can also be used by custom applications
- **Ambari Web:** a web-based interface for the users

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Similar functionality to Ambari is provided by the Cloudera Manager, since Cloudera is not currently a member of the Open Data Platform consortium.

Ganglia

- In anatomy, a ganglion (plural *ganglia*) is a nerve cell cluster or a group of nerve cell bodies located in the peripheral nervous system. Cells found in a ganglion are called ganglion cells, though this term is also sometimes used to refer specifically to retinal ganglion cells.
- In the software use of the term, Ganglia is a scalable distributed system monitor tool for high-performance computing systems such as clusters and grids. Its purpose is to allow the user to remotely view live or historical statistics (such as CPU load averages or network utilization) for all machines that are being monitored.
- *Gmond* (Ganglia monitor daemon) is a multi-threaded daemon which runs on each cluster node you want to monitor. Installation does not require having a common NFS filesystem or a database back-end, installing special accounts or maintaining configuration files. Gmond has four main responsibilities:
 - monitor changes in host state
 - announce relevant changes
 - listen to the state of all other ganglia nodes via a unicast or multicast channel
 - answer requests for an XML description of the cluster state
- Ganglia Meta Daemon (*gmetad*) provides federation in Ganglia by supporting a tree of point-to-point connections amongst representative cluster nodes to aggregate the state of multiple clusters. At each node in the tree, a Ganglia Meta Daemon periodically polls a collection of child data sources, parses the collected XML, saves all numeric, volatile metrics to round-robin databases and exports the aggregated XML over a TCP socket to clients. Data sources may be either *gmond* daemons, representing specific clusters, or other *gmetad* daemons, representing sets of clusters. Data sources use source IP address for access control and can be specified using multiple IP addresses for failover. The latter capability is natural for aggregating data from clusters since each *gmond* daemon has the entire state of its cluster.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

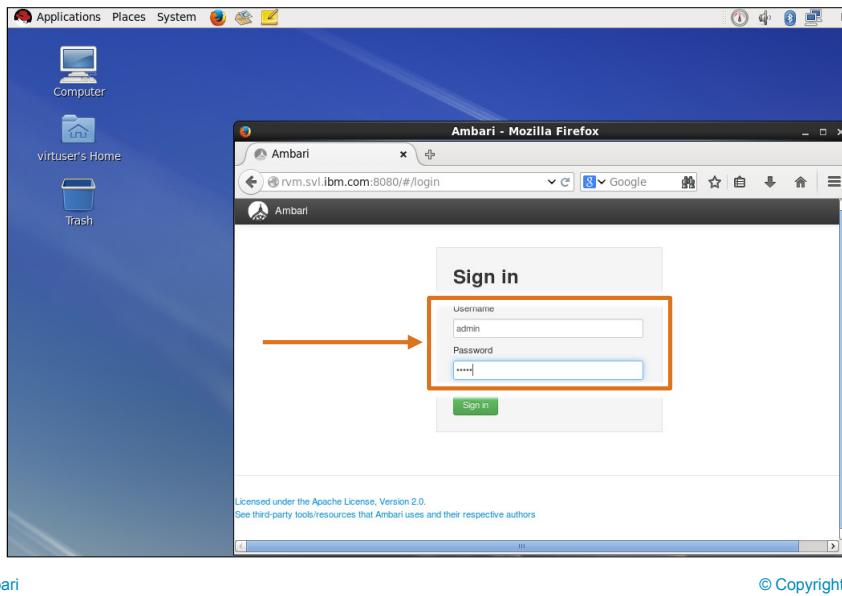
Nagios

- Nagios is open source software that provides computer system monitoring, network monitoring, and infrastructure monitoring. Nagios offers monitoring and alerting services for servers, switches, applications, and services. It alerts the users when things go wrong and alerts them a second time when the problem has been resolved.
- Nagios, originally created under the name *NetSaint*, was written and is currently maintained by Ethan Galstad along with a group of developers who actively maintain both the official and unofficial plugins ([http://en.wikipedia.org/wiki/Plugin_\(computing\)](http://en.wikipedia.org/wiki/Plugin_(computing))). Nagios is a recursive acronym "Nagios Ain't Gonna Insist On Sainthood," "Sainthood" being a reference to the original name *NetSaint*, which was changed in response to a legal challenge by owners of a similar trademark. "Agios" is also a transliteration of the Greek word ἄγιος which means "saint" or "holiness."

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Sign in to the Ambari web interface

- Click the **Firefox icon** on the System Tray (top left of Linux screen)
- URL is **localhost:8080** and sign in as **admin / admin** (default)



Sign in to the Ambari web interface

There are two Ambari interfaces to the outside world (through the firewall around the Hadoop cluster):

- Ambari web Interface, which you will review and use
- custom application API that allows programs to talk to Ambari

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Navigating the Ambari web interface

Main tabs:

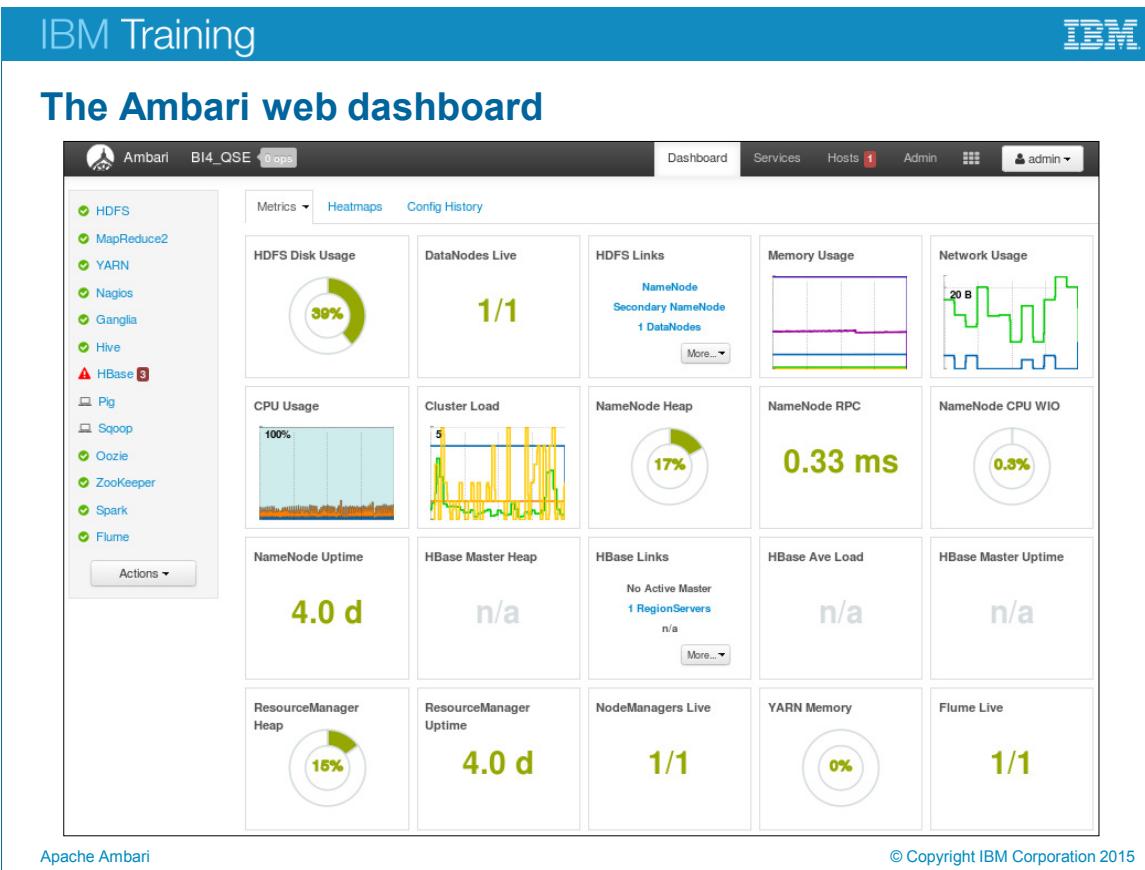
- Dashboard
- Hosts
- Admin
 - Repositories
 - Service Accounts
 - Security
- Views
- admin
 - About
 - Manage Ambari
 - Settings
 - Sign Out

Drop down to add a service, or start or stop all services

Navigating the Ambari web interface

Note that the "admin" tab is an icon that acts as a drop-down menu.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE



The Ambari web dashboard

This slide provides a preview of what you will see in the exercise at the end of this unit.

Notice the various components on the standard dashboard configuration. The typical items include:

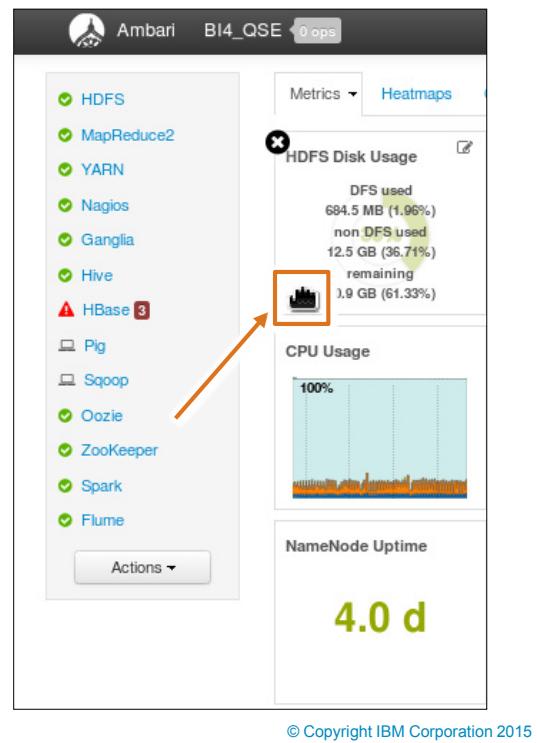
- HDFS Disk Usage: 38%
- DataNodes Live: 1/1; this shows the lab environment (pseudo distributed mode where everything is running in one node). A multi-node cluster is shown in the architecture diagram (shows 3 nodes only, though).

This system shown here has been up 4 days ("4.0 d" for NameNode Uptime and for the ResourceManager Uptime).

Note in the example, which components are running: all, except HBase. HBase is currently stopped (red triangle with exclamation point).

Metric details on the Ambari dashboard

- Use **Services** to monitor and manage selected services running in your Hadoop cluster
- All services installed in your cluster are listed in the leftmost **Services** panel; the metrics are shown in the main body of the dashboard
- Hover the cursor ("hand") over the individual entry on the dashboard
- Metric details are shown (see example at right)



Apache Ambari

© Copyright IBM Corporation 2015

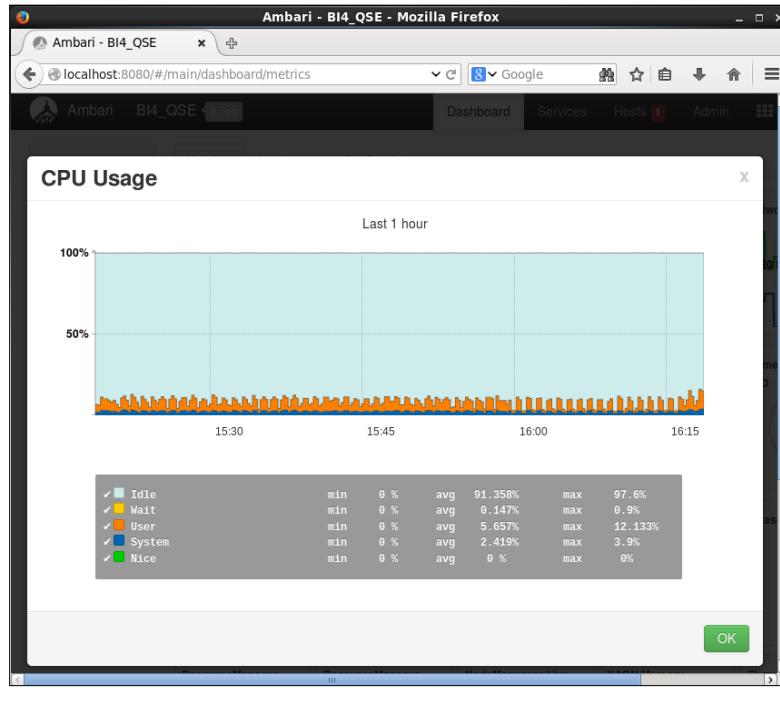
Metric details on the Ambari dashboard

By using the hand (pointed to by the arrow on the individual components), you can get the detailed metrics of the component.

The CPU usage metric detail is on the next slide.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Metric details for time-based cluster components



Apache Ambari

© Copyright IBM Corporation 2015

Metric details for time-based cluster components

For other components, such as CPU usage, you may be not as interested in the instantaneous metric value as you are with current disk usage, but you may be interested in the metric over a recent time period.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Service Actions/Alert and Health Checks

- Service Actions
 - drop-down menu
- Alerts and Health Checks
 - view results of health checks performed on the cluster by **Nagios**
 - the display lists each issue and its rating, sorted first by descending severity, then by descending time

Alerts and Health Checks	
✓ Percent DataNodes live	OK: total:<1>, affected:<0>
✓ Percent DataNodes with space available	OK: total:<1>, affected:<0>
✓ Secondary NameNode process	TCP OK - 0.001 second response time on port 8020
✓ NameNode process on rvm.svl.ibm.com	OK for 10 hours
✓ NameNode edit logs directory status on rvm.svl.ibm.com	OK for 10 hours
✓ Last checkpoint time	OK for 10 hours
✓ NameNode Web UI on	OK for 10 hours

To access more detailed information, select the native **Nagios GUI link** located at the upper right corner of the panel, using the Nagios credentials setup during installation to login to Nagios

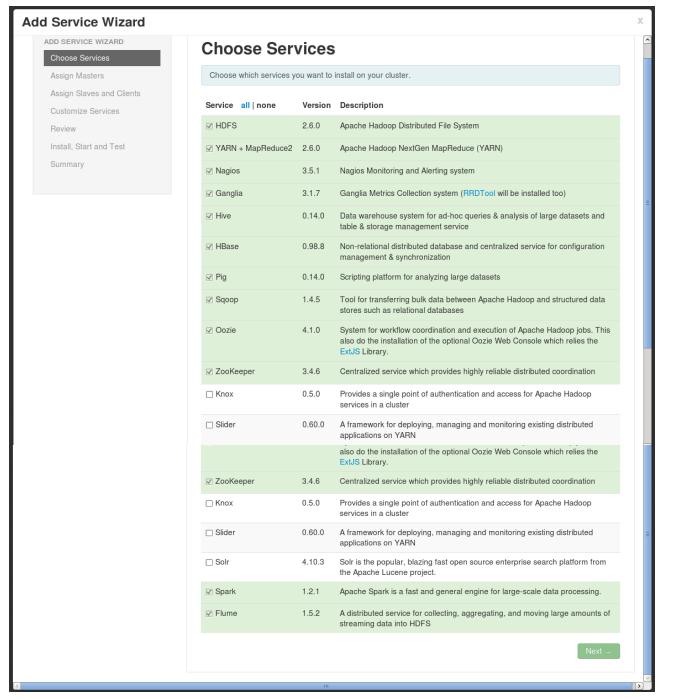
Service Actions/Alert and Health Checks

Ambari is intended to be the center for monitoring performance of the Hadoop cluster, as well as the center for generic and particular alert and health checks.

The Ambari Metrics System ("AMS") is a system for collecting, aggregating and serving Hadoop and system metrics in Ambari-managed clusters. This area is undergoing development and is likely to change in future releases. The slides here are intended to illustrate principles and show the current status of Ambari metrics, as of Ambari 1.7.0.

Add Service Wizard

- Background is pale-green for installed services
- Other services can be added by checking the service(s) and clicking **Next** to setup details



Apache Ambari

© Copyright IBM Corporation 2015

Add Service Wizard

The Add Service Wizard allows you to:

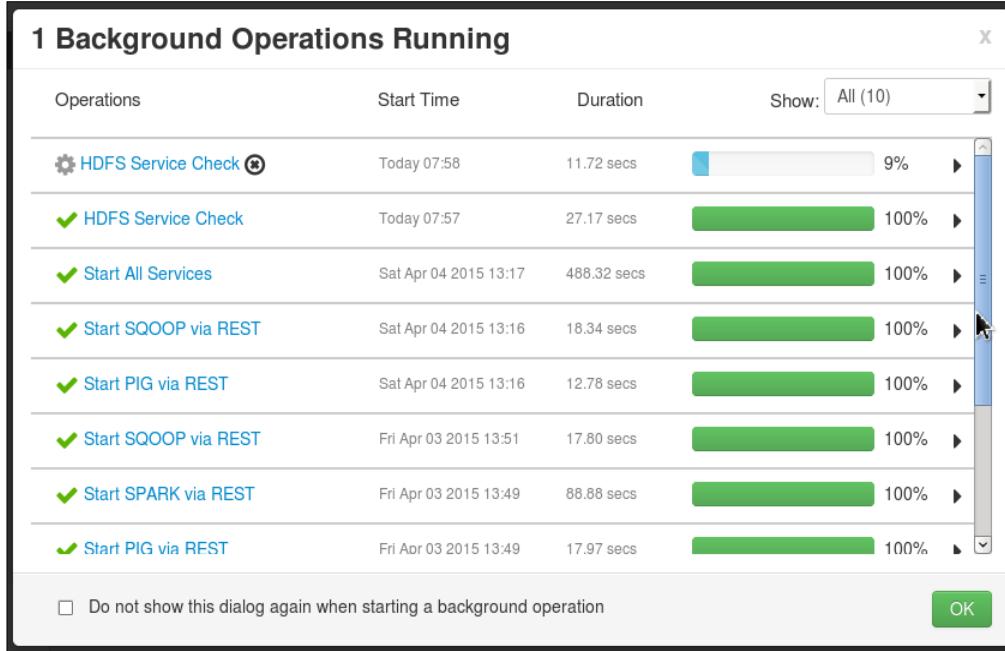
- add a service to the cluster
- add components to the service
- create configuration
- apply configuration to the cluster
- create host components
- install and start the service

For further information, go to:

- <https://cwiki.apache.org/confluence/display/AMBARI/Adding+a+New+Service+to+an+Existing+Cluster>

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Background service check from the admin menu



Apache Ambari

© Copyright IBM Corporation 2015

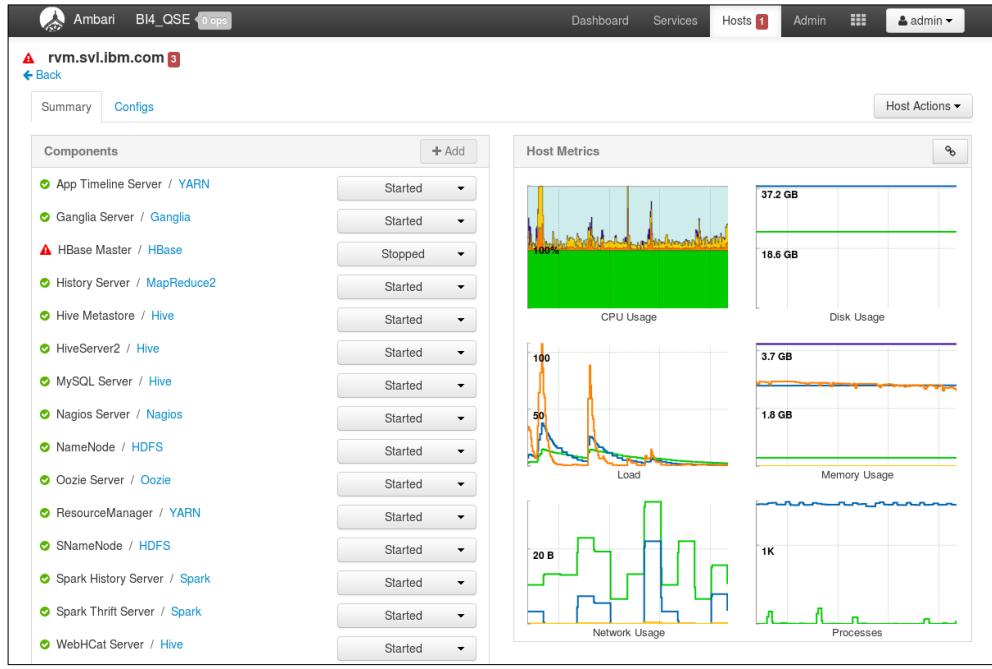
Background service check from the admin menu

When you activate adding a service or starting a service, the action takes place in background mode so that you can continue to perform other operations while the requested change runs.

You can view current background operations (blue), completed successful operations (green), and terminated failed operations (red) in the Background Service Check window.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Host metrics: example of one host (pseudo-distributed)



Apache Ambari

© Copyright IBM Corporation 2015

Host metrics: example of one host (pseudo-distributed)

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

The screenshot shows the Apache Ambari interface for a cluster named BI4_QSE. The left sidebar lists various services: HDFS, MapReduce2, YARN, Nagios, Ganglia, Hive, and HBase. HDFS and HBase are highlighted with red triangles and the number '3'. The main panel has tabs for 'Summary' and 'Configs'. The 'Summary' tab displays the status of the HBase Master as 'Stopped' with a red triangle icon. It also shows 'regionServers' with a red warning icon and 'Regions In Transition' at 0. Below this is the 'HBase Service Metrics' section, which is currently blank. To the right, the 'Alerts and Health Checks' section lists three critical alerts for the HBase Master:

- HBase Master CPU utilization CRITICAL: Data inaccessible, Status code = 0
- HBase Master process on vm.svl.ibm.com Connection refused
- >Percent RegionServers live CRITICAL: total<1>, affected<1>

Metrics are blank

Apache Ambari © Copyright IBM Corporation 2015

Non-functioning/failed services: example of HBase

A currently failed service is shown with a red-triangle on the left side of the display. This service happens to be HBase.

When you look at further details for HBase, in this case, all the metrics will show as blank.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

IBM Training

Managing Ambari



- Click the **admin** drop-down
- Select **Manage Ambari** for this display: 3 areas to manage

Creating users and groups

- This screen is for creating Ambari web users and groups, and not HDFS users and groups
- Default for Ambari Admin is **No** (for example, non-administrative user)
 - Two user roles: User and Admin. Users can view metrics, view service status and configuration, and browse job information. Admins can do all User tasks plus start and stop services, modify configurations, and run service checks.
- Initial Ambari Administrator credential is admin/admin
- HDFS users/groups are managed totally separately (see exercise)

The screenshot shows the Ambari Web GUI with the title 'Users / Create Local User'. On the left, there's a sidebar with 'Clusters' (BI4_QSE), 'Views', and 'User + Group Management' (selected). Under 'User + Group Management', there are 'Users' and 'Groups'. The main form has fields for 'Username' (biadmin), 'Type' (Local), 'Status' (Active), 'Ambari Admin' (Yes checked), 'Password', and 'Password confirmation'. At the bottom right are 'Cancel' and 'Save' buttons. The footer says '© Copyright IBM Corporation 2015'.

Creating users and groups

The individual services in Hadoop are each run under the ownership of a corresponding UNIX account. These accounts are known as service users, and these service users belong to a special UNIX group. In addition there is a special service user for running smoke tests on components during installation and on-demand using the Management Header in the Services View of the Ambari Web GUI. Any of these users and groups can be customized using the Misc tab of the Customize Services step.

If you choose to customize names, Ambari checks to see if these custom accounts already exist; if they do not exist, Ambari creates them. The default accounts are always created during installation whether or not custom accounts are specified. These default accounts are not used and can be removed after install is performed.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

The screenshot shows the Ambari interface with the title "Standard Ambari Service users and groups". On the left, there's a sidebar with "Repositories", "Service Accounts" (which is selected), and "Security". The main content area is titled "Service Users and Groups" and contains a table with the following data:

Name	Value
Proxy group for Hive and Oozie	users
HDFS User	hdfs
MapReduce User	mapred
YARN User	yarn
HBase User	hbase
Hive User	hive
HCat User	hcat
WebHCat User	hcat
Oozie User	oozie
ZooKeeper User	zookeeper
Ganglia User	nobody
Nagios User	nagios
Nagios Group	nagios
Smoke Test User	ambari-qa
Spark User	spark
Hadoop Group	hadoop
Sqoop User	sqoop
Skip group modifications during install	false

At the bottom, it says "Apache Ambari" and "© Copyright IBM Corporation 2015".

Standard Ambari Service users and groups

The standard set of services are:

- HDFS
- MapReduce
- Hive
- HCat
- WebHCat
- Oozie
- HBase
- ZooKeeper
- Ganglia
- Nagios
- Smoke Test

The set of default users that roughly correspond to these services are: hdfs, mapred, hive, hcat, hcat, oozie, hbase, zookeeper, nobody, Nagios, ambari-qa.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

These service users and groups as shown in the slide and are in the following table from the Ambari documentation:

- <https://ambari.apache.org/1.2.5/installing-hadoop-using-ambari/content/ambari-users.html>

Service	Component	Default User Account
HDFS	NameNode SecondaryNameNode DataNode	hdfs
MapReduce	JobTracker HistoryServer TaskTracker	mapred
Hive	Hive Metastore HiveServer2	hive
HCat	HCatalog Server	hcat
WebHCat	WebHCat Server	hcat
Oozie	Oozie Server	oozie
HBase	MasterServer RegionServer	hbase
ZooKeeper	ZooKeeper	zookeeper
Ganglia	Ganglia Server Ganglia Collectors	nobody
Nagios	Nagios Server	nagios ^a
Smoke Test ^b	All	ambari-qa

^a If you plan to use an existing user account named "nagios", that "nagios" account must be in a group named "Nagios." If you customize this account, that account will be created and put in a group "nagios."

^b The Smoke Test user performs smoke tests against cluster services as part of the install process. It also can perform these on-demand from the Ambari Web GUI.

You will find these users in **/etc/hosts** and they have **home directories** (`/home/...`) in the Linux filesystem.

For further information: <https://ambari.apache.org/x.y.z/installing-hadoop-using-ambari/content/ambari-users.html> (note: substitute x.y.z with the relevant version of Ambari). The most current releases of Ambari are 1.7.0 (installed in ODP 4.0.0) and 2.0.0/2.0.1 (expected in upcoming releases of ODP when these versions of Ambari are included in the Open Data Platform [ODP] initiative).

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Managing hosts in a cluster

- Ambari provides the following actions using the **Hosts** tab:
 - Working with Hosts
 - Determining Host Status
 - Filtering the Hosts List
 - Performing Host-Level Actions
 - Viewing Components on a Host
 - Decommissioning Masters and Slaves
 - Deleting a Host from a Cluster
 - Setting Maintenance Mode
 - Adding Hosts to a Cluster

Running Ambari from the command line

- Ambari allows commands to be run at the command line
 - To read settings from Ambari (such as hosts in the cluster)


```
curl -i -uadmin:admin http://rvm.svl.ibm.com:8080/api/v1/hosts
```
 - To deploy services across a cluster using a script
 - To undeploy services which are not being used

(Note that generally services should be stopped manually before removing them. Sometimes stopping from Ambari might not stop some of the sub-components, so sub-components may also need to be stopped.)

```
curl -u admin:admin -H "X-Requested-By: ambari" -X DELETE
      http://localhost:8080/api/v1/clusters/BI4_QSVservices/FLUME
curl -u admin:admin -H "X-Requested-By: ambari" -X DELETE
      http://localhost:8080/api/v1/clusters/BI4_QSVservices/SLIDER
curl -u admin:admin -H "X-Requested-By: ambari" -X DELETE
      http://localhost:8080/api/v1/clusters/BI4_QSVservices/SOLR
```

- An Ambari shell (with prompt) is available
 - <https://cwiki.apache.org/confluence/display/AMBARI/Ambari+Shell>

Running Ambari from the command line

This is an example interaction with command line interface; the hosts are in a Hadoop Cluster (with results returned in JSON format)

```
[root@rvm ~]# curl -i -uadmin:admin http://rvm.svl.ibm.com:8080/api/v1/hosts
HTTP/1.1 200 OK
Set-Cookie: AMBARISESSIONID=1n8t0nb6perytxj09ju3las9z; Path=/
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/plain
Content-Length: 265
Server: Jetty(7.6.7.v20120910)
{
  "href" : "http://rvm.svl.ibm.com:8080/api/v1/hosts",
  "items" : [
    {
      "href" : "http://rvm.svl.ibm.com:8080/api/v1/hosts/rvm.svl.ibm.com",
      "Hosts" : {
        "cluster_name" : "BI4_QSE",
        "host_name" : "rvm.svl.ibm.com"
      }
    }
  ]
}
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Starting / Restarting the Ambari Server

```
[root@rvm ~]# ambari-server restart
Using python  /usr/bin/python2.6
Restarting ambari-server
Using python  /usr/bin/python2.6
Stopping ambari-server
Ambari Server stopped
Using python  /usr/bin/python2.6
Starting ambari-server
Ambari Server running with 'root' privileges.
Organizing resource files at /var/lib/ambari-server/resources...
Server PID at: /var/run/ambari-server/ambari-server.pid
Server out at: /var/log/ambari-server/ambari-server.out
Server log at: /var/log/ambari-server/ambari-server.log
Waiting for server start.....
Ambari Server 'start' completed successfully.
[root@rvm ~]#
```

Note also that Ambari has a Python Shell, but the Python Shell is still under development. Reference:

- <https://cwiki.apache.org/confluence/display/AMBARI/Ambari+python+Shell>

Ambari terminology

- Service
- Component
- Host/Node
- Node Component
- Operation
- Task
- Stage
- Action
- Stage Plan
- Manifest
- Role

Ambari terminology

The following definitions can be found at <http://ambari.apache.org>.

Service: Service refers to services in the Hadoop stack. HDFS, HBase, and Pig are examples of services. A service may have multiple components (for example, HDFS has NameNode, Secondary NameNode, DataNode, etc.). A service can just be a client library (for example, Pig does not have any daemon services, but just has a client library).

Component: A service consists of one or more components. For example, HDFS has 3 components: NameNode, DataNode and Secondary NameNode. Components may be optional. A component may span multiple nodes (for example, DataNode instances on multiple nodes).

Node/Host: Node refers to a machine in the cluster. Node and host are used interchangeably in this document.

Node-Component: Node-component refers to an instance of a component on a particular node. For example, a particular DataNode instance on a particular node is a node-component.

Operation: An operation refers to a set of changes or actions performed on a cluster to satisfy a user request or to achieve a desirable state change in the cluster. For example, starting of a service is an operation and running a smoke test is an operation.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

If a user requests to add a new service to the cluster and that includes running a smoke test as well, then the entire set of actions to meet the user request will constitute an operation. An operation can consist of multiple "actions" that are ordered (see below).

Task: Task is the unit of work that is sent to a node to execute. A task is the work that node has to carry out as part of an action. For example, an "action" can consist of installing a DataNode on Node n1 and installing a DataNode and a secondary NameNode on Node n2. In this case, the "task" for n1 will be to install a DataNode and the "tasks" for n2 will be to install both a DataNode and a secondary NameNode.

Stage: A stage refers to a set of tasks that are required to complete an operation and are independent of each other; all tasks in the same stage can be run across different nodes in parallel.

Action: An 'action' consists of a task or tasks on a machine or a group of machines. Each action is tracked by an action id and nodes report the status at least at the granularity of the action. An action can be considered a stage under execution. In this document a stage and an action have one-to-one correspondence unless specified otherwise. An action id will be a bijection of request-id, stage-id.

Stage Plan: An operation typically consists of multiple tasks on various machines and they usually have dependencies requiring them to run in a particular order. Some tasks are required to complete before others can be scheduled. Therefore, the tasks required for an operation can be divided in various stages where each stage must be completed before the next stage, but all the tasks in the same stage can be scheduled in parallel across different nodes.

Manifest: Manifest refers to the definition of a task which is sent to a node for execution. The manifest must completely define the task and must be serializable. Manifest can also be persisted on disk for recovery or record.

Role: A role maps to either a component (for example, NameNode, DataNode) or an action (for example, HDFS rebalancing, HBase smoke test, other admin commands, etc.)

Unit summary

- Understand the purpose of Apache Ambari in the Open Data Platform
- Understand the overall architecture of Ambari, and Ambari's relation to other services and components of a Hadoop cluster
- List the functions of the main components of Ambari
- Explain how to start and stop services from the Ambari Web Console

Exercise 1

Managing Hadoop clusters with Apache Ambari

Apache Ambari

© Copyright IBM Corporation 2015

Exercise 1: Managing Hadoop clusters with Apache Ambari

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1:

Managing Hadoop clusters with Apache Ambari

Purpose:

You will explore the Apache Ambari web console and perform basic starting and stopping of services, giving you experience in using Apache Ambari to manage your Hadoop cluster.

VM Hostname: <http://ibmclass.localdomain>

User/Password: **biadmin / biadmin**
root/dalvm3

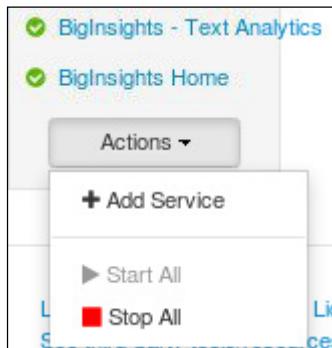
Task 1. Start the Apache Ambari web console and perform basic start/stop of services.

The major reference for Apache Ambari can be found at <http://ambari.apache.org/>.

There are various shells available for Ambari, but not all are fully developed or available at time of developing this course; those shells are not covered in this exercise. These shells include: Ambari shell and Python Shell & Client.

Note: Cluster administration is not normally done by end users, but knowledge of Apache Ambari is extremely useful for all users as it provides concepts needed for both administration and use of the cluster.

1. Connect to and login to your lab environment with the user **biadmin** and the password **biadmin**.
2. Launch **Firefox**, and navigate to the **Ambari** login page, <http://localhost:8080>, logging in as **admin/admin**.
Note which services are currently running from the left pane.
3. At the bottom of the **Services** pane, click **Actions**, and then:
 - if the services are started, click **Stop All**.
 - If the services are stopped, click **Start All**.

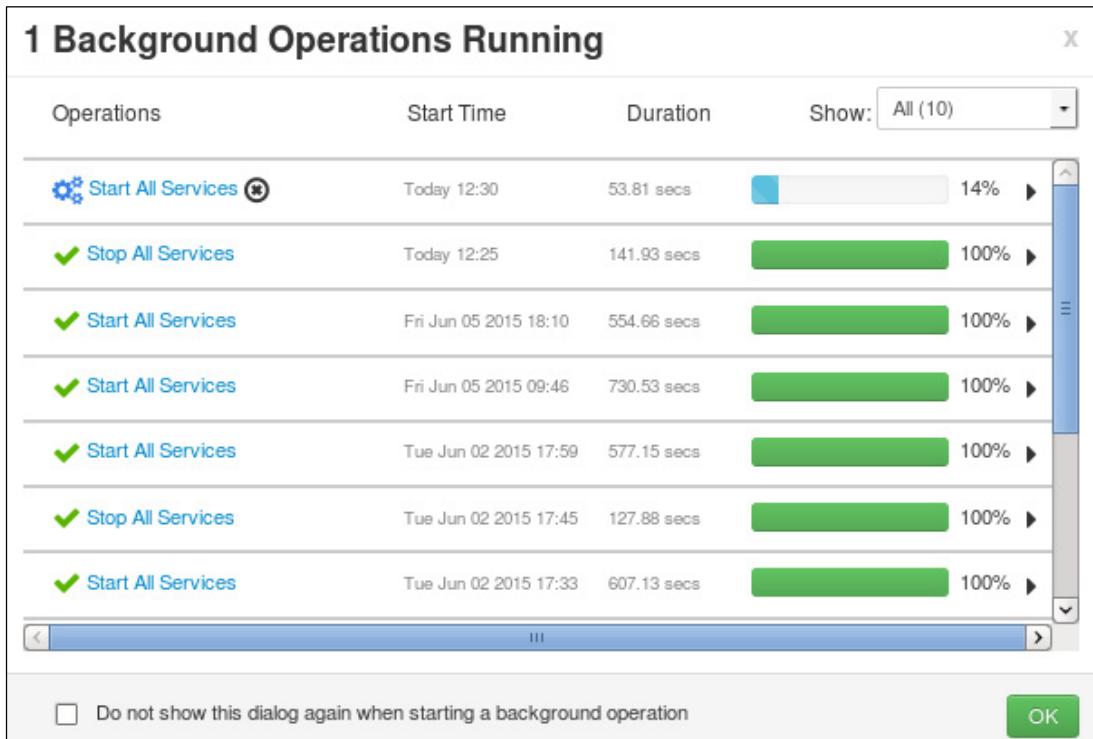


This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

4. When asked for confirmation, click **OK**.



Starting or stopping the services can sometimes take several minutes.



Operations which are in progress are shown in blue, complete and successful in green, and failed in red.

If you stopped the services in step 3, from the **Actions** menu, click **Start All**, and then click **OK** when prompted for confirmation.

5. To see what services can be added, from the **Actions** menu, click **Add Service**. The services shown will include some or all of the following; your listing will probably differ if you have a later version of some or all of these components.

Service	Version	Description
HDFS	2.6.0	Apache Hadoop Distributed File System
YARN + MapReduce2	2.6.0	Apache Hadoop NextGen MapReduce (YARN)
Nagios	3.5.1	Nagios Monitoring and Alerting system
Ganglia	3.1.7	Ganglia Metrics Collection system (RRDTool will be installed too)
Hive	0.14.0	Data warehouse system for ad-hoc queries & analysis of large datasets and table & storage management service
HBase	0.98.8	Non-relational distributed database and centralized service for configuration management & synchronization
Pig	0.14.0	Scripting platform for analyzing large datasets
Sqoop	1.4.5	Tool for transferring bulk data between Apache Hadoop and structured data stores such as relational databases
Oozie	4.1.0	System for workflow coordination and execution of Apache Hadoop jobs. This also do the installation of the optional Oozie Web Console which relies the ExtJS Library
ZooKeeper	3.4.6	Centralized service which provides highly reliable distributed coordination
Knox	0.5.0	Provides a single point of authentication and access for Apache Hadoop services in a cluster
Slider	0.60.0	A framework for deploying, managing and monitoring existing distributed applications on YARN
Solr	4.10.3	Solr is the popular, blazing fast open source enterprise search platform from the Apache Lucene project
Spark	1.2.1	Apache Spark is a fast and general engine for large-scale data processing
Flume	1.5.2	A distributed service for collecting, aggregating, and moving large amounts of streaming data into HDFS

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Services that are installed and available are shown with check-marks. Services without a check-mark (such as Big SQL in this instance) are not yet installed or available:

Add Service Wizard

Service	Version	Description
<input checked="" type="checkbox"/> HDFS	2.6.0	Apache Hadoop Distributed File System
<input checked="" type="checkbox"/> YARN + MapReduce2	2.6.0	Apache Hadoop NextGen MapReduce (YARN)
<input checked="" type="checkbox"/> Nagios	3.5.1	Nagios Monitoring and Alerting system
<input checked="" type="checkbox"/> Ganglia	3.1.7	Ganglia Metrics Collection system (RRDTool will be installed too)
<input checked="" type="checkbox"/> Hive	0.14.0	Data warehouse system for ad-hoc queries & analysis of large datasets and table & storage management service
<input checked="" type="checkbox"/> HBase	0.98.8	Non-relational distributed database and centralized service for configuration management & synchronization
<input checked="" type="checkbox"/> Pig	0.14.0	Scripting platform for analyzing large datasets
<input checked="" type="checkbox"/> Sqoop	1.4.5	Tool for transferring bulk data between Apache Hadoop and structured data stores such as relational databases
<input checked="" type="checkbox"/> Oozie	4.1.0	System for workflow coordination and execution of Apache Hadoop jobs. This also do the installation of the optional Oozie Web Console which relies the ExtJS Library.
<input checked="" type="checkbox"/> ZooKeeper	3.4.6	Centralized service which provides highly reliable distributed coordination
<input checked="" type="checkbox"/> Flume	1.5.2	A distributed service for collecting, aggregating, and moving large amounts of streaming data into HDFS
<input checked="" type="checkbox"/> BigInsights - Big R	3.17	The In-hadoop Analytics with R
<input checked="" type="checkbox"/> BigInsights - BigSheets	4.48	BigSheets is an intuitive spreadsheet-like tool, to create analytic queries without any previous programming experience
<input type="checkbox"/> BigInsights - Big SQL	4.0	SQL on Hadoop
<input checked="" type="checkbox"/> Knox	0.5.0	Provides a single point of authentication and access for Apache Hadoop services in a cluster

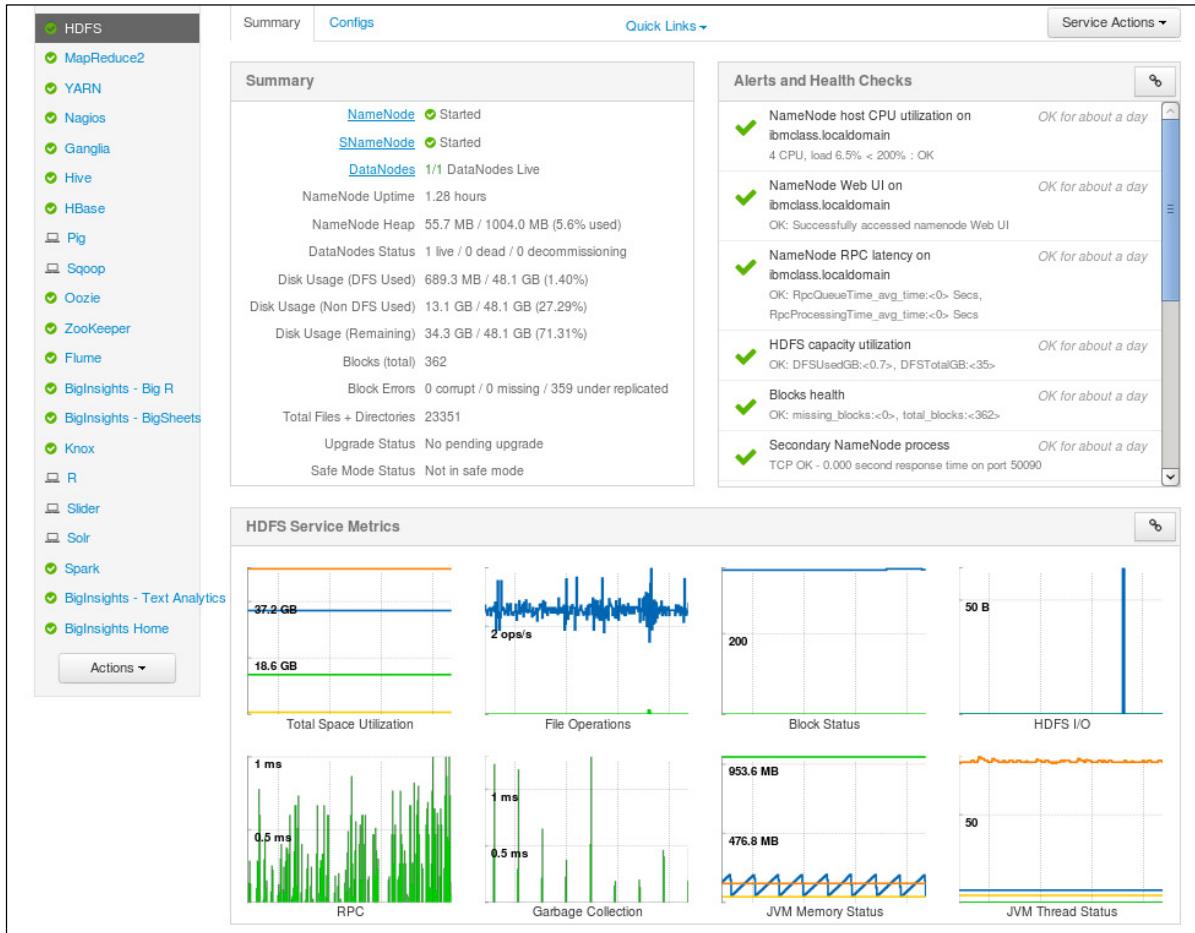


Important: this listing shows the version of each component, whether Open Data Platform or the IBM BigInsights Add-in that you have and whether it is an installed service. You may need this information later for IBM service support.

6. Close the **Add Services** window.
 7. When all of the services have been started, look at center of the page to see the high level information available there.
- You will explore the various individual services that are listed, to better understand what is available.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

8. Click the **HDFS** service to expose the relevant information and metrics:



9. Explore other services on the left panel of the main web page.

Task 2. Explore other aspects of the Ambari web server.

1. Navigate and explore other aspects of the Ambari web server interface using the earlier content pages and slides in this unit to guide you.
2. Close all open windows.

Results:

You explored the Apache Ambari web console and performed basic starting and stopping of services, gaining experience in using Apache Ambari to manage your Hadoop cluster.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit 8 Hadoop and HDFS

IBM Training



Hadoop and HDFS

IBM BigInsights v4.0

© Copyright IBM Corporation 2015
Course materials may not be reproduced in whole or in part without the written permission of IBM.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit objectives

- Understand the basic need for a big data strategy in terms of parallel reading of large data files and internode network speed in a cluster
- Describe the nature of the Hadoop Distributed File System (HDFS)
- Explain the function of the NameNode and DataNodes in an Hadoop cluster
- Explain how files are stored and blocks ("splits") are replicated

Agenda

- Why Hadoop?
 - Comparison with RDBMS
- Hadoop architecture
 - MapReduce
 - HDFS
 - Hadoop Common
- The origin and history of HDFS
 - How files are stored; replication of blocks
 - Name Node
 - Secondary Name Node
 - Fault tolerance



Hadoop and HDFS

© Copyright IBM Corporation 2015

Agenda

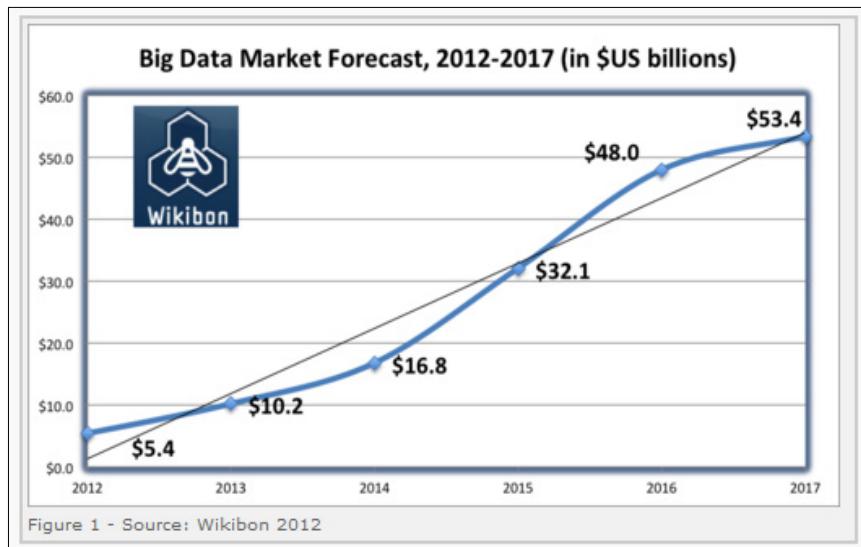
Here's an overview of what we will cover in this unit.

This unit will explain the underlying technologies that are important to solving the big data challenge and tell about BigInsights. First, you will have an overview, and then you will dive more deeply into BigInsights itself, and explore how it fits into a broader IT environment that could involve data warehouses, relational DBMSs, streaming engines, etc.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

The importance of Hadoop

- "We believe that more than half of the world's data will be stored in Apache Hadoop within five years" - Hortonworks



Hadoop and HDFS

© Copyright IBM Corporation 2015

The importance of Hadoop

There has been much buzz about Hadoop and big data in the marketplace over the past year, with Wikibon predicting a \$50B marketplace by 2017. Hortonworks says on their web site that they believe that half of the world's data will be stored in Hadoop within the next 5 years. With so much at stake, there are a lot of vendors looking for some of the action and the big database vendors are not standing still.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Hardware improvements through the years

- CPU speeds:
 - 1990: 44 MIPS at 40 MHz
 - 2010: 147,600 MIPS at 3.3 GHz
- RAM memory
 - 1990: 640K conventional memory
(256K extended memory recommended)
 - 2010: 8-32GB (and more)
- Disk capacity
 - 1990: 20MB
 - 2010: 1TB
- Disk latency (speed of reads and writes) -
not much improvement in last 7-10 years,
currently around 70-80 MB/sec

How long does it take to read 1TB of data (at 80 MB/sec)?

- 1 disk - 3.4 hrs
- 10 disks - 20 min
- 100 disks - 2 min
- 1000 disks - 12 sec

Hardware improvements through the years

Before exploring Hadoop, you will review why Hadoop technology is so important.

Moore's law has been true for a long time, but no matter how many more transistors are added to CPUs, and how powerful they become, it is **disk latency** where the bottleneck is. This chart makes the point: scaling up (more powerful computers with power CPUs) is not the answer to all problems since disk latency is the main issue. Scaling out (cluster of computers) is a better approach.

One definition of a commodity server is "a piece of fairly standard hardware that can be purchased at retail, to have any particular software installed on it".

A typical Hadoop cluster node on Intel hardware in 2015:

- two quad core CPUs
- 12 GB to 24 GB memory
- four to six disk drives of 2 terabyte (TB) capacity

Incidentally, the second slowest component in a cluster of computers is the inter-node network connection. You will find that this leads to the importance of where to place the data that is to be read.

This topic is well discussed on the internet. As an example, try doing a Google search of **best hardware configuration for hadoop**. You should note that the hardware required for management nodes is considerably higher. Here you are primarily presented with the hardware requirements for worker nodes, aka DataNodes.

Reference:

- http://docs.hortonworks.com/HDP2Alpha/index.htm#Hardware_Recommendations_for_Hadoop.htm

What hardware is not used for Hadoop

- RAID
 - Not suitable for the data in a large cluster
 - Wanted: Just-a-Bunch-Of-Disks (JBOD)
- Linux Logical Volume Manager (LVM)
 - HDFS and GPFS are already abstractions that run on top of disk filesystems
 - LVM is an abstraction that can obscure the real disk
- Solid-state disk (SSD)
 - Low latency of SSD is not useful for streaming file data
 - Low storage capacity for high cost (currently not commodity hardware)

RAID is often used on Master Nodes
(but never Data Nodes)
as part of fault tolerance mechanisms

What hardware is not used for Hadoop

There is the possibility of software RAID in future versions of Hadoop:

- <http://wiki.apache.org/hadoop/HDFS-RAID>

Discussion of why not to use RAID:

- <http://hortonworks.com/blog/why-not-raid-0-its-about-time-and-snowflakes>
- <http://hortonworks.com/blog/proper-care-and-feeding-of-drives-in-a-hadoop-cluster-a-conversation-with-stackiqs-dr-bruno/>

Parallel data processing is the answer

- There has been:
 - GRID computing: spreads processing load
 - distributed workload: hard to manage applications, overhead on developer
 - parallel databases: DB2 DPF, Teradata, Netezza, etc. (distribute the data)
- Challenges:
 - heterogeneity
 - openness
 - security
 - scalability
 - concurrency
 - fault tolerance
 - transparency



Distributed computing: Multiple computers appear as one super computer, communicate with each other by message passing, and operate together to achieve a common goal

Parallel data processing is the answer

Further comments:

- With GRID computing, the idea is to increase processing power of multiple computers and bring data to place where processing capacity is available.
- Distributed workload is bring processing to data. This is great, but writing applications that do this is very difficult. You need to worry how to exchange data between the different nodes, you need to think about what happens if one of these nodes goes down, you need to decide where to put the data, decide to which node transmit the data, and figure out when all nodes finish, how to send the data to some central place. All of this is very challenging. You could spend a lot of time coding on passing the data, rather than dealing with the problem itself.
- Parallel databases are in use today, but they are generally for structured relational data.
- Hadoop, is the answer to parallel data processing without having the issues of GRID, distributed workload or parallel databases.

What is Hadoop?



- Apache open source software framework for reliable, scalable, distributed computing over massive amount of data:
 - hides underlying system details and complexities from user
 - developed in Java
- Consists of 3 sub projects:
 - MapReduce
 - Hadoop Distributed File System (HDFS)
 - Hadoop Common
- Supported by many Apache/Hadoop-related projects:
 - HBase, Zookeeper, Avro, etc.
- Meant for heterogeneous commodity hardware.



The Apache Software Foundation

What is Hadoop?

Hadoop is an open source project of the Apache Foundation.

It is a framework written in Java originally developed by Doug Cutting who named it after his son's toy elephant.

Hadoop uses Google's MapReduce and Google File System (GFS) technologies as its foundation.

It is optimized to handle massive amounts of data which could be structured, unstructured or semi-structured, and uses commodity hardware (relatively inexpensive computers).

This massive parallel processing is done with great performance. However, it is a batch operation handling massive amounts of data, so the response time is not immediate. As of Hadoop version 0.20.2, updates are not possible, but appends were made possible starting in version 0.21.

What is the value of a system if the information it stores or retrieves is not consistent?

Hadoop replicates its data across different computers, so that if one goes down, the data is processed on one of the replicated computers.

You may be familiar with OLTP (Online Transactional processing) workloads where data is randomly accessed on structured data like a relational database, such as when you access your bank account.

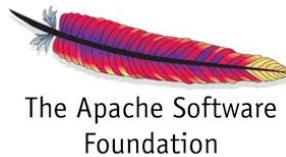
You may also be familiar with OLAP (Online Analytical processing) or DSS (Decision Support Systems) workloads where data is sequentially access on structured data, like a relational database, to generate reports that provide business intelligence.

You may not be that familiar with the concept of big data. Big data is a term used to describe large collections of data (also known as datasets) that may be unstructured, and grow so large and quickly that is difficult to manage with regular database or statistics tools.

Hadoop is not used for OLTP nor OLAP, but is used for big data, and it complements these two to manage data. Hadoop is not a replacement for a RDBMS.

Hadoop open source projects

- Hadoop is supplemented by an extensive ecosystem of open source projects.



Hadoop open source projects

The Open Data Platform (www.opendataplatform.org):

- The ODP Core will initially focus on Apache Hadoop (inclusive of HDFS, YARN, and MapReduce) and Apache Ambari. Once the ODP members and processes are well established, the scope of the ODP Core may expand to include other open source projects.

Apache Ambari aims to make Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters. Ambari provides an intuitive, easy-to-use Hadoop management web UI backed by its RESTful APIs.

Reference summary from <http://hadoop.apache.org>:

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

The project includes these modules:

- Hadoop Common: The common utilities that support the other Hadoop modules.
- Hadoop Distributed File System (HDFS™): A distributed file system that provides high throughput access to application data.
- Hadoop YARN: A framework for job scheduling and cluster resource management.
- Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.'
- Ambari™: A web-based tool for provisioning, managing, and monitoring Hadoop clusters. It also provides a dashboard for viewing cluster health and ability to view MapReduce, Pig and Hive applications visually.
- Avro™: A data serialization system.
- Cassandra™: A scalable multi-master database with no single points of failure.
- Chukwa™: A data collection system for managing large distributed systems.
HBase™: A scalable, distributed database that supports structured data storage for large tables.
- Hive™: A data warehouse infrastructure that provides data summarization and ad hoc querying.
- Mahout™: A Scalable machine learning and data mining library.
- Pig™: A high-level data-flow language and execution framework for parallel computation.
- Spark™: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- Tez™: A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases.
- ZooKeeper™: A high-performance coordination service for distributed applications.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Advantages and disadvantages of Hadoop

- Hadoop is good for:
 - processing massive amounts of data through parallelism
 - handling a variety of data (structured, unstructured, semi-structured)
 - using inexpensive commodity hardware
- Hadoop is not good for:
 - processing transactions (random access)
 - when work cannot be parallelized
 - low latency data access
 - processing lots of small files
 - intensive calculations with small amounts of data

Advantages and disadvantages of Hadoop

Hadoop is cannot resolve all data-related problems. It is being designed to handle big data. Hadoop works better when handling one single huge file rather than many small files. Hadoop complements existing RDBMS technology.

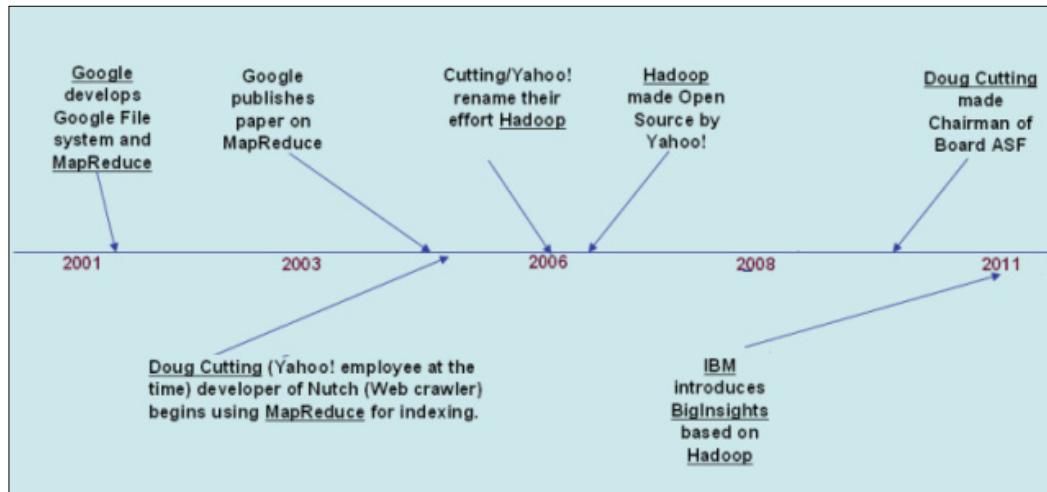
Wikipedia: **Low latency** allows human-unnoticeable delays between an input being processed and the corresponding output providing real time characteristics. This can be especially important for internet connections utilizing services such as online gaming and VOIP.

Hadoop is not good for low latency data access. In practice you can replace latency with delay. So low latency means negligible delay in processing. Low latency data access means negligible delay accessing data. Hadoop is not designed for low latency.

Hadoop works best with very large files. The larger the file, the less time Hadoop spends seeking for the next data location on disk and the more time Hadoop runs at the limit of the bandwidth of your disks. Seekers are generally expensive operations that are useful when you only need to analyze a small subset of your dataset. Since Hadoop is designed to run over your entire dataset, it is best to minimize seekers by using large files.

Hadoop is good for applications requiring high throughput of data: Clustered machines can read data in parallel for high throughput.

Time line for Hadoop



Time line for Hadoop

Review this slide for the history of Hadoop/MapReduce technology.

Google published information on the GFS (Google File System): Ghemawat, S., Gobioff, H., & Leung, S.-T. (2002). The Google File System. Retrieved from <http://static.googleusercontent.com/media/research.google.com/en/us/archive/gfs-sosp2003.pdf>

The original MapReduce paper: Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. Retrieved from <http://static.googleusercontent.com/media/research.google.com/en/us/archive/mapreduce-osdi04.pdf>

Doug Cutting was developing Lucene (Text index for documents, subproject of Nutch), and Nutch (Web index for documents). He could not get Nutch to scale. Then he saw Google's papers on GFS and MapReduce and created Hadoop (open source version of the info from those papers). Hadoop allowed Nutch to scale.

Timeline:

- 2003: Google launches project Nutch to handle billions of searches and indexing millions of web pages.
- Oct 2003: Google releases papers with GFS (Google File System)
- Dec 2004: Google releases papers with MapReduce
- 2005: Nutch used GFS and MapReduce to perform operations
- 2006: Yahoo! created Hadoop based on GFS and MapReduce (with Doug Cutting and team)
- 2007: Yahoo started using Hadoop on a 1000 node cluster
- Jan 2008: Apache took over Hadoop
- Jul 2008: Tested a 4000 node cluster with Hadoop successfully
- 2009: Hadoop successfully sorted a petabyte of data in less than 17 hours to handle billions of searches and indexing millions of web pages.
- Dec 2011: Hadoop releases version 1.0

For later releases, and the release numbering structure, refer to:

- <http://wiki.apache.org/hadoop/Roadmap>

Hadoop: major components

- Hadoop Distributed File System (HDFS)
 - where Hadoop stores data
 - a file system that spans all the nodes in a Hadoop cluster
 - links together the file systems on many local nodes to make them into one large file system that spans all the data nodes of the cluster
- MapReduce framework
- How Hadoop understands and assigns work to the nodes (machines)
- Evolving: MR v1, MR v2, etc.
- Morphed into YARN and other processing frameworks

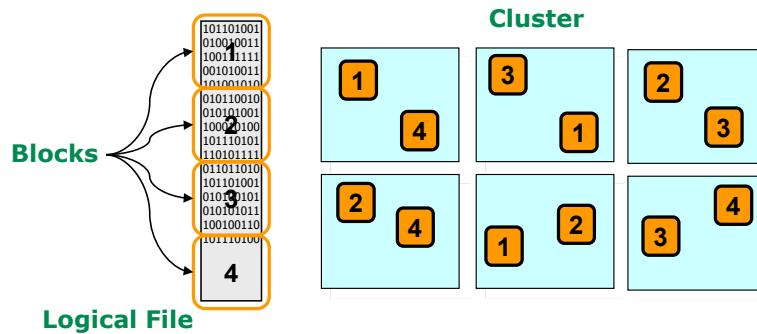
Hadoop: major components

There are two aspects of Hadoop that are important to understand:

1. MapReduce is a software framework introduced by Google to support distributed computing on large data sets of clusters of computers.
2. The Hadoop Distributed File System (HDFS) is where Hadoop stores its data. This file system spans all the nodes in a cluster. Effectively, HDFS links together the data that resides on many local nodes, making the data part of one big file system. You can use other file systems with Hadoop, but HDFS is quite common.

Brief introduction to HDFS and MapReduce

- Driving principles
 - data is stored across the entire cluster
 - programs are brought to the data, not the data to the program
- Data is stored across the entire cluster (the DFS)
 - the entire cluster participates in the file system
 - blocks of a single file are distributed across the cluster
 - a given block is typically replicated as well for resiliency



Hadoop and HDFS

© Copyright IBM Corporation 2015

Brief introduction to HDFS and MapReduce

The driving principle of MapReduce is a simple one: spread the data out across a huge cluster of machines and then, rather than bringing the data to your programs as you do in a traditional programming, write your program in a specific way that allows the program to be moved to the data. Thus, the entire cluster is made available in both reading the data as well as processing the data.

The Distributed File System (DFS) is at the heart of MapReduce. It is responsible for spreading data across the cluster, by making the entire cluster look like one giant file system. When a file is written to the cluster, blocks of the file are spread out and replicated across the whole cluster (in the diagram, notice that every block of the file is replicated to three different machines).

Adding more nodes to the cluster instantly adds capacity to the file system and automatically increases the available processing power and parallelism.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Hadoop Distributed File System (HDFS) principles

- Distributed, scalable, fault tolerant, high throughput
- Data access through MapReduce
- Files split into **blocks** (aka **splits**)
- **3 replicas** for each piece of data by default
- Can **create, delete, and copy**, but cannot update
- Designed for **streaming reads**, not random access
- **Data locality** is an important concept: processing data on or near the physical storage to decrease transmission of data



Hadoop and HDFS

© Copyright IBM Corporation 2015

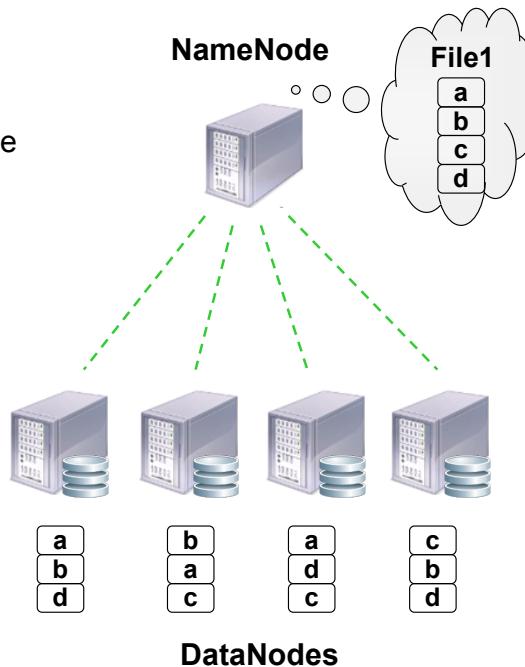
Hadoop Distributed File System (HDFS) principles

This slide lists the important principles behind the Hadoop Distributed File System (HDFS).

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

HDFS architecture

- Master/Slave architecture
- Master: **NameNode**
 - manages the file system namespace and metadata
 - FsImage
 - Edits Log
 - regulates client access to files
- Slave: **DataNode**
 - many per cluster
 - manages storage attached to the nodes
 - periodically reports status to NameNode



Hadoop and HDFS

© Copyright IBM Corporation 2015

HDFS architecture

Important points still to be discussed:

- Secondary NameNode
- Import checkpoint
- Rebalancing
- SafeMode
- Recovery Mode

The entire file system namespace, including the mapping of blocks to files and file system properties, is stored in a file called the **FsImage**. The FsImage is stored as a file in the NameNode's local file system. It contains the metadata on disk (not exact copy of what is in RAM, but a checkpoint copy).

The NameNode uses a transaction log called the **EditLog** (or **Edits Log**) to persistently record every change that occurs to file system metadata, synchronizes with metadata in RAM after each write.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

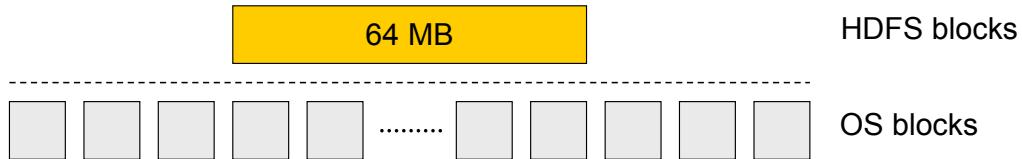
The **NameNode** can be a potential single point of failure (this has been resolved in later releases of HDFS with Secondary NameNode, various forms of high availability, and in Hadoop v2 with NameNode federation and high availability as out-of-the-box options).

- Use better quality hardware for all management nodes, and in particular do not use inexpensive commodity hardware for the NameNode.
- Mitigate by backing up to other storage.

In case of power failure on NameNode, recover is performed using the FsImage and the EditLog.

HDFS blocks

- HDFS is designed to support very large files
- Each file is split into blocks: Hadoop default is 128 MB
- Blocks reside on different physical DataNodes
- Behind the scenes, each HDFS block is supported by multiple operating system blocks



- If a file or a chunk of the file is smaller than the block size, only the needed space is used. For example, a 210MB file is split as:



HDFS blocks

Data in a Hadoop cluster is broken down into smaller pieces (called blocks) and distributed throughout the cluster. In this way, the map and reduce functions can be executed on smaller subsets of your larger data sets, and this provides the scalability that is needed for big data processing.

- <http://www-01.ibm.com/software/data/infosphere/hadoop/hdfs/>

In earlier versions of Hadoop/HDFS, the default blocksize was often quoted as 64 MB, but the current default setting for Hadoop/HDFS is noted in <http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml>

- `dfs.blocksize = 134217728`
The default block size for new files, in bytes. You can use the following suffix (case insensitive): k(kilo), m(mega), g(giga), t(tera), p(peta), e(exa) to specify the size (such as 128k, 512m, 1g, etc.), or you can provide the complete size in bytes (such as 134217728 for 128 MB).

It should be noted that Linux itself has both a logical block size (typically 4 KB) and a physical or hardware block size (typically 512 bytes).

Linux filesystems:

- For ext2 or ext3, the situation is relatively simple: each file occupies a certain number of blocks. All blocks on a given filesystem have the same size, usually one of 1024, 2048 or 4096 bytes.
- What is the physical blocksize?

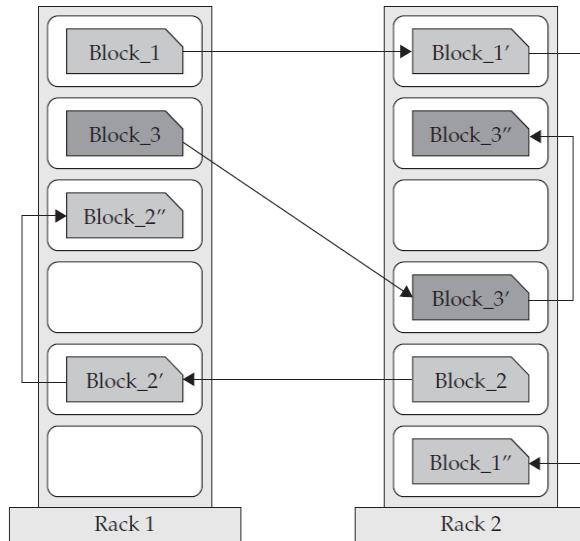
```
[user@vm Desktop]$ lsblk -o NAME,PHY-SEC
NAME      PHY-SEC
sr0        512
sda        512
└─sda1     512
└─sda2     512
└─sda3     512
```

- What is the logical blocksize for these file systems (such as /dev/sda1)? This has to be run as root.

```
[root@vm ~]# blockdev --getbsz /dev/sda1
1024
```

HDFS replication of blocks

- Blocks of data are replicated to multiple nodes
 - behavior is controlled by **replication factor**, configurable per file
 - default is **3 replicas**
- Approach:
 - first replica goes on any node in the cluster
 - second replica on a node in a different rack
 - third replica on a different node in the second rack
- **The approach cuts inter-rack network bandwidth, which improves write performance**



Hadoop and HDFS

© Copyright IBM Corporation 2015

HDFS replication of blocks

To ensure reliability and performance, the placement of replicas is critical to HDFS. HDFS can be distinguished from most other distributed file systems by replica placement. This feature requires tuning and experience. Improving data reliability, availability, and network bandwidth utilization are the purpose of a rack-aware replica placement policy. The current implementation for the replica placement policy is a first effort in this direction. The short-term goals of implementing this policy are to validate it on production systems, learn more about its behavior, and build a foundation to test and research more sophisticated policies.

Large HDFS instances run on a cluster of computers that commonly spread across many racks. Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.

The NameNode determines the rack id each DataNode belongs to via the process outlined in block awareness (http://hadoop.apache.org/docs/r0.17.1/hdfs_user_guide.html#Rack+Awareness). A simple but non-optimal policy is to place replicas on unique racks. This prevents losing data when an entire rack fails and allows use of bandwidth from multiple racks when reading data. This policy evenly distributes replicas in the cluster which makes it easy to balance load on component failure, however this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

For the common case, when the replication factor is three, HDFS's placement policy is to put one replica on one node in the local rack, another on a different node in the local rack, and the last on a different node in a different rack. This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure; this policy does not impact data reliability and availability guarantees. However, it does reduce the aggregate network bandwidth used when reading data since a block is placed in only two unique racks rather than three. With this policy, the replicas of a file do not evenly distribute across the racks. One third of replicas are on one node, two thirds of replicas are on one rack, and the other third are evenly distributed across the remaining racks. This policy improves write performance without compromising data reliability or read performance.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Setting rack network topology (Rack Awareness)

- Defined by a script which specifies which node is in which rack, where the rack is the network **switch** to which the node is connected, not the metal framework where the nodes are physically stacked.
- The script is referenced in **net.topology.script.property.file** in the Hadoop configuration file **core-site.xml**. For example:

```
<property>
  <name>net.topology.script.file.name</name>
  <value>/etc/hadoop/conf/rack-topology.sh </value>
</property>
```

- The network topology script (**net.topology.script.file.name** in the example above) receives as arguments one or more IP addresses of nodes in the cluster. It returns on stdout a list of rack names, one for each input.
- One simple approach is to use IP addressing of **10.x.y.z** where x = cluster number, y = rack number, z = node within rack; and an appropriate script to decode this into **y/z**.

Setting rack network topology (Rack Awareness)

For small clusters in which all servers are connected by a single switch, there are only two levels of locality: on-machine and off-machine. When loading data from a DataNode's local drive into HDFS, the NameNode will schedule one copy to go into the local DataNode, and will pick two other machines at random from the cluster.

For larger Hadoop installations which span multiple racks, it is important to ensure that replicas of data exist on multiple racks so that the loss of a switch does not render portions of the data unavailable, due to all of the replicas being underneath it.

HDFS can be made rack-aware by the use of a script which allows the master node to map the network topology of the cluster. While alternate configuration strategies can be used, the default implementation allows you to provide an executable script which returns the rack address of each of a list of IP addresses.

The network topology script receives as arguments one or more IP addresses of nodes in the cluster. It returns on stdout a list of rack names, one for each input. The input and output order must be consistent.

To set the rack mapping script, specify the key topology.script.file.name in conf/Hadoop-site.xml. This provides a command to run to return a rack id; it must be an executable script or program. By default, Hadoop will attempt to send a set of IP addresses to the file as several separate command line arguments. You can control the maximum acceptable number of arguments with the topology.script.number.args key.

Rack ids in Hadoop are hierarchical and look like path names. By default, every node has a rack id of /default-rack. You can set rack ids for nodes to any arbitrary path, such as /foo/bar-rack. Path elements further to the left are higher up the tree, so a reasonable structure for a large installation may be /top-switch-name/rack-name.

The following example script performs rack identification based on IP addresses given a hierarchical IP addressing scheme enforced by the network administrator. This may work directly for simple installations; more complex network configurations may require a file- or table-based lookup process. Care should be taken in that case to keep the table up-to-date as nodes are physically relocated, etc. This script requires that the maximum number of arguments be set to 1.

```
#!/bin/bash
# Set rack id based on IP address.
# Assumes network administrator has complete control
# over IP addresses assigned to nodes and they are
# in the 10.x.y.z address space. Assumes that
# IP addresses are distributed hierarchically. e.g.,
# 10.1.y.z is one data center segment and 10.2.y.z is another;
# 10.1.1.z is one rack, 10.1.2.z is another rack in
# the same segment, etc.)
#
# This is invoked with an IP address as its only argument
# get IP address from the input
ipaddr=$0
# select "x.y" and convert it to "x/y"
segments=`echo $ipaddr | cut --delimiter=. --fields=2-3 --output-delimiter=/`  
echo /${segments}
```

A more complex rack-aware script:

```

File name: rack-topology.sh
#!/bin/bash
# Adjust/Add the property "net.topology.script.file.name"
# to core-site.xml with the "absolute" path to this
# file. ENSURE the file is "executable".
# Supply appropriate rack prefix
RACK_PREFIX=default
# To test, supply a hostname as script input:
if [ $# -gt 0 ]; then
CTL_FILE=${CTL_FILE:-"rack_topology.data"}
HADOOP_CONF=${HADOOP_CONF:-"/etc/hadoop/conf"}
if [ ! -f ${HADOOP_CONF}/${CTL_FILE} ]; then
  echo -n "/$RACK_PREFIX/rack "
  exit 0
fi
while [ $# -gt 0 ] ; do
  nodeArg=$1
  exec< ${HADOOP_CONF}/${CTL_FILE}
  result=""
  while read line ; do
    ar=( $line )
    if [ "${ar[0]}" = "$nodeArg" ] ; then
      result="${ar[1]}"
    fi
  done
  shift
  if [ -z "$result" ] ; then
    echo -n "/$RACK_PREFIX/rack "
  else
    echo -n "/$RACK_PREFIX/rack_$result "
  fi
done
else
  echo -n "/$RACK_PREFIX/rack "
fi

```

Sample Topology Data File

```

File name: rack_topology.data
# This file should be:
#   - Placed in the /etc/hadoop/conf directory
#   - On the Namenode (and backups IE: HA, Failover, etc)
#   - On the Job Tracker OR Resource Manager (and any Failover JT's/RM's)
# This file should be placed in the /etc/hadoop/conf directory.
# Add Hostnames to this file. Format <host ip> <rack_location>
192.168.2.10 01
192.168.2.11 02
192.168.2.12 03

```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Compression of files

- File compression brings two benefits:
 - reduces the space need to store files
 - speeds up data transfer across the network or to/from disk
- **But is the data splitable?** (necessary for parallel reading)
- Use codecs, such as [org.apache.hadoop.io.compress.SnappyCodec](#)

Compression Format	Algorithm	Filename extension	Splittable?
DEFLATE	DEFLATE	.deflate	No
gzip	DEFLATE	.gz	No
bzip2	bzip2	.bz2	Yes
LZO	LZO	.lzo / .cmx	Yes, If indexed in preprocessing
LZ4	LZ4	.lz4	No
Snappy	Snappy	.snappy	No

Compression of files

gzip

gzip is naturally supported by Hadoop. gzip is based on the DEFLATE algorithm, which is a combination of LZ77 and Huffman Coding.

bzip2

bzip2 is a freely available, patent free, high-quality data compressor. It typically compresses files to within 10% to 15% of the best available techniques (the PPM family of statistical compressors), whilst being around twice as fast at compression and six times faster at decompression.

LZO

The LZO compression format is composed of many smaller (~256K) blocks of compressed data, allowing jobs to be split along block boundaries. Moreover, it was designed with speed in mind: it decompresses about twice as fast as gzip, meaning it is fast enough to keep up with hard drive read speeds. It does not compress quite as well as gzip; expect files that are on the order of 50% larger than their gzipped version. But that is still 20-50% of the size of the files without any compression at all, which means that IO-bound jobs complete the map phase about four times faster.

LZO = Lempel Ziv Oberhumer, <http://en.wikipedia.org/wiki/Lempel-Ziv-Oberhumer>. A free software tool which implements it is Izop. The original library was written in ANSI C, and it has been made available under the GNU General Purpose License. Versions of LZO are available for the Perl, Python, and Java languages. The copyright for the code is owned by Markus F. X. J. Oberhumer.

LZ4

LZ4 is a lossless data compression algorithm that is focused on compression and decompression speed. It belongs to the LZ77 family of byte-oriented compression schemes. The algorithm gives a slightly worse compression ratio than algorithms like gzip. However, compression speeds are several times faster than gzip while decompression speeds can be significantly faster than LZO .

[http://en.wikipedia.org/wiki/LZ4_\(compression_algorithm\)](http://en.wikipedia.org/wiki/LZ4_(compression_algorithm)), the reference implementation in C by Yann Collet is licensed under a BSD license.

Snappy

Snappy is a compression/decompression library. It does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression. For instance, compared to the fastest mode of zlib, Snappy is an order of magnitude faster for most inputs, but the resulting compressed files are anywhere from 20% to 100% bigger. On a single core of a Core i7 processor in 64-bit mode, Snappy compresses at about 250 MB/sec or more and decompresses at about 500 MB/sec or more. Snappy is widely used inside Google, in everything from BigTable and MapReduce to RPC systems.

All packages produced by the Apache Software Foundation (ASF), such as Hadoop, are implicitly licensed under the Apache License, Version 2.0, unless otherwise explicitly stated. The licensing of other algorithms, such as LZO, that are not licensed under ASF may pose some problems for distributions that rely solely on the Apache License.

Which compression format should I use?

- Most to least effective:
 - use a container format (Sequence file, Avro, ORC, or Parquet)
 - for files use a fast compressor such as LZO, LZ4, or Snappy
 - use a compression format that supports splitting, such as bz2 (slow) or one that can be indexed to support splitting, such as LZO
 - split files into chunks and compress each chunk separately using a supported compression format (does not matter if splittable) - choose a chunk size so that compressed chunks are approximately the size of an HDFS block
 - store files uncompressed

Take advantage of compression - but the compression format should depend on file size, data format, and tools used

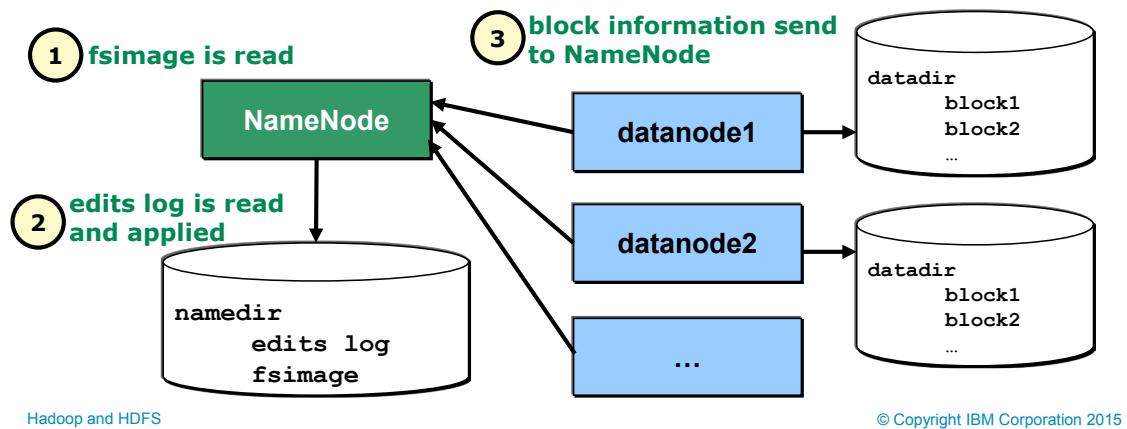
Which compression format should I use?

References:

- http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Data_Compression
- <http://comphadoop.weebly.com>
- http://www.slideshare.net/Hadoop_Summit/kamat-singh-june27425pmroom210cv2
- http://www.cloudera.com/content/cloudera/en/documentation/core/v5-3-x/topics/admin_data_compression_performance.html

NameNode startup

1. NameNode reads fsimage in memory
2. NameNode applies editlog changes
3. NameNode waits for block data from data nodes
 - NameNode does not store the physical-location information of the blocks
 - NameNode exits SafeMode when 99.9% of blocks have at least one copy accounted for



NameNode startup

During start up, the NameNode loads the file system state from the fsimage and the edits log file. It then waits for DataNodes to report their blocks so that it does not prematurely start replicating the blocks though enough replicas already exist in the cluster.

During this time NameNode stays in SafeMode. SafeMode for the NameNode is essentially a read-only mode for the HDFS cluster, where it does not allow any modifications to file system or blocks. Normally the NameNode leaves SafeMode automatically after the DataNodes have reported that most file system blocks are available.

If required, HDFS can be placed in SafeMode explicitly using the command `hadoop dfsadmin -safemode`. The NameNode front page shows whether SafeMode is on or off.

NameNode files (as stored in HDFS)

```
[root@vm hdfs]# cd /hadoop/hdfs
[root@vm hdfs]# ls -l
total 12
drwxr-x---. 3 hdfs hadoop 4096 Apr 20 05:45 data
drwxr-xr-x. 3 hdfs hadoop 4096 Apr 20 05:45 namenode
drwxr-xr-x. 3 hdfs hadoop 4096 Apr 20 05:45 namesecondary
[root@vm hdfs]# ls -l namenode/current:
total 53296
...
-rw-r--r--. 1 hdfs hadoop 1048576 Apr 17 04:00 edits_00000000000000047319-000000000000000048279
-rw-r--r--. 1 hdfs hadoop 1048576 Apr 17 04:33 edits_00000000000000048280-000000000000000048317
-rw-r--r--. 1 hdfs hadoop 1048576 Apr 17 04:50 edits_00000000000000048318-000000000000000048340
-rw-r--r--. 1 hdfs hadoop 1048576 Apr 17 05:46 edits_00000000000000048341-000000000000000048809
-rw-r--r--. 1 hdfs hadoop 1048576 Apr 20 05:00 edits_00000000000000048810-000000000000000049264
-rw-r--r--. 1 hdfs hadoop 1048576 Apr 20 05:17 edits_00000000000000049265-000000000000000049322
-rw-r--r--. 1 hdfs hadoop 1048576 Apr 20 05:44 edits_inprogress_00000000000000049323
-rw-r--r--. 1 hdfs hadoop 1322806 Apr 16 23:39 fsimage_00000000000000046432
-rw-r--r--. 1 hdfs hadoop 62 Apr 16 23:39 fsimage_00000000000000046432.md5
-rw-r--r--. 1 hdfs hadoop 1394702 Apr 20 04:11 fsimage_00000000000000048809
-rw-r--r--. 1 hdfs hadoop 62 Apr 20 04:11 fsimage_00000000000000048809.md5
-rw-r--r--. 1 hdfs hadoop 6 Apr 20 05:37 seen_txid
-rw-r--r--. 1 hdfs hadoop 207 Apr 20 04:11 VERSION
```

Hadoop and HDFS

© Copyright IBM Corporation 2015

NameNode files (as stored in HDFS)

These are the actual storage files (in HDFS) where the NameNode stores its metadata:

- fsimage
- edits
- VERSION

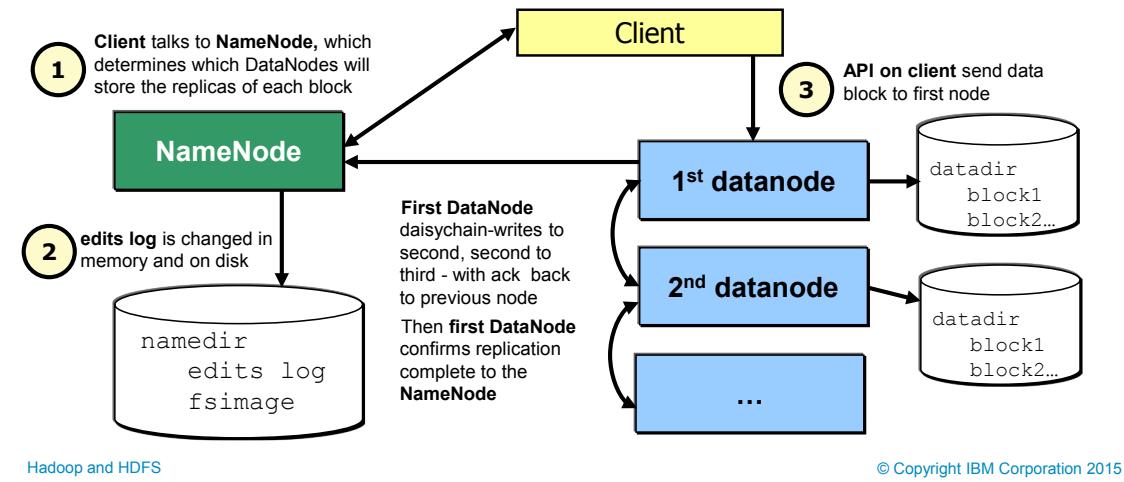
There is a current **edits_inprogress** file that is accumulating edits (adds, deletes) since the last update of the fsimage. This current edits file is closed off and the changes incorporated into a new version of the fsimage based on whichever of two configurable events occurs first:

- edits file reaches a certain size (here 1 MB, default 64 MB)
- time limit between updates is reached, and there have been updates (default 1 hour)

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Adding a file to HDFS: replication pipelining

1. File is added to NameNode memory by persisting info in edits log
2. Data is written in blocks to DataNodes
 - DataNode starts chained copy to two other DataNodes
 - if at least one write for each block succeeds, the write is successful



Adding a file to HDFS: replication pipelining

Data Blocks

HDFS is designed to support very large files. Applications that are compatible with HDFS are those that deal with large data sets. These applications write their data only once but they read it one or more times and require these reads to be satisfied at streaming speeds. HDFS supports write-once-read-many semantics on files. A typical block size used by HDFS is 128 MB. Thus, an HDFS file is chopped up into 128 MB "splits," and if possible, each split (or block) will reside on a different DataNode.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Staging

A client request to create a file does not reach the NameNode immediately. In fact, initially the HDFS client caches the file data into a temporary local file. Application writes are transparently redirected to this temporary local file. When the local file accumulates data worth over one HDFS block size, the client contacts the NameNode. The NameNode inserts the file name into the file system hierarchy and allocates a data block for it. The NameNode responds to the client request with the identity of the DataNode and the destination data block. Then the client flushes the block of data from the local temporary file to the specified DataNode. When a file is closed, the remaining un-flushed data in the temporary local file is transferred to the DataNode. The client then tells the NameNode that the file is closed. At this point, the NameNode commits the file creation operation into a persistent store. If the NameNode dies before the file is closed, the file is lost.

The above approach was adopted after careful consideration of target applications that run on HDFS. These applications need streaming writes to files. If a client writes to a remote file directly without any client side buffering, the network speed and the congestion in the network impacts throughput considerably. This approach is not without precedent. Earlier distributed file systems, such as AFS, have used client side caching to improve performance. A POSIX requirement has been relaxed to achieve higher performance of data uploads.

Replication Pipelining

When a client is writing data to an HDFS file, its data is first written to a local file as explained previously. Suppose the HDFS file has a replication factor of three. When the local file accumulates a full block of user data, the client retrieves a list of DataNodes from the NameNode. This list contains the DataNodes that will host a replica of that block. The client then flushes the data block to the first DataNode. The first DataNode starts receiving the data in small portions (4 KB), writes each portion to its local repository and transfers that portion to the second DataNode in the list. The second DataNode, in turn starts receiving each portion of the data block, writes that portion to its repository and then flushes that portion to the third DataNode. Finally, the third DataNode writes the data to its local repository. Thus, a DataNode can be receiving data from the previous one in the pipeline and at the same time forwarding data to the next one in the pipeline. The data is pipelined from one DataNode to the next.

For good descriptions of the process, see the tutorials at:

- *HDFS Users Guide* at <http://hadoop.apache.org/docs/r2.7.0/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html>
- *An introduction to the Hadoop Distributed File System* at <http://www.ibm.com/developerworks/library/wa-introhdfs/>
- *The Hadoop Distributed File System* at <https://developer.yahoo.com/hadoop/tutorial/module2.html>

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Managing the cluster

- Adding a data node
 - Start new DataNode (pointing to NameNode)
 - If required, run balancer to rebalance blocks across the cluster:
`hadoop balancer`
- Removing a node
 - Simply remove DataNode
 - Better: Add node to exclude file and wait till all blocks have been moved
 - Can be checked in server admin console server:50070
- Checking filesystem health
 - Use: `hadoop fsck`

Managing the cluster

Apache Hadoop clusters grow and change with use. The normal method is to use Apache Ambari to build your initial cluster with a base set of Hadoop services targeting known use cases. You may want to add other services for new use cases, and even later you may need to expand the storage and processing capacity of the cluster.

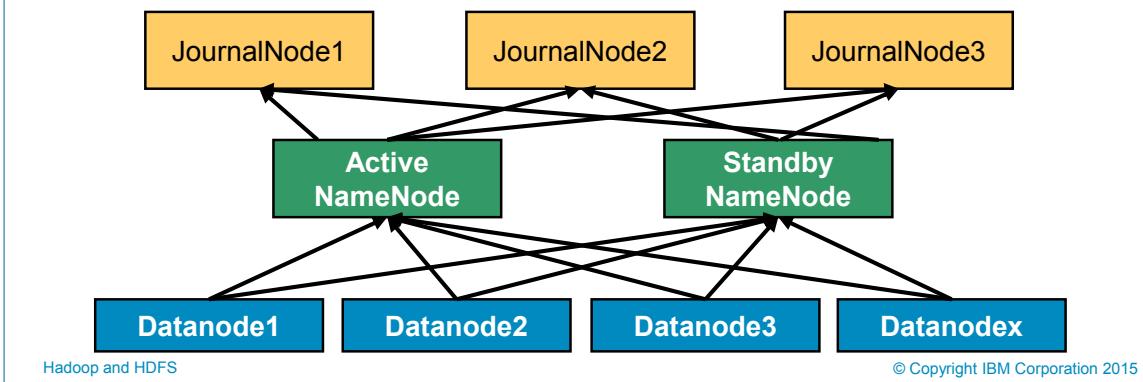
Ambari can help in both scenarios, the initial configuration and the later expansion/reconfiguration of your cluster.

When you can add more hosts to the cluster you can assign these hosts to run as DataNodes (and NodeManagers under YARN, as you will see later). This allows you to expand both your HDFS storage capacity and your overall processing power.

Similarly you can removed DataNodes if they are malfunctioning or you want to reorganize your cluster.

HDFS-2 NameNode HA (High Availability)

- HDFS-2 adds NameNode High Availability
- Standby NameNode needs filesystem transactions and block locations for fast failover
- Every filesystem modification is logged to at least 3 quorum journal nodes by active Namenode
 - Standby Node applies changes from journal nodes as they occur
 - Majority of journal nodes define reality
 - Split Brain is avoided by JournalNodes (They will only allow one NameNode to write to them)
- DataNodes send block locations and heartbeats to both NameNodes
- Memory state of Standby NameNode is very close to Active NameNode
 - Much faster failover than cold start



HDFS-2 NameNode HA (High Availability)

With Hadoop 2, high availability is supported out-of-the-box.

Features:

- two available NameNodes: Active and Standby
- transactions logged to Quorum Journal Nodes (QJM)
- standby node periodically gets updates
- DataNodes send block locations and heartbeats to **both** NameNodes
- when failures occur, Standby can take over with very small downtime
- no cold start

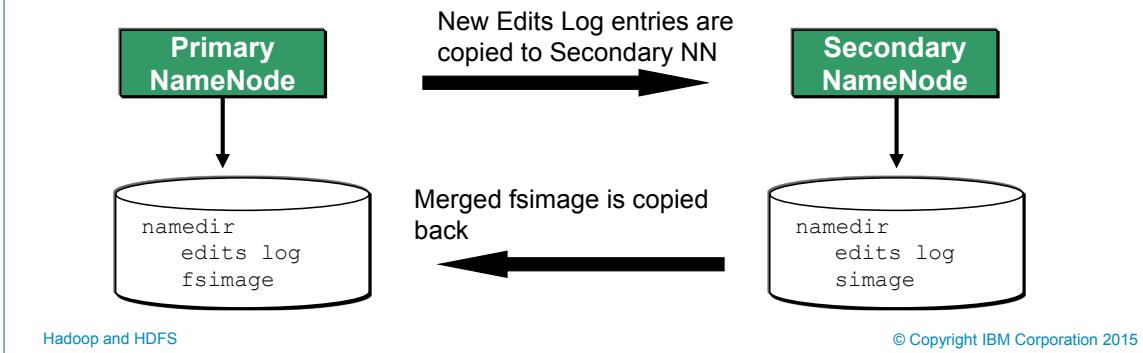
Deployment:

- need to have two dedicated NameNodes in the cluster
- QJM may coexist with other services (at least 3 in a cluster)
- no need of a Secondary NameNode

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Secondary NameNode

- During operation primary NameNode cannot merge fsImage and edits log
- This is done on the secondary NameNode
 - Every couple minutes, secondary NameNode copies new edit log from primary NN
 - Merges edits log into fsimage
 - Copies the new merged fsImage back to primary NameNode
- Not HA but faster startup time
 - Secondary NN does not have complete image. In-flight transactions would be lost
 - Primary NameNode needs to merge less during startup
- Was temporarily deprecated because of NameNode HA but has some advantages
 - (No need for Quorum nodes, less network traffic, less moving parts)



Secondary NameNode

In the older approach, a Secondary NameNode is used.

The NameNode stores the HDFS filesystem information in a file named fsimage. Updates to the file system (add/remove blocks) are not updating the fsimage file, but instead are logged into a file, so the I/O is fast append only streaming as opposed to random file writes. When restarting, the NameNode reads the fsimage and then applies all the changes from the log file to bring the filesystem state up to date in memory. This process takes time.

The job of the Secondary NameNode's is not to be a secondary to the name node, but only to periodically read the filesystem changes log and apply them into the fsimage file, thus bringing it up to date. This allows the namenode to start up faster next time.

Unfortunately, the Secondary NameNode service is not a standby secondary namenode, despite its name. Specifically, it does not offer HA for the NameNode. This is well illustrated in the slide above.

Note that more recent distributions have NameNode High Availability using NFS (shared storage) and/or NameNode High Availability using a Quorum Journal Manager (QJM).

Possible FileSystem setup approaches

- Hadoop 2 with HA
 - no single point of failure
 - wide community support
- Hadoop 2 without HA (or, Hadoop 1.x in older versions)
 - copy namedir to NFS (RAID)
 - have virtual IP for backup NameNode
 - still some failover time to read blocks, no instant failover but less overhead
- GPFS (requires BigInsights Management Module add-in)
 - no single point of failure
 - POSIX compliant
 - advanced features like cold storage, backup, and restore
 - GPFS FPO (file placement option) is now called Spectrum Scale

Possible FileSystem setup approaches

The slide shows three approaches to high availability.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Federated NameNode (HDFS2)

- New in Hadoop2: NameNodes (NN) can be federated
 - Historically NameNodes can become a bottleneck on huge clusters
 - One million blocks or ~100TB of data require roughly one GB RAM in NN
- Blockpools
 - Administrator can create separate blockpools/namespaces with different NNs
 - DataNodes register on all NNs
 - DataNodes store data of all blockpools (otherwise setup separate clusters)
 - New **ClusterID** identifies all NNs in a cluster.
 - A namespace and its block pool together are called Namespace Volume
 - You define which blockpool to use by connecting to a specific NN
 - Each NameNode still has its own separate backup/secondary/checkpoint node
- Benefits
 - One NN failure will not impact other blockpools
 - Better scalability for large numbers of file operations

Hadoop and HDFS

© Copyright IBM Corporation 2015

Federated NameNode (HDFS2)

References:

- <http://hadoop.apache.org/docs/current2/hadoop-project-dist/hadoop-hdfs/Federation.html#Background>
- <http://hadoop.apache.org/docs/r2.7.0/hadoop-project-dist/hadoop-hdfs/Federation.html>

With Federated NameNodes in HDFS2, there are main layers:

- Namespace
 - Consists of directories, files and blocks.
 - It supports all the namespace related file system operations such as create, delete, modify and list files and directories.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

- Block Storage Service, which has two parts:
 - Block Management (performed in the NameNode)
 - Provides DataNode cluster membership by handling registrations, and periodic heart beats.
 - Processes block reports and maintains location of blocks.
 - Supports block related operations such as create, delete, modify and get block location.
 - Manages replica placement, block replication for under replicated blocks, and deletes blocks that are over replicated.
 - Storage is provided by DataNodes by storing blocks on the local file system and allowing read/write access.

The prior HDFS architecture allows only a single namespace for the entire cluster. In that configuration, a single NameNode manages the namespace. HDFS Federation addresses this limitation by adding support for multiple NameNodes/namespaces to HDFS.

Multiple NameNodes / Namespaces

In order to scale the name service horizontally, federation uses multiple independent NameNodes / namespaces. The NameNodes are federated; the NameNodes are independent and do not require coordination with each other. The DataNodes are used as common storage for blocks by all the NameNodes. Each DataNode registers with all the NameNodes in the cluster. DataNodes send periodic heartbeats and block reports. They also handle commands from the NameNodes.

Users may use ViewFS to create personalized namespace views. ViewFS is analogous to client-side mount tables in some UNIX/Linux systems.

fs: file system shell (1 of 4)

- **File System Shell (fs)**

- Invoked as follows:

```
hadoop fs <args>
```

- Example: listing the current directory in HDFS

```
hadoop fs -ls .
```

Note that the current directory is designated by dot (".")

- the here symbol in Linux/UNIX

If you want the root of the HDFS file system, you would use slash ("/")

fs - file system shell

HDFS can be manipulated through a Java API or through a command line interface. All commands for manipulating HDFS through Hadoop's command line interface begin with "hadoop", a space, and "fs" (or "dfs"). This is the file system shell. This is followed by the command name as an argument to "hadoop fs". These commands start with a dash. For example, the "ls" command for listing a directory is a common UNIX command and is preceded with a dash. As on UNIX systems, ls can take a path as an argument. In this example, the path is the current directory, represented by a single dot.

fs is one of the command options for **hadoop**. If you just type the command **hadoop** by itself, you will see other options.

A good tutorial at this stage can be found at
<https://developer.yahoo.com/hadoop/tutorial/module2.html>.

fs: file system shell (2 of 4)

- FS shell commands take URIs as argument

- URI format:

scheme://authority/path

- Scheme:

- For the local filesystem, the scheme is **file**
- For HDFS, the scheme is **hdfs**

- Authority is the hostname and port of the NameNode

```
hadoop fs -copyFromLocal file:///myfile.txt
dfs://localhost:9000/user/virtuser/myfile.txt
```

- Scheme and authority are often optional

- Defaults are taken from configuration file **core-site.xml**

Just as for the **ls** command, the file system shell commands can take paths as arguments. These paths can be expressed in the form of uniform resource identifiers or URIs. The URI format consists of a scheme, an authority, and path. There are multiple schemes supported. The local file system has a scheme of "file". HDFS has a scheme called "hdfs."

For example, if you want to copy a file called "myfile.txt" from your local filesystem to an HDFS file system on the localhost, you can do this by issuing the command shown.

The **copyFromLocal** command takes a URI for the source and a URI for the destination.

"Authority" is the hostname of the NameNode. For example, if the NameNode is in localhost and accessed on port 9000, the authority would be **localhost:9000**.

The scheme and the authority do not always need to be specified. Instead you may rely on their default values. These defaults can be overridden by specifying them in a file named **core-site.xml** in the **conf** directory of your Hadoop installation.

fs: file system shell (3 of 4)

- Many POSIX-like commands
 - cat, chgrp, chmod, chown, cp, du, ls, mkdir, mv, rm, stat, tail
- Some HDFS-specific commands
 - copyFromLocal, put, copyToLocal, get, getmerge, setrep
- **copyFromLocal / put**
 - Copy files from the local file system into fs

```
hadoop fs -copyFromLocal localsrc dst
```

or

```
hadoop fs -put localsrc dst
```

HDFS supports many POSIX-like commands. HDFS is not a fully POSIX (Portable operating system interface for UNIX) compliant file system, but it supports many of the commands. The HDFS commands are mostly easily-recognized UNIX commands like `cat` and `chmod`. There are also a few commands that are specific to HDFS such as `copyFromLocal`.

Note that:

- `localsrc` and `dst` are placeholders for your actual file(s)
- `localsrc` can be a directory or a list of files separated by space(s)
- `dst` can be a new file name (in HDFS) for a single-file-copy, or a directory (in HDFS), that is the destination directory

Example:

```
hadoop fs -put *.txt ./Gutenberg
```

copies all the text files in the local Linux directory with the suffix of **.txt** to the directory **Gutenberg** in the users home directory in HDFS

The "direction" implied by the names of these commands (`copyFromLocal`, `put`) is relative to the user, who can be thought to be situated outside HDFS.

Also, you should note there is no **cd** (change directory) command available for hadoop.

fs: file system shell (4 of 4)

- **copyToLocal / get**

- Copy files from fs into the local file system

```
hadoop fs -copyToLocal [-ignorecrc] [-crc]<src><localdst>
```

or

```
hadoop fs -get [-ignorecrc] [-crc] <src> <localdst>
```

- Creating a directory: **mkdir**

```
hadoop fs -mkdir /newdir
```

The **copyToLocal** (aka **get**) command copies files out of the file system you specify and into the local file system.

get

Usage: **hadoop fs -get [-ignorecrc] [-crc] <src> <localdst>**

- Copy files to the local file system. Files that fail the CRC check may be copied with the **-ignorecrc** option. Files and CRCs may be copied using the **-crc** option.
- Example: **hadoop fs -get hdfs:/mydir/file file:///home/hdpadmin/localfile**

Another important note: for files in Linux, where you would use the **file://** authority, two slashes represent files relative to your current Linux directory (**pwd**). To reference files absolutely, use three slashes (and mentally pronounce as "slash-slash pause slash").

Unit summary

- Understand the basic need for a big data strategy in terms of parallel reading of large data files and internode network speed in a cluster
- Describe the nature of the Hadoop Distributed File System (HDFS)
- Explain the function of the NameNode and DataNodes in an Hadoop cluster
- Explain how files are stored and blocks ("splits") are replicated

Unit summary

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1

File access and basic commands with HDFS

Exercise 1: File access and basic commands with HDFS

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1: File access and basic commands with HDFS

Purpose:

This exercise is intended to provide you with experience in using the Hadoop Distributed File System (HDFS). The basic HDFS file system commands learned here will be used throughout the remainder of the course.

You will also be moving some data into HDFS that will be used in later units of this course. The files that you will need are stored in the Linux directory /home/labfiles.

VM Hostname: <http://ibmclass.localdomain>

User/Password: biadmin / biadmin
root/dalvm3

Task 1. Learn some of the basic HDFS file system commands.

The major reference for the HDFS File System Commands can be found on the Apache Hadoop website. Additional commands can be found there. The URL for Hadoop 2.7.0 is:

<http://hadoop.apache.org/docs/r2.7.0/hadoop-project-dist/hadoop-common/FileSystemShell.html>

Use the documentation that corresponds to your current release of HDFS (substitute the appropriate value for r2.7.0, if different).

The HDFS File System Commands are generally prefixed by:

`hadoop fs`

but can also be executed with `dfs` instead of `fs` and also in the form:

`hdfs fs`

Other HDFS commands - generally administrative - use parameters other than `fs` (for example, `fsck`, `rebalance`, ...).

1. Connect to your classroom VMware image and login as **biadmin**.
2. In a new terminal window, type `cd` to change to your home directory.
3. Type `pwd` to verify that you are in biadmin's home directory.

4. To verify that, as the user biadmin, you have a file system directory in HDFS, type `hadoop fs -ls ..`

Note: At the end of this command there is a period (it means the current directory in HDFS).

If you completed the first section of this course, running this command will give four results. Otherwise your results appear as shown below:

```
[biadmin@ibmclass ~]$ hadoop fs -ls .
[biadmin@ibmclass ~]$
```

In Linux, there is no response to the command; this indicates success. If you get this response, go to step 8.

If, however, you receive an error response such as:

```
[biadmin@ibmclass ~]$ hadoop fs -ls .
ls: '.' : No such file or directory
[biadmin@ibmclass ~]$
```

then you do not have a home directory in HDFS. You will need to have one created for you. The ability to create a home directory for you in HDFS is not something that you can do; it must be done as the hdfs user. You will do this in steps 5 through 7.

5. To login as the hdfs user, use `sudo su - hdfs`

```
[biadmin@ibmclass ~]$ sudo su - hdfs
[sudo] password for biadmin:
[hdfs@ibmclass ~]$
```

or `su -` (the root password is `dalvm3`)

```
[biadmin@ibmclass ~]$ su -
Password:
[root@ibmclass ~]# su - hdfs
[hdfs@ibmclass ~]$
```

6. Execute the following three commands:

```
hadoop fs -mkdir /user/biadmin
hadoop fs -chown biadmin /user/biadmin
hadoop fs -chgrp biadmin /user/biadmin
```

```
[hdfs@ibmclass ~]$ hadoop fs -mkdir /user/biadmin
[hdfs@ibmclass ~]$ hadoop fs -ls /
Found 8 items
drwxrwxrwx  - yarn    hadoop      0 2015-04-15 12:21 /app-logs
drwxr-xr-x  - hdfs   hdfs       0 2015-04-15 12:19 /apps
drwxrwxrwx  - hdfs   hadoop      0 2015-05-19 17:29 /biginsights
drwxr-xr-x  - hdfs   hdfs       0 2015-04-15 12:13 /iop
drwxr-xr-x  - mapred hdfs      0 2015-04-15 12:12 /mapred
drwxr-xr-x  - hdfs   hdfs      0 2015-04-15 12:12 /mr-history
drwxrwxrwx  - hdfs   hdfs      0 2015-05-20 18:17 /tmp
drwxr-xr-x  - hdfs   hdfs      0 2015-06-04 09:02 /user
[hdfs@ibmclass ~]$ hadoop fs -ls /user
Found 8 items
drwxrwx---  - ambari-qa hdfs      0 2015-04-15 12:25 /user/ambari-qa
drwxr-xr-x  - hdfs   hdfs      0 2015-06-04 09:02 /user/biadmin
drwxrwxrwx  - bigr   hdfs      0 2015-05-20 18:17 /user/bigr
drwxr-xr-x  - hcat   hdfs      0 2015-04-15 12:23 /user/hcat
drwxrwxrwx  - hive   hdfs      0 2015-04-15 12:19 /user/hive
drwxrwxr-x  - oozie  hdfs      0 2015-04-15 12:20 /user/oozie
drwxr-xr-x  - spark   hadoop     0 2015-04-15 12:14 /user/spark
drwxrwxr-x  - tauser  hdfs      0 2015-05-20 12:18 /user/tauser
[hdfs@ibmclass ~]$ hadoop fs -chown biadmin /user/biadmin
[hdfs@ibmclass ~]$ hadoop fs -chgrp biadmin /user/biadmin
[hdfs@ibmclass ~]$ hadoop fs -ls /user
Found 8 items
drwxrwx---  - ambari-qa hdfs      0 2015-04-15 12:25 /user/ambari-qa
drwxr-xr-x  - biadmin  biadmin     0 2015-06-04 09:02 /user/biadmin
drwxrwxrwx  - bigr   hdfs      0 2015-05-20 18:17 /user/bigr
drwxr-xr-x  - hcat   hdfs      0 2015-04-15 12:23 /user/hcat
drwxrwxrwx  - hive   hdfs      0 2015-04-15 12:19 /user/hive
drwxrwxr-x  - oozie  hdfs      0 2015-04-15 12:20 /user/oozie
drwxr-xr-x  - spark   hadoop     0 2015-04-15 12:14 /user/spark
drwxrwxr-x  - tauser  hdfs      0 2015-05-20 12:18 /user/tauser
```

7. Logout from hdfs (and root, if necessary) by using **exit** or **Ctrl-D**.

Alternatively, you can close your current terminal window and open a new one.

You have now have a home directory in HDFS. This home directory is /user/biadmin (note that this is "user" and not "usr" and not "home").

8. To validate that you now have a home directory in HDFS, run the following commands:

```
hadoop fs -ls
hadoop fs -ls .
hadoop fs -ls /user
```

```
[biadmin@ibmclass ~]$ hadoop fs -ls
[biadmin@ibmclass ~]$

[biadmin@ibmclass ~]$ hadoop fs -ls .
[biadmin@ibmclass ~]$

[hdfs@ibmclass ~]$ hadoop fs -ls /user
Found 8 items
drwxrwx---  - ambari-qa hdfs      0 2015-04-15 12:25 /user/ambari-qa
drwxr-xr-x  - biadmin   biadmin    0 2015-06-04 09:02 /user/biadmin
drwxrwxrwx  - bigr     hdfs      0 2015-05-20 18:17 /user/bigr
drwxr-xr-x  - hcat     hdfs      0 2015-04-15 12:23 /user/hcat
drwx-----  - hive     hdfs      0 2015-04-15 12:19 /user/hive
drwxrwxr-x  - oozie    hdfs      0 2015-04-15 12:20 /user/oozie
drwxr-xr-x  - spark    hadoop    0 2015-04-15 12:14 /user/spark
drwxrwxr-x  - tauser   hdfs      0 2015-05-20 12:18 /user/tauser
```

If you completed the first section of this course, your results may differ slightly.

You are now ready to do some exploration of the HDFS file system and your Hadoop ecosystem.

9. To create a subdirectory called **Gutenberg** in your HDFS user directory, type `hadoop fs -mkdir Gutenberg`. Note that the directory you just created, defaulted to your home directory. This is how Linux handles this command. If you wanted the directory elsewhere, then you need to fully qualify it.
10. Type `hadoop fs -ls` to list your directory.
11. Use the recursive option (-R) of `ls` to see if there are any files in your Gutenberg directory (there are none at the moment): `hadoop fs -ls -R`

```
[biadmin@ibmclass ~]$ hadoop fs -ls
[biadmin@ibmclass ~]$ hadoop fs -mkdir Gutenberg
[biadmin@ibmclass ~]$ hadoop fs -ls
Found 1 items
drwxr-xr-x  - biadmin biadmin      0 2015-06-04 10:41 Gutenberg
[biadmin@ibmclass ~]$ hadoop fs -ls -R
drwxr-xr-x  - biadmin biadmin      0 2015-06-04 10:41 Gutenberg
[biadmin@ibmclass ~]$
```

12. Use the `hadoop fs -put` command to move files from the Linux `/home/labfiles` directory into your HDFS directory, Gutenberg:

```
hadoop fs -put /home/biadmin/labfiles/*.txt Gutenberg
```

This command can also be executed as:

```
hadoop fs -copyFromLocal /home/biadmin/labfiles/*.txt Gutenberg
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

13. Repeat the list command with the recursive option (step 10).

```
[biadmin@ibmclass ~]$ hadoop fs -put /home/labfiles/*.txt Gutenberg
[biadmin@ibmclass ~]$ hadoop fs -ls .
Found 1 items
drwxr-xr-x  - biadmin biadmin          0 2015-06-04 11:01 Gutenberg
[biadmin@ibmclass ~]$ hadoop fs -ls -R
drwxr-xr-x  - biadmin biadmin          0 2015-06-04 11:01 Gutenberg
-rw-r--r--  3 biadmin biadmin  421504 2015-06-04 11:01 Gutenberg/Frankenstein.txt
-rw-r--r--  3 biadmin biadmin  697802 2015-06-04 11:01
Gutenberg/Pride_and_Prejudice.txt
-rw-r--r--  3 biadmin biadmin  757223 2015-06-04 11:01
Gutenberg/Tale_of_Two_Cities.txt
-rw-r--r--  3 biadmin biadmin 281398 2015-06-04 11:01 Gutenberg/The_Prince.txt
[biadmin@ibmclass ~]$
```

Note here in the listing of files the following for the last file:

-rw-r--r-- 3 biadmin biadmin 281398

Here you will see the read-write (*rw*) permissions that you would find with a typical Linux file.

The "3" here is the typical replication factor for the blocks (or "splits") of the individual files. These files are too small (last file is 281KB) to have more than one block (the max block size is 128MB), but *each block of each file* is replicated three times.

You may see "1" instead of "3" in a single-node cluster (pseudo-distributed mode). That too is normal for a single-node cluster as it does not really make sense to replicate multiple copies on a single node.

Task 2. Explore one of the HDFS administrative commands.

There are a number of HDFS administration commands in addition to the HDFS file system commands. A reference for administration commands is:

<https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/CommandsManual.html>.

We will look at just one of them, `fsck`. We will run it as: `hdfs fsck`

Administrative commands cannot be normally run as regular users. You will do the following as the `hdfs` user.

`fsck`

Runs a HDFS filesystem checking utility.

Usage: `hadoop fsck [GENERIC OPTIONS] <path> [-move | -delete | -openforwrite] [-files [-blocks [-locations | -racks]]]`

COMMAND_OPTION	Description
<code>path</code>	Start checking from this path.
<code>-move</code>	Move corrupted files to /lost+found
<code>-delete</code>	Delete corrupted files.
<code>-openforwrite</code>	Print out files opened for write.
<code>-files</code>	Print out files being checked.
<code>-blocks</code>	Print out block report.
<code>-locations</code>	Print out locations for every block.
<code>-racks</code>	Print out network topology for data-node locations.

- To log in as the `hdfs` user, type `sudo su - hdfs`.

```
[biadmin@ibmclass ~]$ sudo su - hdfs
[sudo] password for biadmin:
[hdfs@ibmclass ~]$
```

or `su -` (the root password is `dalvm3`)

```
[biadmin@ibmclass ~]$ su -
Password:
[root@ibmclass ~]# su - hdfs
[hdfs@ibmclass ~]$
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

2. Type the following command: `hdfs fsck /`

In your lab environment, you may get a number of errors related to replication. Ignore those, as you are running a pseudo-distributed cluster and this replication is not standard.

The results that you should see will be similar to the following:

```
...
Status: HEALTHY
Total size:    713494586 B (Total open files size: 340 B)
Total dirs:    12748
Total files:   341
Total symlinks:          0 (Files currently being written: 3)
Total blocks (validated): 330 (avg. block size 2162104 B) (Total open file blocks
(not validated): 3)
Minimally replicated blocks: 330 (100.0 %)
Over-replicated blocks:    0 (0.0 %)
Under-replicated blocks:   330 (100.0 %)
Mis-replicated blocks:     0 (0.0 %)
Default replication factor: 3
Average block replication: 1.0
Corrupt blocks:           0
Missing replicas:         660 (66.666664 %)
Number of data-nodes:      1
Number of racks:          1
FSCK ended at Thu Jun 04 11:11:43 GMT-05:00 2015 in 218 milliseconds
```

The filesystem under path '/' is HEALTHY

3. Close all open windows.

Results:

You used basic Hadoop Distributed File System (HDFS) file system commands, moving some data into HDFS that will be used in later units of this course.

Unit 9 MapReduce and YARN

IBM Training



MapReduce and YARN

IBM BigInsights v4.0

© Copyright IBM Corporation 2015
Course materials may not be reproduced in whole or in part without the written permission of IBM.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit objectives

- Describe the MapReduce model v1
- List the limitations of Hadoop 1 and MapReduce 1
- Review the Java code required to handle the Mapper class, the Reducer class, and the program driver needed to access MapReduce
- Describe the YARN model
- Compare Hadoop 2/YARN with Hadoop 1

Topic: Introduction to MapReduce processing based on MR1

MapReduce and Yarn

© Copyright IBM Corporation 2015

Topic: Introduction to MapReduce processing based on MR1

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Agenda

- **MapReduce Introduction**
- MapReduce Tasks
- WordCount Example
- Splits
- Execution



MapReduce and Yarn

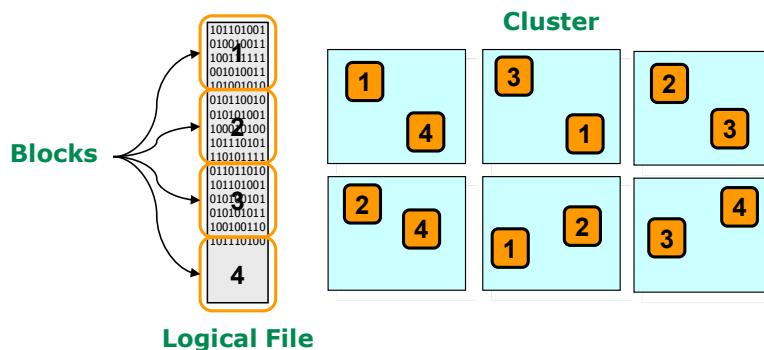
© Copyright IBM Corporation 2015

Agenda

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

MapReduce: the Distributed File System (DFS)

- Driving principles
 - data is stored across the entire cluster
 - programs are brought to the data, not the data to the program
- Data is stored across the entire cluster (the DFS)
 - the entire cluster participates in the file system
 - blocks of a single file are distributed across the cluster
 - a given block is typically replicated as well for resiliency



MapReduce and Yarn

© Copyright IBM Corporation 2015

MapReduce: the Distributed File System

The driving principle of MapReduce is a simple one: spread your data out across a huge cluster of machines and then, rather than bringing the data to your programs as you do in traditional programming, you write your program in a specific way that allows the program to be moved to the data. Thus, the entire cluster is made available in both reading the data as well as processing the data.

A Distributed File System (DFS) is at the heart of MapReduce. It is responsible for spreading data across the cluster, by making the entire cluster look like one giant file system. When a file is written to the cluster, blocks of the file are spread out and replicated across the whole cluster (in the diagram, notice that every block of the file is replicated to three different machines).

Adding more nodes to the cluster instantly adds capacity to the file system and, as we'll see on the next slide, automatically increases the available processing power and parallelism.

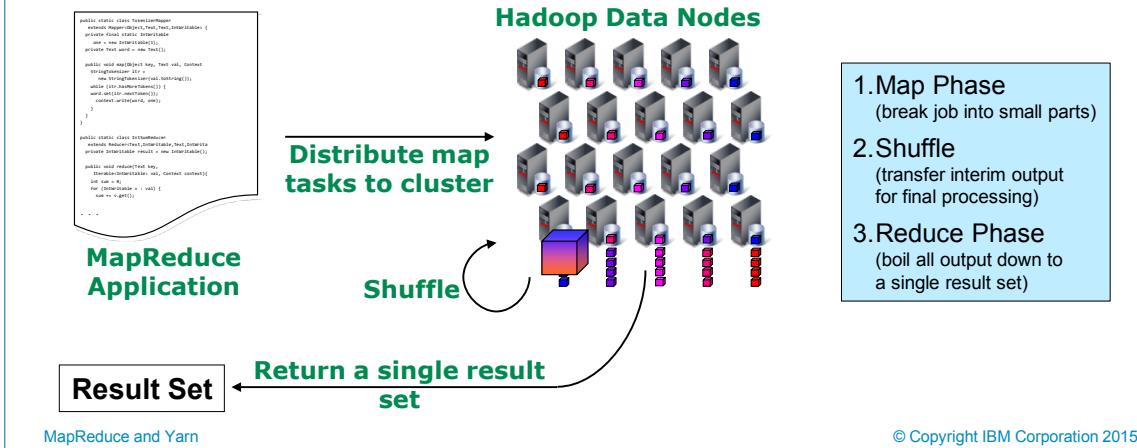
This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

MapReduce v1 explained

- **Hadoop computational model**

- Data stored in a distributed file system spanning many inexpensive computers
- Bring function to the data
- Distribute application to the compute resources where the data is stored

- **Scalable to thousands of nodes and petabytes of data**



MapReduce v1 explained

There are two aspects of Hadoop that are important to understand:

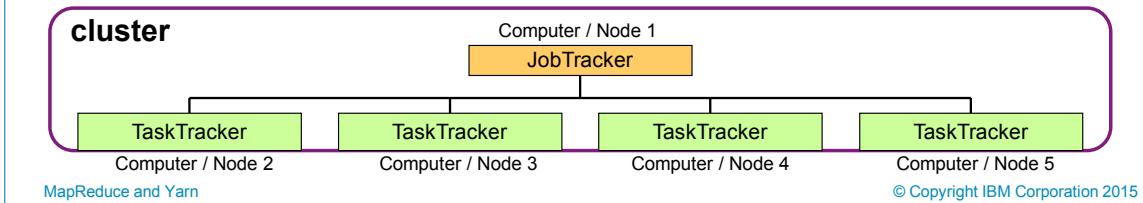
- MapReduce is a software framework introduced by Google to support distributed computing on large data sets of clusters of computers.
- The Hadoop Distributed File System (HDFS) is where Hadoop stores its data. This file system spans all the nodes in a cluster. Effectively, HDFS links together the data that resides on many local nodes, making the data part of one big file system. Furthermore, HDFS assumes nodes will fail, so it replicates a given chunk of data across multiple nodes to achieve reliability. The degree of replication can be customized by the Hadoop administrator or programmer. However, by default is to replicate every chunk of data across 3 nodes: 2 on the same rack, and 1 on a different rack.

The key to understanding Hadoop lies in the MapReduce programming model. This is essentially a representation of the divide and conquer processing model, where your input is split into many small pieces (the map step), and the Hadoop nodes process these pieces in parallel. Once these pieces are processed, the results are distilled (in the reduce step) down to a single answer.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

MapReduce v1 engine

- **Master/Slave architecture**
 - Single master (JobTracker) controls job execution on multiple slaves (TaskTrackers).
- **JobTracker**
 - Accepts MapReduce jobs submitted by clients
 - Pushes map and reduce tasks out to TaskTracker nodes
 - Keeps the work as physically close to data as possible
 - Monitors tasks and TaskTracker status
- **TaskTracker**
 - Runs map and reduce tasks
 - Reports status to JobTracker
 - Manages storage and transmission of intermediate output



MapReduce v1 engine

If one TaskTracker is very slow, it can delay the entire MapReduce job, especially towards the end of a job, where everything can end up waiting for the slowest task. With speculative-execution enabled, however, a single task can be executed on multiple slave nodes.

For jobs scheduling, by default Hadoop uses FIFO (First in, First Out), and optional 5 scheduling priorities to schedule jobs from a work queue. Other scheduling algorithms are available as add-ins: Fair Scheduler, Capacity Scheduler, and so on.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

The MapReduce programming model

- "Map" step
 - Input is split into pieces (HDFS blocks or "splits")
 - Worker nodes process the individual pieces in parallel (under global control of a Job Tracker)
 - Each worker node stores its result in its local file system where a reducer is able to access it
- "Reduce" step
 - Data is aggregated ("reduced" from the map steps) by worker nodes (under control of the Job Tracker)
 - Multiple reduce tasks parallelize the aggregation
 - Output is stored in HDFS (and thus replicated)

The MapReduce programming model

From Wikipedia on MapReduce (<http://en.wikipedia.org/wiki/MapReduce>):

MapReduce is a framework for processing huge datasets on certain kinds of distributable problems using a large number of computers (nodes), collectively referred to as a cluster (if all nodes use the same hardware) or as a grid (if the nodes use different hardware). Computational processing can occur on data stored either in a file system (unstructured) or within a database (structured).

"Map" step: The master node takes the input, chops it up into smaller sub-problems, and distributes those to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes that smaller problem, and passes the answer back to its master node.

"Reduce" step: The master node then takes the answers to all the sub-problems and combines them in some way to get the output - the answer to the problem it was originally trying to solve.

The MapReduce execution environments

- APIs vs. Execution Environment
 - APIs are implemented by applications and are largely independent of execution environment
 - Execution Environment defines how MapReduce jobs are executed
- MapReduce APIs
 - org.apache.mapred:
 - Old API, largely superseded some classes still used in new API
 - Not changed with YARN
 - org.apache.mapreduce:
 - New API, more flexibility, widely used
 - Applications may have to be recompiled to use YARN (not binary compatible)
- Execution Environments
 - Classic JobTracker/TaskTracker from Hadoop v1
 - YARN (MapReduce v2): Flexible execution environment to run MapReduce and much more
 - No single JobTracker, instead ApplicationMaster jobs for every application

The MapReduce execution environments

There are two aspects of Hadoop that are important to understand:

1. MapReduce is a software framework introduced by Google to support distributed computing on large data sets of clusters of computers. I'll explain more about MapReduce in a minute.
2. The Hadoop Distributed File System (HDFS) is where Hadoop stores its data. This file system spans all the nodes in a cluster. Effectively, HDFS links together the data that resides on many local nodes, making the data part of one big file system. I'll explain more about HDFS in a minute, too. You can use other file systems with Hadoop, but HDFS is quite common.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Agenda

- MapReduce introduction
- **MapReduce phases**
 - Map
 - Shuffle
 - Reduce
 - Combiner
- WordCount example
- Splits
- Execution



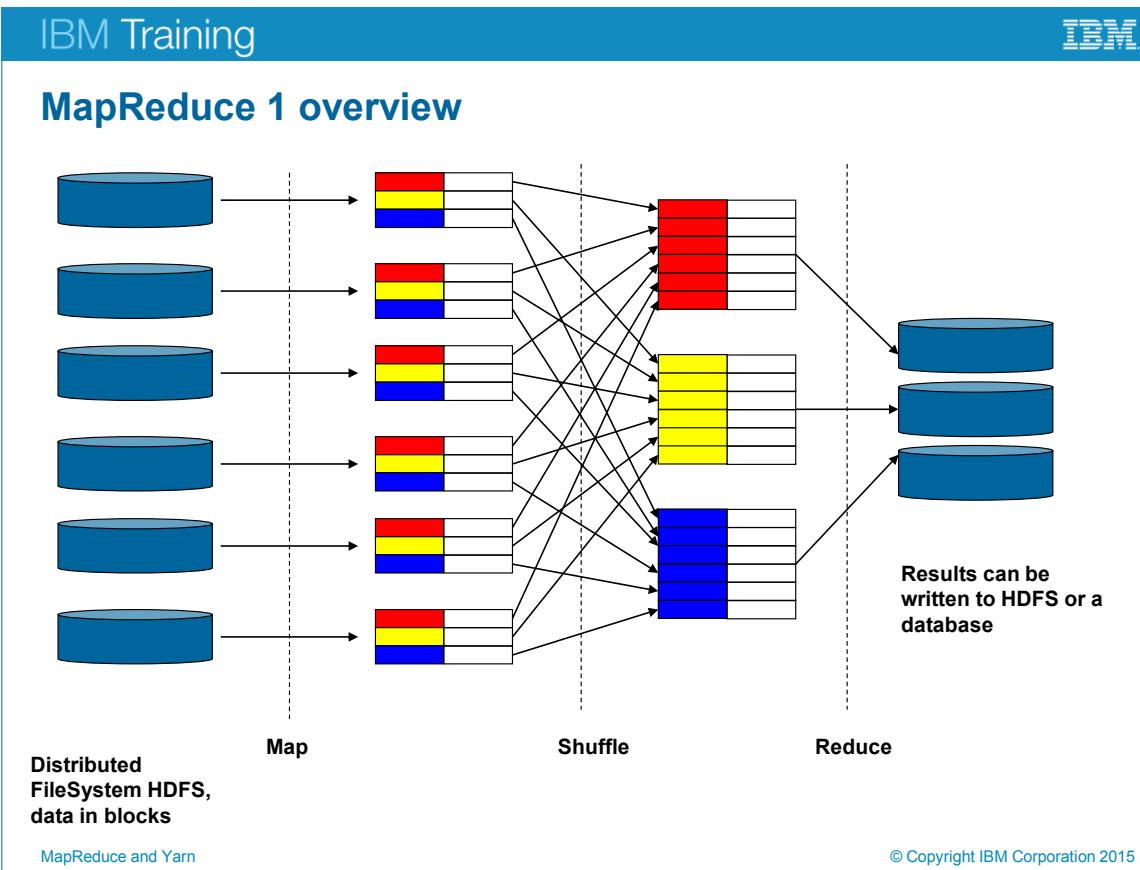
MapReduce and Yarn

© Copyright IBM Corporation 2015

Agenda

You will review the MapReduce phases: Map, Shuffle, and Reduce. In addition, there is an optional Combiner phase.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE



MapReduce 1 overview

This slide provides an overview of the MR1 process.

File blocks (stored on different DataNodes) in HDFS are read and processed by the Mappers.

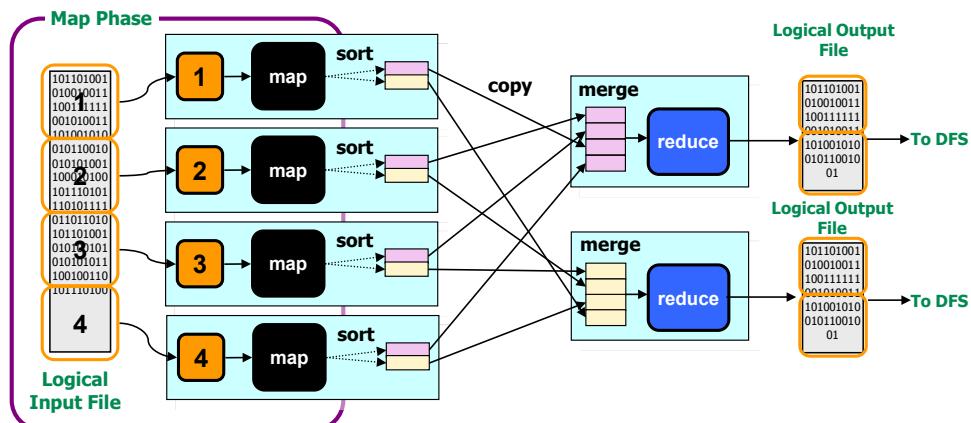
The output of the Mapper processes are shuffled (sent) to the Reducers (one output file from each Mapper to each Reducer); the files here are not replicated and are stored local to the Mapper node.

The Reducers produce the output and that output is stored in HDFS, with one file for each Reducer.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

MapReduce: Map Phase

- Mappers
 - Small program (typically), distributed across the cluster, local to data
 - Handed a *portion* of the input data (called a split)
 - Each mapper parses, filters, or transforms its input
 - Produces grouped <key, value> pairs



MapReduce: Map Phase

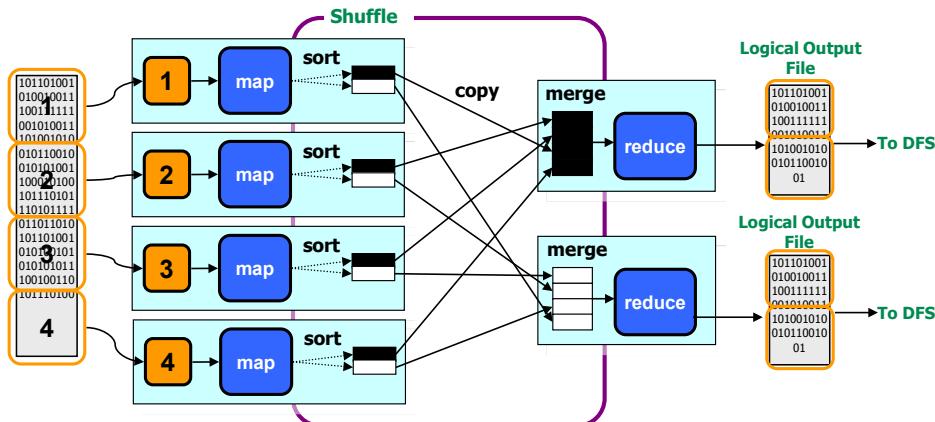
Earlier, you learned that if you write your programs in a special way, the programs can be brought to the data. This special way is called MapReduce, and it involves breaking your program down into two discrete parts: Map and Reduce.

A mapper is typically a relatively small program with a relatively simple task: it is responsible for reading a portion of the input data, interpreting, filtering or transforming the data as necessary and then finally producing a stream of <key, value> pairs. What these keys and values are is not of importance in the scope of this topic, but just keep in mind that these values can be as big and complex as you need.

Notice in the diagram, how the MapReduce environment will automatically take care of taking your small "map" program (the black boxes) and pushing that program out to every machine that has a block of the file you are trying to process. This means that the bigger the file, the bigger the cluster, the more mappers get involved in processing the data. That's a pretty powerful idea.

MapReduce: Shuffle Phase

- The output of each mapper is locally grouped together by **key**
- One node is chosen to process data for each unique **key**
- All of this movement (shuffle) of data is transparently orchestrated by MapReduce



MapReduce and Yarn

© Copyright IBM Corporation 2015

MapReduce: Shuffle Phase

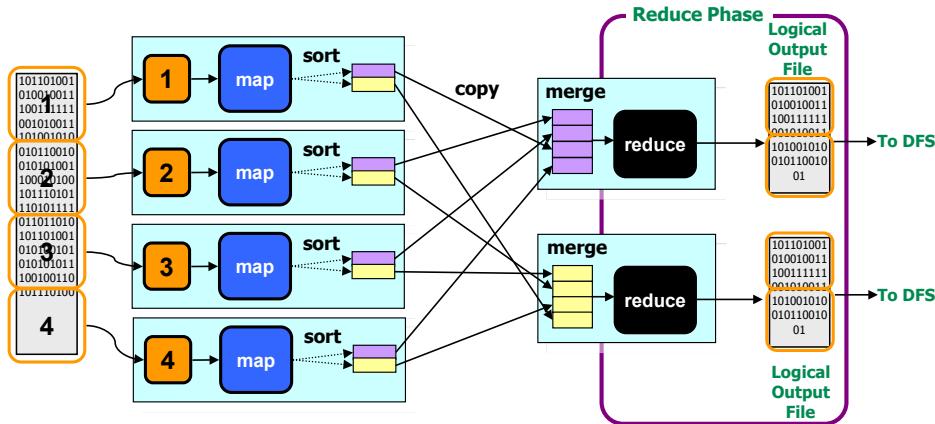
This next phase is called "Shuffle" and is orchestrated behind the scenes by MapReduce.

The idea here is that all of the data that is being emitted from the mappers is first locally grouped by the <key> that your program chose, and then for each unique key, a node is chosen to process all of the values from all of the mappers for that key.

For example, if you used U.S. state (such as MA, AK, NY, CA, etc.) as the key of your data, then one machine would be chosen to send all of the California data to, and another chosen for all of the New York data, and so on. Each machine would be responsible for processing the data for its selected state. In the diagram above, the data clearly only had two keys (shown as white and black boxes), but keep in mind that there may be many records with the same key coming from a given mapper.

MapReduce: Reduce Phase

- Reducers
 - Small programs (typically) that aggregate all of the values for the key that they are responsible for
 - Each reducer writes output to its own file



MapReduce and Yarn

© Copyright IBM Corporation 2015

MapReduce: Reduce Phase

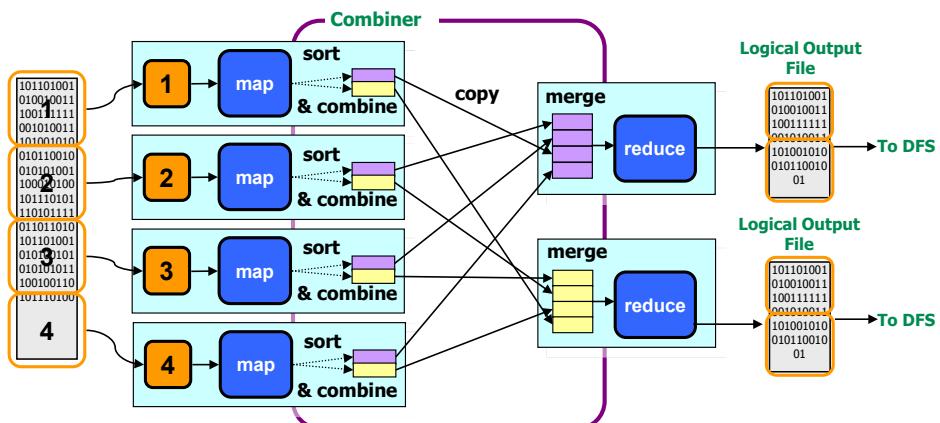
Reducers are the last part of the picture. Again, these are small programs (typically small) that are responsible for sorting and/or aggregating all of the values with the key that was assigned to work on. Just like with mappers, the more unique keys you have the more parallelism.

Once each reducer has completed whatever is assigned to do, such as add up the total sales for the state it was assigned, and it in turn, emits key/value pairs that get written to storage that can then be used as the input to the next MapReduce job.

This is a simplified overview of MapReduce.

MapReduce: Combiner (Optional)

- The data that will go to each reduce node is sorted and merged before going to the reduce node, pre-doing some of the work of the receiving reduce node in order to minimize network traffic between map and reduce nodes.



MapReduce: Combiner (Optional)

At the same time as the Sort is done during the Shuffle work on the Mapper node, an optional Combiner function may be applied.

For each key, all key/values with that key are sent to the same Reducer node (that is the purpose of the Shuffle phase).

Rather than sending multiple key/value pairs with the same key value to the Reducer node, the values are combined into one key/value pair. This is only possible where the reduce function is additive (or does not lose information when combined).

Since only one key/value pair is sent, the file transferred from Mapper node to Reducer node is smaller and network traffic is minimized.

Agenda

- MapReduce introduction
- MapReduce tasks
- **WordCount example**
- Splits
- Execution



MapReduce and Yarn

© Copyright IBM Corporation 2015

Agenda

You will review wordcount under MR v1 and how the running job differs under YARN (including the similarities).

Later, you will review the Java code for the Java wordcount program to foster a brief discussion of the major parts of the program from a code perspective (thus illustrating Map, Combiner, Reduce).

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

WordCount example

- In the example of a list of animal names
 - MapReduce can automatically split files on line breaks
 - The file has been split into two blocks on two nodes
- To count how often each big cat is mentioned, in SQL you would use:

```
SELECT COUNT(NAME) FROM ANIMALS
WHERE NAME IN (Tiger, Lion ...)
GROUP BY NAME;
```

Node 1

Tiger
Lion
Lion
Panther
Wolf
...

Node 2

Tiger
Tiger
Wolf
Panther
...

WordCount example

In a file with two blocks (or "splits") of data, animal names are listed. There is one animal name per line in the files.

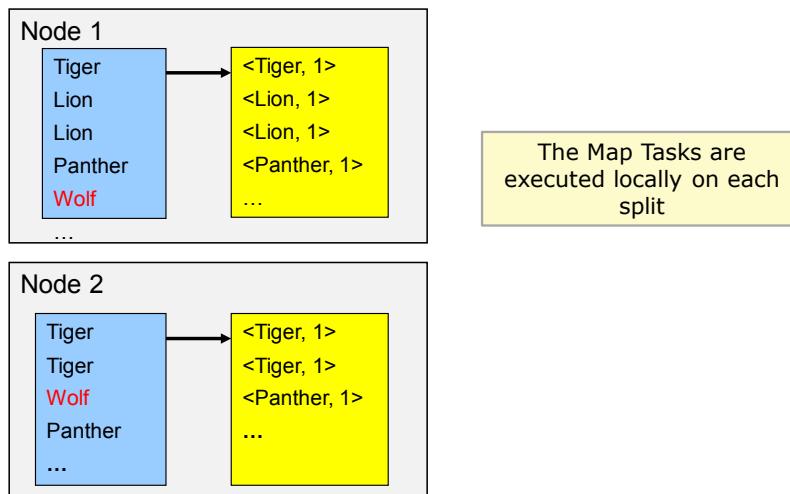
Rather than count the number of mentions of each animal, you are interested only in members of the cat family.

Since the blocks are held on different nodes, software running on the individual nodes process the blocks separately.

If you were using SQL, which is not used, the SQL would be as stated in the slide.

Map Task

- There are two requirements for the Map task:
 - filter out the non big-cat rows
 - prepare count by transforming to <Text(name), Integer(1)>



MapReduce and Yarn

© Copyright IBM Corporation 2015

Map Task

Reviewing the description of MapReduce from Wikipedia (<http://en.wikipedia.org/wiki/MapReduce>):

MapReduce is a framework for processing huge datasets on certain kinds of distributable problems using a large number of computers (nodes), collectively referred to as a cluster (if all nodes use the same hardware) or as a grid (if the nodes use different hardware). Computational processing can occur on data stored either in a file system (unstructured) or within a database structured).

"Map" step: The master node takes the input, breaks it up into smaller sub-problems, and distributes those to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes that smaller problem, and passes the answer back to its master node.

"Reduce" step: The master node then takes the answers to all the sub-problems and combines them in some way to get the output, the answer to the problem it was originally trying to solve.

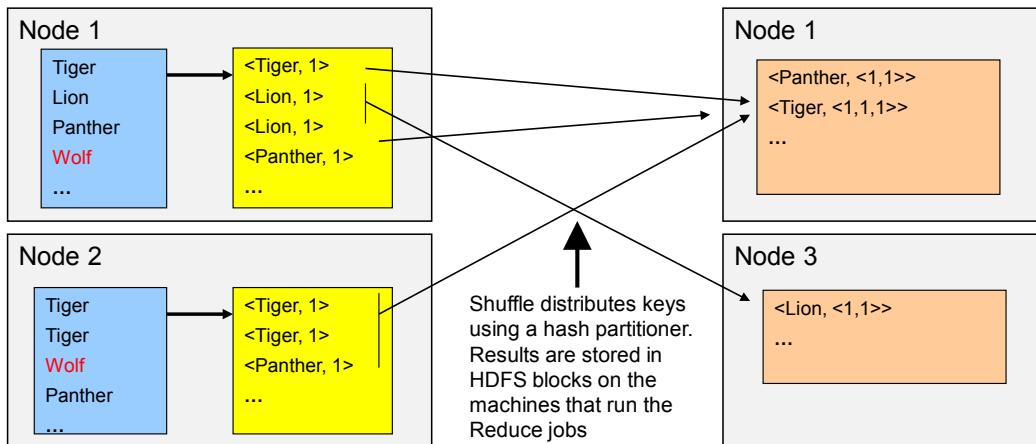
The Map step shown does the following processing:

- each Map node reads its own "split" (block) of data
- the information required (in this case, the names of animals) is extracted from each record (in this case, one line = one record)
- data is filtered (keeping only the names of cat family animals)
- key-value pairs are created (in this case, key = animal and value = 1)
- key-value pairs are accumulated into locally stored files on the individual nodes where the Map task is being executed

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Shuffle

- Shuffle moves all values of one key to the same target node
- Distributed by a Partitioner Class (normally hash distribution)
- Reduce Tasks can run on any node - here on Nodes 1 and 3
 - The number of Map and Reduce tasks do not need to be identical
 - Differences are handled by the hash partitioner



MapReduce and Yarn

© Copyright IBM Corporation 2015

Shuffle

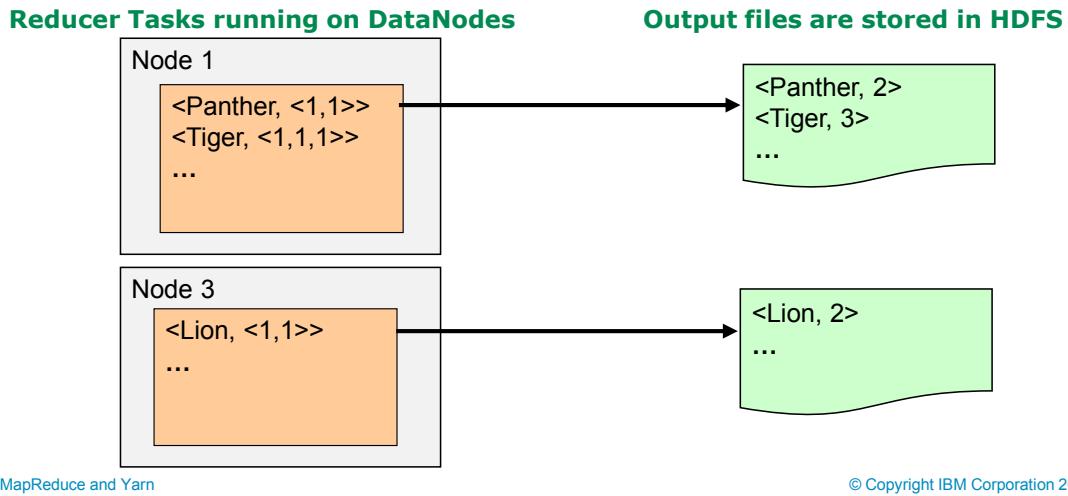
Shuffle distributes the key-value pairs to the nodes where the Reducer task will run. Each Mapper task produces one file for each Reducer task. A hash function running on the Mapper node determines which Reducer task will receive any particular key-value pair. All key-value pairs with a particular key will be sent to the same Reducer task.

Reduce tasks can run on any node, either different from the set of nodes where the Map task run or on the same DataNodes. In the slide example, Node 1 is used for one Reduce task, but a new node, Node 3, is used for a second Reduce node.

There is no relation between the number of Map tasks (generally one node for each block of the file(s) begin read) and the number of Reduce tasks. Commonly the number of Reduce tasks is smaller than the number of Map tasks.

Reduce

- The reduce task computes aggregated values for each key
 - Normally the output is written to the DFS
 - Default is one output part-file per Reduce task
 - Reduce tasks aggregate all values of a specific key - our case, the count of the particular animal type



Reduce

Note that these two Reducer tasks are running on Nodes 1 and 3.

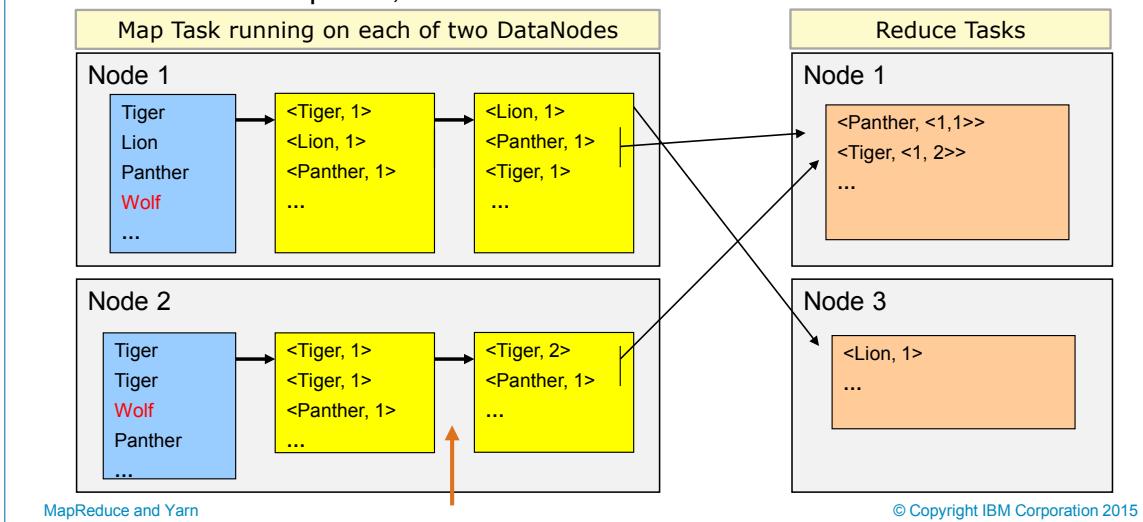
The Reduce node then takes the answers to all the sub-problems and combines them in some way to get the output - the answer to the problem it was originally trying to solve.

In this case, the Reduce step shown on this slide does the following processing:

- Each Reduce node receives data sent to it from the various Map nodes
- This data has been previously sorted (and possibly partially merged)
- The Reduce node aggregates the data; in the case of wordcount, it sums the counts received for each particular word (each animal in this case)
- One file is produced for each Reduce task and that is written to HDFS where the blocks will automatically be replicated

Optional: Combiner

- For performance, an aggregate in the Map task can be helpful
- Reduces the amount of data sent over the network
 - Also reduces Merge effort, since data is premerged in Map
 - Done in the Map task, before Shuffle



Optional: Combiner

The Combiner phase is optional. When it is used, it runs on the Mapper node and preprocesses the data that will be sent to Reduce tasks by pre-merging and pre-aggregating the data in the files that will be transmitted to the Reduce tasks.

The Combiner thus reduces the amount of data that will be sent to the Reducer tasks, and that speeds up the processing as smaller files need to be transmitted to the Reducer task nodes.

Source code for WordCount.java (1 of 3)

```

1. package org.myorg;
2.
3. import java.io.IOException;
4. import java.util.*;
5.
6. import org.apache.hadoop.fs.Path;
7. import org.apache.hadoop.conf.*;
8. import org.apache.hadoop.io.*;
9. import org.apache.hadoop.mapred.*;
10. import org.apache.hadoop.util.*;
11.
12. public class WordCount {
13.
14.     public static class Map extends MapReduceBase implements Mapper<LongWritable, Text,
15.                                         Text, IntWritable> {
16.         private final static IntWritable one = new IntWritable(1);
17.         private Text word = new Text();
18.
19.         public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
20.                         output, Reporter reporter) throws IOException {
21.             String line = value.toString();
22.             StringTokenizer tokenizer = new StringTokenizer(line);
23.             while (tokenizer.hasMoreTokens()) {
24.                 word.set(tokenizer.nextToken());
25.                 output.collect(word, one);
26.             }
27.         }
28.     }
29. }
```

MapReduce and Yarn

© Copyright IBM Corporation 2015

Source code for WordCount.java

This is a slightly simplified version of WordCount.java for MapReduce 1. The full program is slightly larger, and there are some recommended differences for compiling for MapReduce 2 with Hadoop 2.

Code from the Hadoop classes is brought in with the import statements. Like an iceberg, most of the actual code executed at runtime is hidden from the programmer; it runs deep down in the Hadoop code itself.

The interest here is the Mapper class, **Map**.

This class reads the file (you will see on the driver class slide later as `arg[0]`) as a string. The string is tokenized, for example, broken into words separated by space(s).

Note the following shortcomings of the code:

- No lowercasing is done, thus **The** and **the** are treated as separate words that will be counted separately.
- Any adjacent punctuation is appended to the word, thus "**the**" and **the** will be counted separately, and any word followed by punctuation, for example **cow,** is counted separately from **cow** (the same word without trailing punctuation).

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

You will see these shortcomings in the output. Note that this is the standard wordcount program and the interest is not in the actual results but only in the process at this stage.

The wordcount program is to Hadoop Java programs functions as the "Hello, world!" program is to the C language. It is generally the first program that people experience when coming to the new technology.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Source code for WordCount.java (2 of 3)

```
28.  public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable,  
29.    Text, IntWritable> {  
30.      public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,  
31.        IntWritable> output, Reporter reporter) throws IOException {  
32.          int sum = 0;  
33.          while (values.hasNext()) {  
34.            sum += values.next().get();  
35.          }  
36.        }  
37.      }
```

The Reducer class, **Reduce**, is displayed in the slide.

The key-value pairs arrive at this class already sorted (courtesy of the core Hadoop classes that you do not see), thus adjacent records will have the same key.

While the key does not change, the values are aggregated (in this case, summed) using: **sum += ...**

Source code for WordCount.java (3 of 3)

```

38. public static void main(String[] args) throws Exception {
39.     JobConf conf = new JobConf(WordCount.class);
40.     conf.setJobName("wordcount");
41.
42.     conf.setOutputKeyClass(Text.class);
43.     conf.setOutputValueClass(IntWritable.class);
44.
45.     conf.setMapperClass(Map.class);
46.     conf.setCombinerClass(Reduce.class);
47.     conf.setReducerClass(Reduce.class);
48.
49.     conf.setInputFormat(TextInputFormat.class);
50.     conf.setOutputFormat(TextOutputFormat.class);
51.
52.     FileInputFormat.setInputPaths(conf, new Path(args[0]));
53.     FileOutputFormat.setOutputPath(conf, new Path(args[1]));
54.
55.     JobClient.runJob(conf);
56. }
57. }
58. }
```

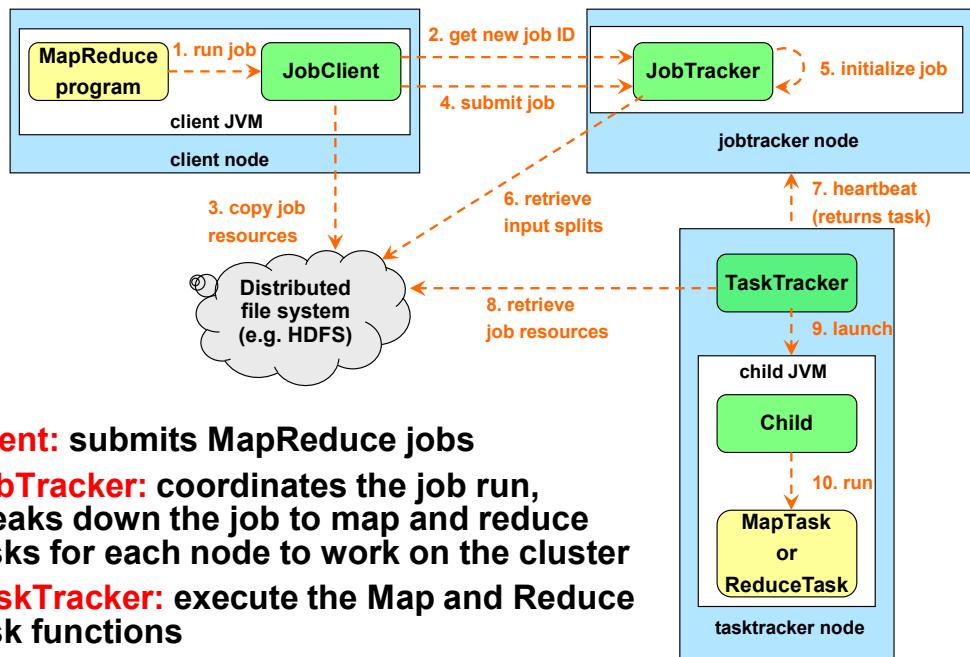
The driver routine, embedded in main, does the following work:

- sets the JobName for runtime
- sets the Mapper class to Map
- sets the Reducer class to Reduce
- sets the Combiner class to Reduce
- sets the input file to arg[0]
- sets the output directory to arg[1]

The combiner runs on the Map task and use the same code as the Reducer task.

The names of the output files will be generated inside the Hadoop code.

How does Hadoop run MapReduce jobs?



MapReduce and Yarn

© Copyright IBM Corporation 2015

How does Hadoop run MapReduce jobs?

The process of running a MapReduce job on Hadoop consists of 10 major steps.

1. The first step is the MapReduce program you've written tells the Job Client to run a MapReduce job.
2. This sends a message to the JobTracker which produces a unique ID for the job.
3. The Job Client copies job resources, such as a jar file containing a Java code you have written to implement the map or the reduce task, to the shared file system, usually HDFS.
4. Once the resources are in HDFS, the Job Client can tell the JobTracker to start the job.
5. The JobTracker does its own initialization for the job. It calculates how to split the data so that it can send each "split" to a different mapper process to maximize throughput.
6. It retrieves these "input splits" from the distributed file system, not the data itself.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

7. The TaskTrackers are continually sending heartbeat messages to the JobTracker. Now that the JobTracker has work for them, it will return a map task or a reduce task as a response to the heartbeat.
8. The TaskTrackers need to obtain the code to execute, so they get it from the shared file system.
9. Then they can launch a Java Virtual Machine with a child process running in it and this child process runs your map code or your reduce code. The result of the map operation will remain in the local disk for the given TaskTracker node (not in HDFS).

The output of the Reduce task is stored in HDFS file system using the number of copies specified by replication factor.

Classes

- There are three main Java classes provided in Hadoop to read data in MapReduce:
 - **InputSplitter** dividing a file into splits
 - Splits are normally the block size but depends on number of requested Map tasks, whether any compression allows splitting, etc.
 - **RecordReader** takes a split and reads the files into records
 - For example, one record per line (LineRecordReader)
 - But note that a record can be split across splits
 - **InputFormat** takes each record and transforms it into a <key, value> pair that is then passed to the Map task
- Lots of additional helper classes may be required to handle compression, etc.
 - For example, IBM BigInsights provides additional compression handlers for LZO compression, etc.

Classes

The InputSplitter, RecordSplitter, and InputFormat classes are provided inside the Hadoop code.

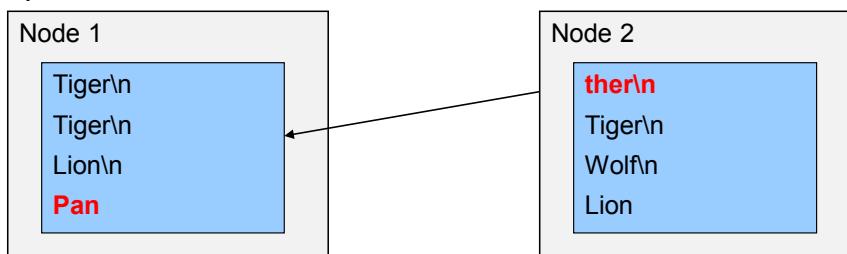
Other helper classes are needed to support Java MapReduce programs. Some of these are provided from inside the Hadoop code itself, but distribution vendors and end programmers can provide other classes that either override or supplement standard code. Thus some vendors provide the LZO compressions algorithm to supplement standard compression codecs (such as codecs for bzip2, etc).

Splits

- Files in MapReduce are stored in blocks (128 MB)
- MapReduce divides data into fragments or **splits**.
 - One map task is executed on each split
- Most files have records with defined **split points**
 - Most common is the end of line character
- The `InputSplitter` class is responsible for taking a HDFS file and transforming it into splits.
 - Aim is to process as much data as possible locally

RecordReader

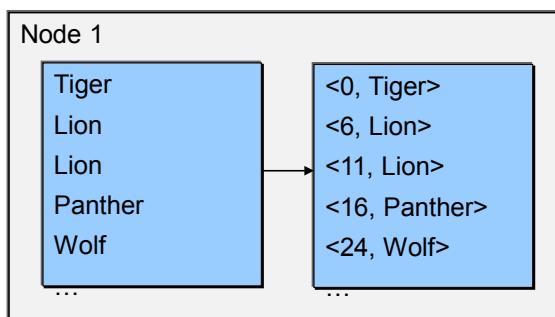
- Most of the time a split will not happen at a block end
- Files are read into Records by the RecordReader class
 - Normally the RecordReader will start and stop at the split points.
- LineRecordReader will read over the end of the split till the line end.
 - HDFS will send the missing piece of the last record over the network
- Likewise LineRecordReader for Block2 will disregard the first incomplete line



In this example RecordReader1 will not stop at Pan but will read on until the end of the line. Likewise RecordReader2 will ignore the first line.

InputFormat

- MapReduce Tasks read files by defining an InputFormat class
 - Map tasks expect <key, value> pairs
- To read line-delimited textfiles Hadoop provides the TextInputFormat class
 - It returns one key, value pair per line in the text
 - The value is the content of the line
 - The key is the character offset to the new line character (end of line)



MapReduce and Yarn

© Copyright IBM Corporation 2015

InputFormat

InputFormat describes the input-specification for a Map-Reduce job.

The Map-Reduce framework relies on the *InputFormat* of the job to:

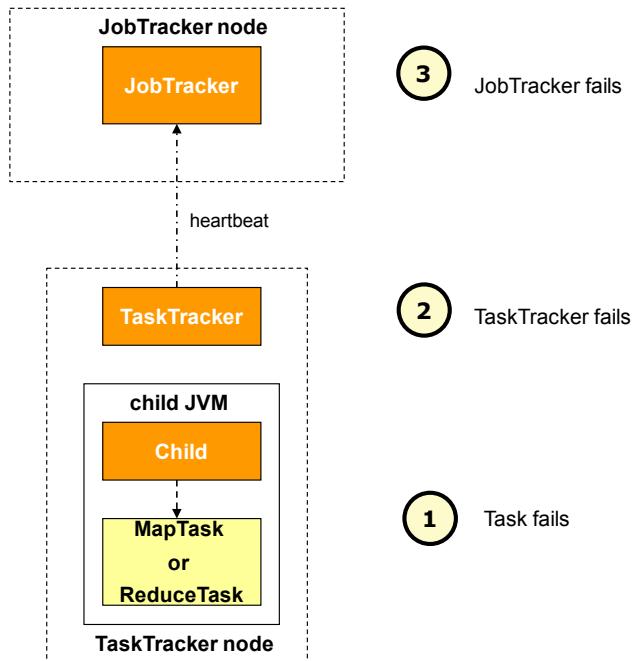
- Validate the input-specification of the job.
- Split-up the input file(s) into logical *InputSplits*, each of which is then assigned to an individual Mapper.
- Provide the *RecordReader* implementation to be used to glean input records from the logical *InputSplit* for processing by the Mapper.

The default behavior of file-based *InputFormats*, typically sub-classes of *FileInputFormat*, is to split the input into logical *InputSplits* based on the total size, in bytes, of the input files. However, the *FileSystem* blocksize of the input files is treated as an upper bound for input splits. A lower bound on the split size can be set via *mapreduce.input.fileinputformat.split.minsize*.

Clearly, logical splits based on input-size is insufficient for many applications since record boundaries are to be respected. In such cases, the application has to also implement a *RecordReader* on whom lies the responsibility to respect record-boundaries and present a record-oriented view of the logical *InputSplit* to the individual task.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Fault tolerance



MapReduce and Yarn

© Copyright IBM Corporation 2015

Fault tolerance

What happens when something goes wrong?

Failures can happen at the task level (1), TaskTracker level (2), or JobTracker level (3).

The primary way that Hadoop achieves fault tolerance is through restarting tasks. Individual task nodes (TaskTrackers) are in constant communication with the head node of the system, called the JobTracker. If a TaskTracker fails to communicate with the JobTracker for a period of time (by default, 1 minute), the JobTracker will assume that the TaskTracker in question has crashed. The JobTracker knows which map and reduce tasks were assigned to each TaskTracker.

If the job is still in the mapping phase, then other TaskTrackers will be asked to re-execute all map tasks previously run by the failed TaskTracker. If the job is in the reducing phase, then other TaskTrackers will re-execute all reduce tasks that were in progress on the failed TaskTracker.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Reduce tasks, once completed, have been written back to HDFS. Thus, if a TaskTracker has already completed two out of three reduce tasks assigned to it, only the third task must be executed elsewhere. Map tasks are slightly more complicated: even if a node has completed ten map tasks, the reducers may not have all copied their inputs from the output of those map tasks. If a node has crashed, then its mapper outputs are inaccessible. So any already-completed map tasks must be re-executed to make their results available to the rest of the reducing machines. All of this is handled automatically by the Hadoop platform.

This fault tolerance underscores the need for program execution to be side-effect free. If Mappers and Reducers had individual identities and communicated with one another or the outside world, then restarting a task would require the other nodes to communicate with the new instances of the map and reduce tasks, and the re-executed tasks would need to reestablish their intermediate state. This process is notoriously complicated and error-prone in the general case. MapReduce simplifies this problem drastically by eliminating task identities or the ability for task partitions to communicate with one another. An individual task sees only its own direct inputs and knows only its own outputs, to make this failure and restart process clean and dependable.

Topic: Issues with and limitations of Hadoop v1 and MapReduce v1

MapReduce and Yarn

© Copyright IBM Corporation 2015

Topic: Issues with and limitations of Hadoop v1 and MapReduce v1

The original Hadoop (v1) and MapReduce (v1) had limitations, and a number of issues surfaced over time. You will review these in preparation for looking at the differences and changes introduced with Hadoop 2 and MapReduce v2.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Issues with the original MapReduce paradigm

- Centralized handling of job control flow
- Tight coupling of a specific programming model with the resource management infrastructure
- Hadoop is now being used for all kinds of tasks beyond its original design

Issues with the original MapReduce paradigm

These will be reviewed in more detail in this unit.

Limitations of classic MapReduce (MRv1)

The most serious limitations of classical MapReduce are:

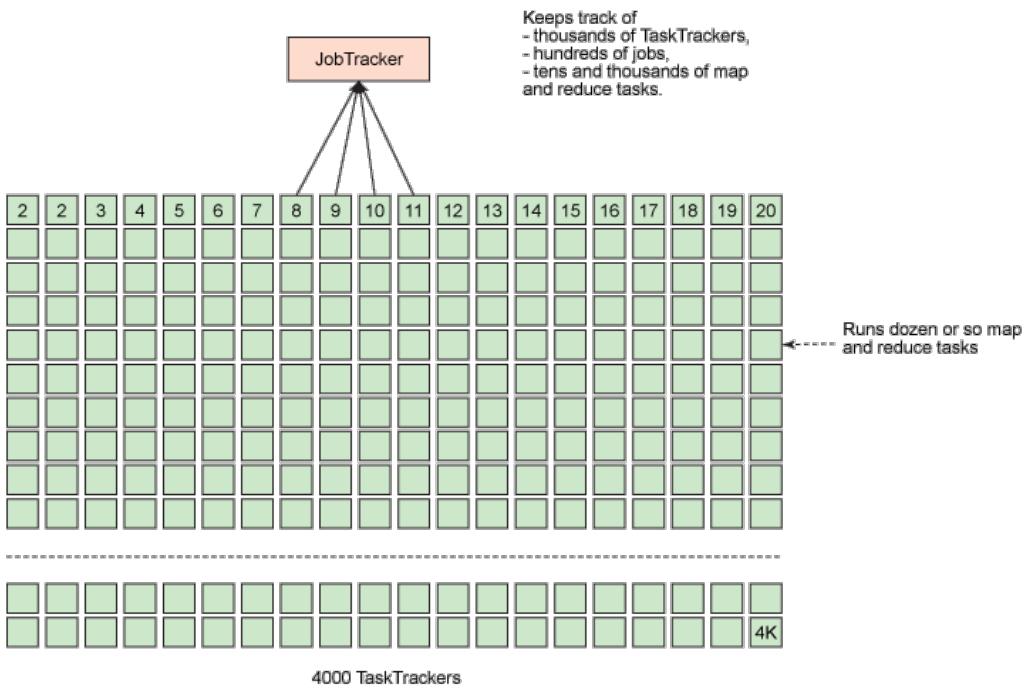
- Scalability
- Resource utilization
- Support of workloads different from MapReduce.
- In the MapReduce framework, the job execution is controlled by two types of processes:
 - A single master process called *JobTracker*, which coordinates all jobs running on the cluster and assigns map and reduce tasks to run on the *TaskTrackers*
 - A number of subordinate processes called *TaskTrackers*, which run assigned tasks and periodically report the progress to the *JobTracker*

Limitations of classic MapReduce (MRv1)

This topic is well covered in the article:

- *Introduction to YARN*: <http://www.ibm.com/developerworks/library/bd-yarn-intro>

Scalability in MRv1: Busy JobTracker



MapReduce and Yarn

© Copyright IBM Corporation 2015

Scalability in MRv1: Busy JobTracker

In Hadoop MapReduce, the JobTracker is charged with two distinct responsibilities:

- Management of computational resources in the cluster, which involves maintaining the list of live nodes, the list of available and occupied map and reduce slots, and allocating the available slots to appropriate jobs and tasks according to selected scheduling policy
- Coordination of all tasks running on a cluster, which involves instructing TaskTrackers to start map and reduce tasks, monitoring the execution of the tasks, restarting failed tasks, speculatively running slow tasks, calculating total values of job counters, and more

The large number of responsibilities given to a single process caused significant scalability issues, especially on a larger cluster where the JobTracker had to constantly keep track of thousands of TaskTrackers, hundreds of jobs, and tens of thousands of map and reduce tasks. The diagram represents this issue. On the contrary, the TaskTrackers usually run only a dozen or so tasks, which were assigned to them by the hard-working JobTracker.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

YARN overhauls MRv1

- MapReduce has undergone a complete overhaul with YARN, splitting up the two major functionalities of JobTracker (resource management and job scheduling/monitoring) into separate daemons
- **ResourceManager (RM)**
 - The global ResourceManager and per-node slave, the NodeManager (NM), form the data-computation framework
 - The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system
- **ApplicationMaster (AM)**
 - The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks
 - An application is either a single job in the classical sense of Map-Reduce jobs or a directed acyclic graph (DAG) of jobs

YARN overhauls MRv1

The fundamental idea of YARN/MRv2 is to split up the two major functionalities of the JobTracker, resource management and job scheduling/monitoring, into separate daemons. The idea is to have a global ResourceManager (RM) and per-application ApplicationMaster (AM). An application is either a single job in the classical sense of Map-Reduce jobs or a DAG of jobs.

The ResourceManager and per-node slave, the NodeManager (NM), form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

The ResourceManager has two main components: Scheduler and ApplicationsManager.

The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is a pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based on the resource requirements of the applications; it does so based on the abstract notion of a resource Container which incorporates elements such as memory, cpu, disk, network etc. In the first version, only memory is supported.

The Scheduler has a pluggable policy plug-in, which is responsible for partitioning the cluster resources among the various queues, applications etc. The current Map-Reduce schedulers such as the CapacityScheduler and the FairScheduler would be some examples of the plug-in.

The CapacityScheduler supports hierarchical queues to allow for more predictable sharing of cluster resources.

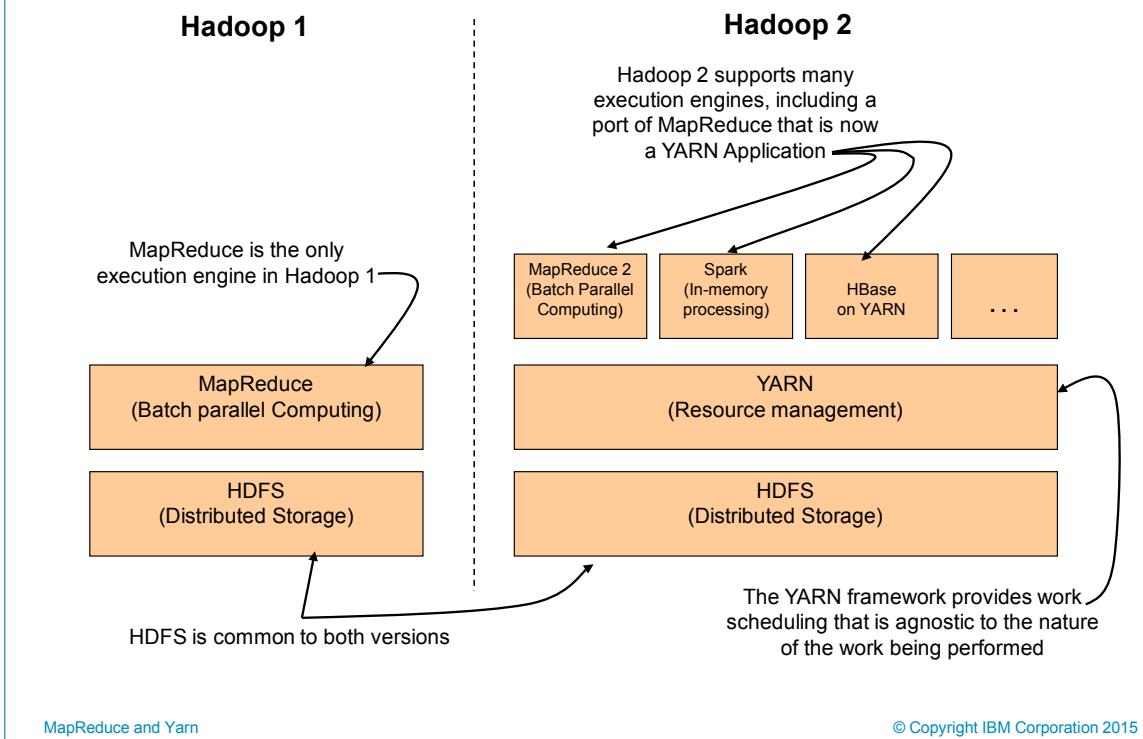
The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure.

The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.

The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

MRV2 maintains API compatibility with previous stable release (hadoop-1.x). This means that all Map-Reduce jobs should still run unchanged on top of MRv2 with just a recompile.

Hadoop 1 and 2 architectures compared



Hadoop 1 and 2 architectures compared

This diagram is developed from that in Alex Holmes, *Hadoop in Practice*, 2nd ed. (Shelter Island, NY: Manning, 2015).

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

YARN features

- Scalability
- Multi-tenancy
- Compatibility
- Serviceability
- Higher cluster utilization
- Reliability/Availability

YARN features: scalability

- There is one Application Master per job, which is why YARN scales better than the previous Hadoop v1 architecture
 - The Application Master for a given job can run on an arbitrary cluster node, and it runs until the job reaches termination
- Separation of functionality allows the individual operations to be improved with less effect on other operations
- YARN supports rolling upgrades without downtime

ResourceManager focuses exclusively on scheduling, allowing clusters to expand to thousands of nodes managing petabytes of data

YARN features: scalability

YARN lifts the scalability ceiling in Hadoop by splitting the roles of the Hadoop Job Tracker into two processes. A ResourceManager controls access to the clusters resources (memory, CPU, etc.), and the ApplicationManager (one per job) controls task execution.

YARN can run on larger clusters than MapReduce 1. MapReduce 1 hits scalability bottlenecks in the region of 4,000 nodes and 40,000 tasks, stemming from the fact that the JobTacker has to manage both jobs and tasks. YARN overcomes these limitations by virtue of its split ResourceManager/ApplicationMaster architecture: it is designed to scale up to 10,000 nodes and 100,000 tasks.

In contrast to the jobtracker, each instance of an application has a dedicated ApplicationMaster, which runs for the duration of the application. This model is actually closer to the original Google MapReduce paper, which describes how a master process is started to coordinate map and reduce tasks running on a set of workers.

YARN features: multi-tenancy

- YARN allows multiple access engines (either open-source or proprietary) to use Hadoop as the common standard for batch, interactive, and real-time engines that can simultaneously access the same data sets
- YARN uses a shared pool of nodes for all jobs
- YARN allows the allocation of Hadoop clusters of fixed size from the shared pool

Multi-tenant data processing improves an enterprise's return on its Hadoop investment

YARN features: multi-tenancy

Multi-tenancy generally refers to a set of features that enable multiple business users and processes to share a common set of resources, such as an Apache Hadoop cluster via policy rather than physical separation, yet without negatively impacting service level agreements (SLA), violating security requirements, or even revealing the existence of each party.

What YARN does is essentially de-couple Hadoop workload management from resource management. This means that multiple applications can share a common infrastructure pool. While this idea is not new to many, it is new to Hadoop. Earlier versions of Hadoop consolidated both workload and resource management functions into a single JobTracker. This approach resulted in limitations for customers hoping to run multiple applications on the same cluster infrastructure.

To borrow from object-oriented programming terminology, multi-tenancy is an over-loaded term. It means different things to different people depending on their orientation and context. To say a solution is multi-tenant is not helpful unless being specific about the meaning. Some interpretations of multi-tenancy in big data environments are:

- Support for multiple concurrent Hadoop jobs
- Support for multiple lines of business on a shared infrastructure
- Support for multiple application workloads of different types (Hadoop and non-Hadoop)
- Provisions for security isolation between tenants
- Contract-oriented service level guarantees for tenants
- Support for multiple versions of applications and application frameworks concurrently

Organizations that are sophisticated in their view of multi-tenancy will need all of these capabilities and more. YARN promises to address some of these requirements, and does so in large measure. But, you will find in future releases of Hadoop that there will be other approaches that are being addressed to provide other forms of multi-tenancy.

While an important technology, the world is not suffering from a shortage of resource managers. Some Hadoop providers (including IBM) are supporting YARN, while others are supporting Apache Mesos.

YARN features: compatibility

- To the end user (a developer, not an administrator), the changes are almost invisible
- Possible to run unmodified MapReduce jobs using the same MapReduce API and CLI
 - May require a recompile

There is no reason not to migrate from MRv1 to YARN

YARN features: compatibility

To ease the transition from Hadoop v1 to YARN, a major goal of YARN and the MapReduce framework implementation on top of YARN was to ensure that existing MapReduce applications that were programmed and compiled against previous MapReduce APIs (we'll call these MRv1 applications) can continue to run with little or no modification on YARN (we'll refer to these as MRv2 applications).

For the vast majority of users who use the org.apache.hadoop.mapred APIs, MapReduce on YARN ensures full binary compatibility. These existing applications can run on YARN directly without recompilation. You can use .jar files from your existing application that code against mapred APIs, and use bin/hadoop to submit them directly to YARN.

Unfortunately, it was difficult to ensure full binary compatibility to the existing applications that compiled against MRv1 org.apache.hadoop.mapreduce APIs. These APIs have gone through many changes. For example, several classes stopped being abstract classes and changed to interfaces. Therefore, the YARN community compromised by only supporting source compatibility for org.apache.hadoop.mapreduce APIs. Existing applications that use MapReduce APIs are source-compatible and can run on YARN either with no changes, with simple recompilation against MRv2 .jar files that are shipped with Hadoop 2, or with minor updates.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

YARN features: higher cluster utilization

- Higher cluster utilization, whereby resources not used by one framework can be consumed by another
- The NodeManager is a more generic and efficient version of the TaskTracker.
 - Instead of having a fixed number of map and reduce slots, the NodeManager has a number of dynamically created resource containers
 - The size of a container depends upon the amount of resources assigned to it, such as memory, CPU, disk, and network IO

YARN's dynamic allocation of cluster resources improves utilization over the more static MapReduce rules used in early versions of Hadoop (v1)

YARN features: higher cluster utilization

The NodeManager is a more generic and efficient version of the TaskTracker. Instead of having a fixed number of map and reduce slots, the NodeManager has a number of dynamically created resource containers. The size of a container depends upon the amount of resources it contains, such as memory, CPU, disk, and network IO.

Currently, only memory and CPU are supported (YARN-3); cgroups might be used to control disk and network IO in the future.

The number of containers on a node is a product of configuration parameters and the total amount of node resources (such as total CPUs and total memory) outside the resources dedicated to the slave daemons and the OS.

YARN features: reliability and availability

- High availability for the ResourceManager
 - An application recovery is performed after the restart of ResourceManager
 - The ResourceManager stores information about running applications and completed tasks in HDFS
 - If the ResourceManager is restarted, it recreates the state of applications and re-runs only incomplete tasks
- Highly available NameNode, making the Hadoop cluster much more efficient, powerful, and reliable

High Availability is work in progress, and is close to completion - features have been actively tested by the community

YARN major features summarized

- Multi-tenancy
 - YARN allows multiple access engines (either open-source or proprietary) to use Hadoop as the common standard for batch, interactive, and real-time engines that can simultaneously access the same data sets
 - Multi-tenant data processing improves an enterprise's return on its Hadoop investments.
- Cluster utilization
 - YARN's dynamic allocation of cluster resources improves utilization over more static MapReduce rules used in early versions of Hadoop
- Scalability
 - Data center processing power continues to rapidly expand. YARN's ResourceManager focuses exclusively on scheduling and keeps pace as clusters expand to thousands of nodes managing petabytes of data.
- Compatibility
 - Existing MapReduce applications developed for Hadoop 1 can run YARN without any disruption to existing processes that already work

YARN major features summarized

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Topic: The architecture of YARN

MapReduce and Yarn

© Copyright IBM Corporation 2015

Topic: The architecture of YARN

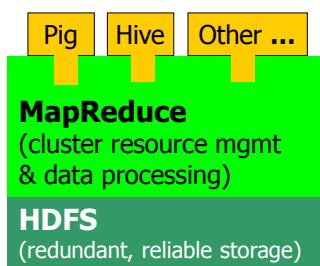
This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Hadoop v1 to Hadoop v2

Single Use System

Batch Apps Usually

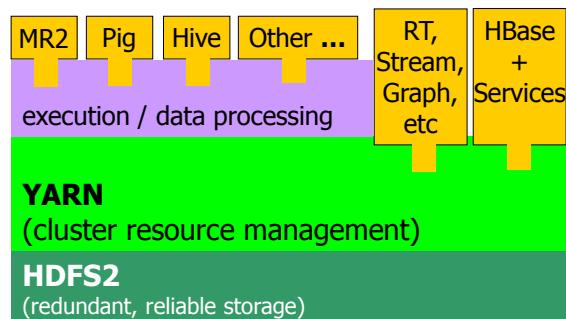
Hadoop 1.0



Multi Purpose Platform

Batch, Interactive, Online, Streaming

Hadoop 2.0



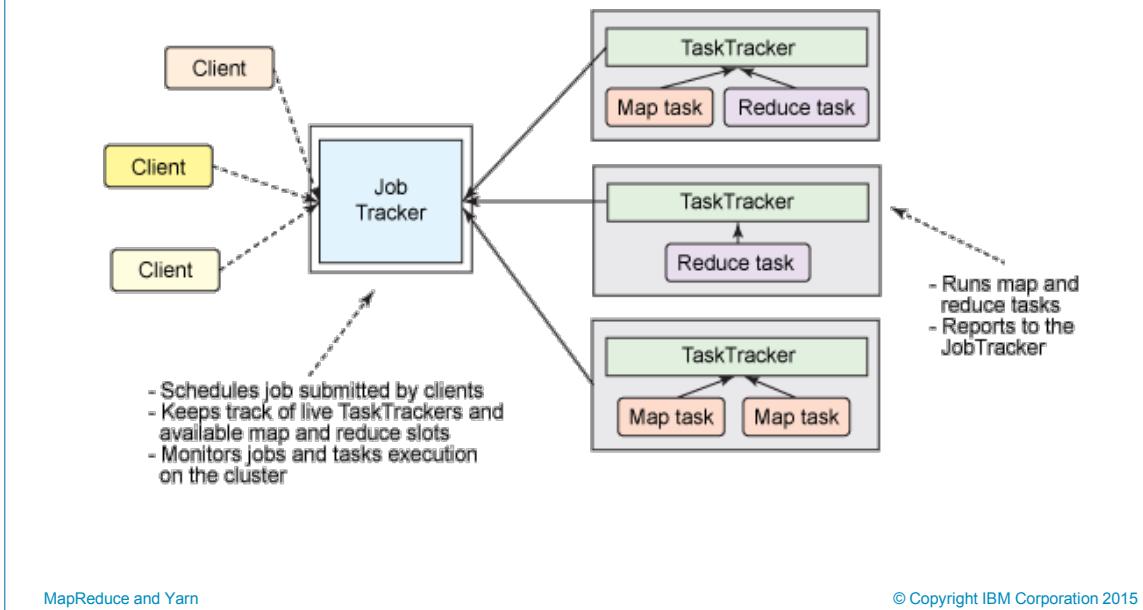
Hadoop v1 to Hadoop v2

The most notable change from Hadoop v1 to Hadoop v2 is the separation of cluster and resource management from the execution and data processing environment.

This allows for a new variety of application types to run, including MapReduce v2.

Architecture of MRv1

- Classic version of MapReduce (MRv1)



MapReduce and Yarn

© Copyright IBM Corporation 2015

Architecture of MRv1

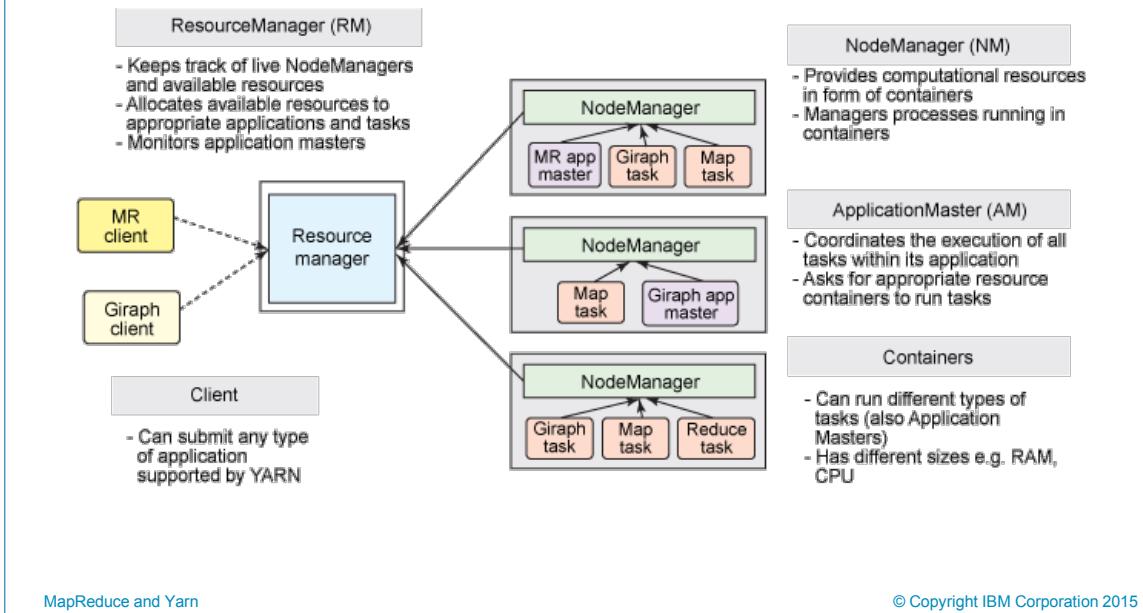
The effect most prominently seen with the overall job control.

In MapReduce v1 there is just one JobTracker that is responsible for allocation of resources, task assignment to data nodes (as TaskTrackers), and ongoing monitoring ("heartbeat") as each job is run (the TaskTrackers constantly report back to the JobTracker on the status of each running task).

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

YARN architecture

- High level architecture of YARN



YARN architecture

In the YARN architecture, a global ResourceManager runs as a master daemon, usually on a dedicated machine, that arbitrates the available cluster resources among various competing applications. The ResourceManager tracks how many live nodes and resources are available on the cluster and coordinates what applications submitted by users should get these resources and when.

The ResourceManager is the single process that has this information so it can make its allocation (or rather, scheduling) decisions in a shared, secure, and multi-tenant manner (for instance, according to an application priority, a queue capacity, ACLs, data locality, etc.).

When a user submits an application, an instance of a lightweight process called the ApplicationMaster is started to coordinate the execution of all tasks within the application. This includes monitoring tasks, restarting failed tasks, speculatively running slow tasks, and calculating total values of application counters. These responsibilities were previously assigned to the single JobTracker for all jobs. The ApplicationMaster and tasks that belong to its application run in resource containers controlled by the NodeManagers.

The NodeManager is a more generic and efficient version of the TaskTracker. Instead of having a fixed number of map and reduce slots, the NodeManager has a number of dynamically created resource containers. The size of a container depends upon the amount of resources it contains, such as memory, CPU, disk, and network IO.

Currently, only memory and CPU (YARN-3) are supported. cgroups might be used to control disk and network IO in the future. The number of containers on a node is a product of configuration parameters and the total amount of node resources (such as total CPUs and total memory) outside the resources dedicated to the slave daemons and the OS.

Interestingly, the ApplicationMaster can run any type of task inside a container. For example, the MapReduce ApplicationMaster requests a container to launch a map or a reduce task, while the Giraph ApplicationMaster requests a container to run a Giraph task. You can also implement a custom ApplicationMaster that runs specific tasks and, in this way, invent a shiny new distributed application framework that changes the big data world. I encourage you to read about Apache Twill, which aims to make it easy to write distributed applications sitting on top of YARN.

In YARN, MapReduce is simply degraded to a role of a distributed application (but still a very popular and useful one) and is now called MRv2. MRv2 is simply the re-implementation of the classical MapReduce engine, now called MRv1, that runs on top of YARN.

Terminology changes from MRv1 to YARN

YARN terminology	Instead of MRv1 terminology
ResourceManager	Cluster Manager
ApplicationMaster (but dedicated and short-lived)	JobTracker
NodeManager	TaskTracker
Distributed Application	One particular MapReduce job
Container	Slot

YARN in BigInsights

- Acronym for Yet Another Resource Negotiator
- New resource manager included in Hadoop 2.x and later
- De-couples Hadoop workload and resource management
- Introduces a general purpose application container
- Hadoop 2.2.0 includes the first GA version of YARN
- Most Hadoop vendors support YARN including IBM

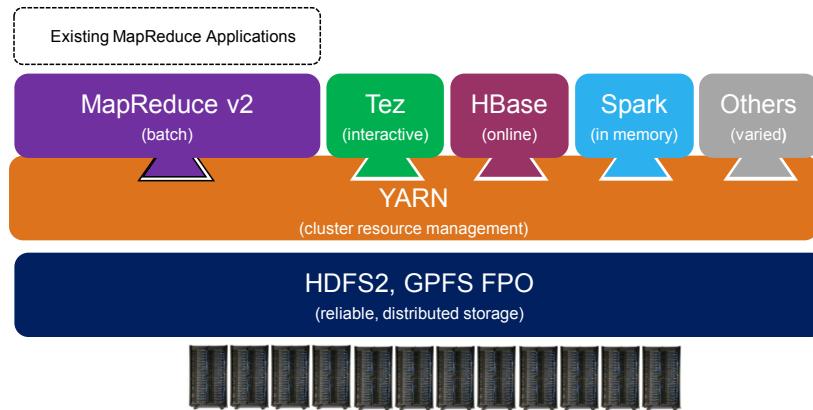
YARN in BigInsights

YARN is a key component in the Open Data Platform Initiative and hence in IBM BigInsights 4.

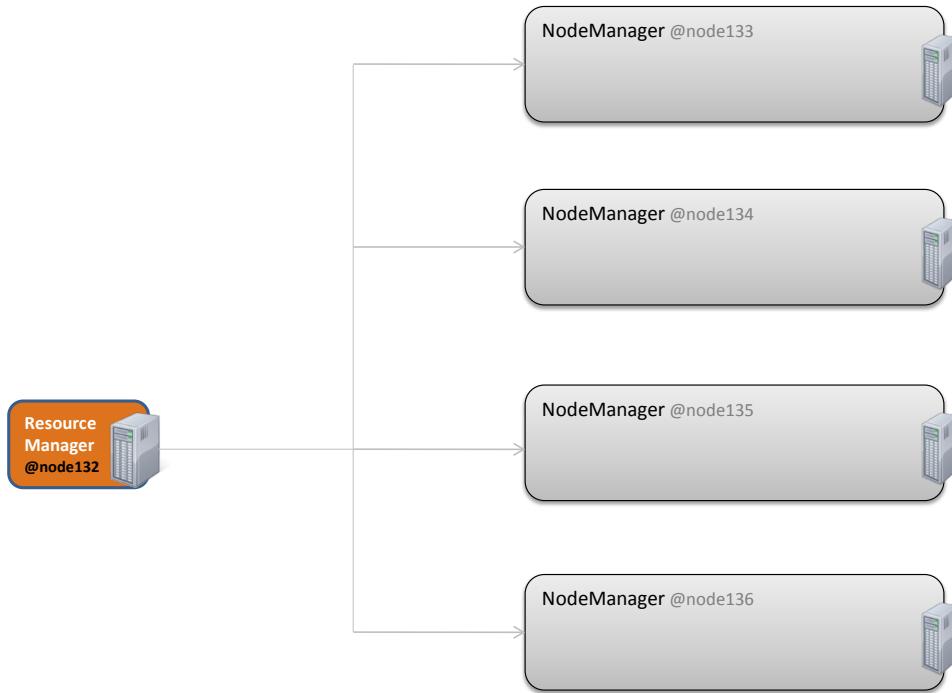
BigInsights 4 is based on Hadoop 2.6.0. Other (later) revisions are currently being tested for the ODP initiative.

YARN high level architecture

- In BigInsights, customers can take advantage of YARN and applications written to YARN APIs
- Your choice of HDFS or GPFS FPO for greater flexibility (GPFS requires the BigInsights Enterprise Management add-in)



Running an application in YARN (1 of 7)



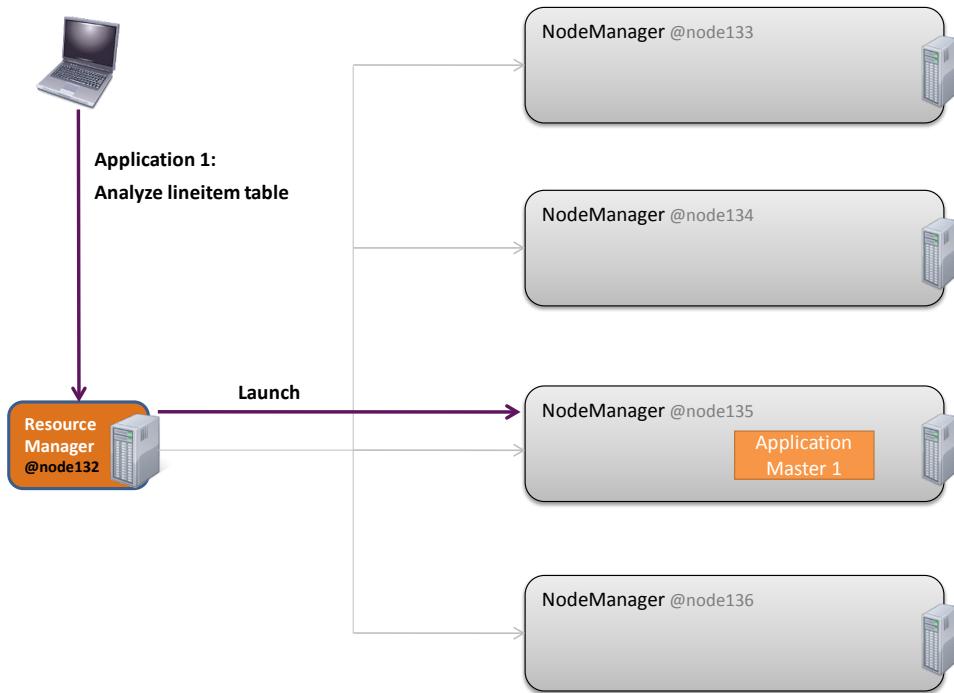
MapReduce and Yarn

© Copyright IBM Corporation 2015

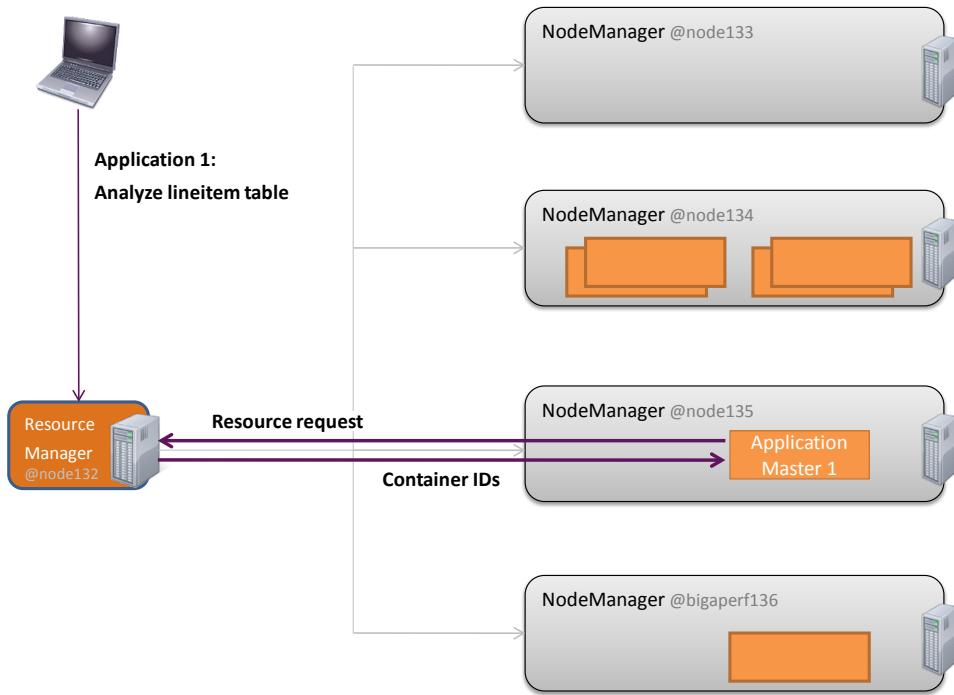
Running an application in YARN

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

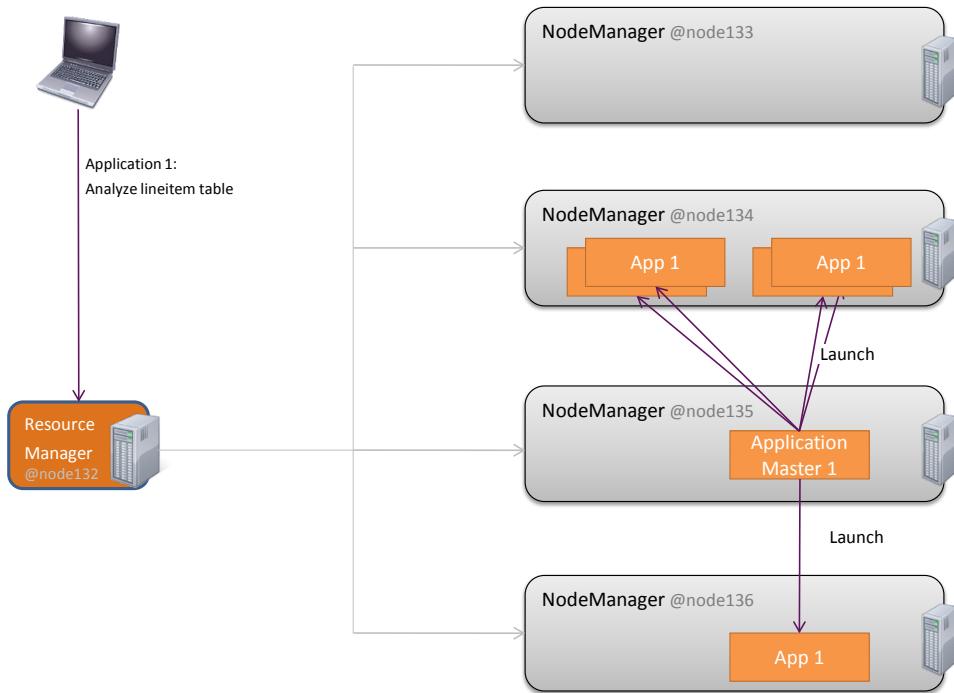
Running an application in YARN (2 of 7)



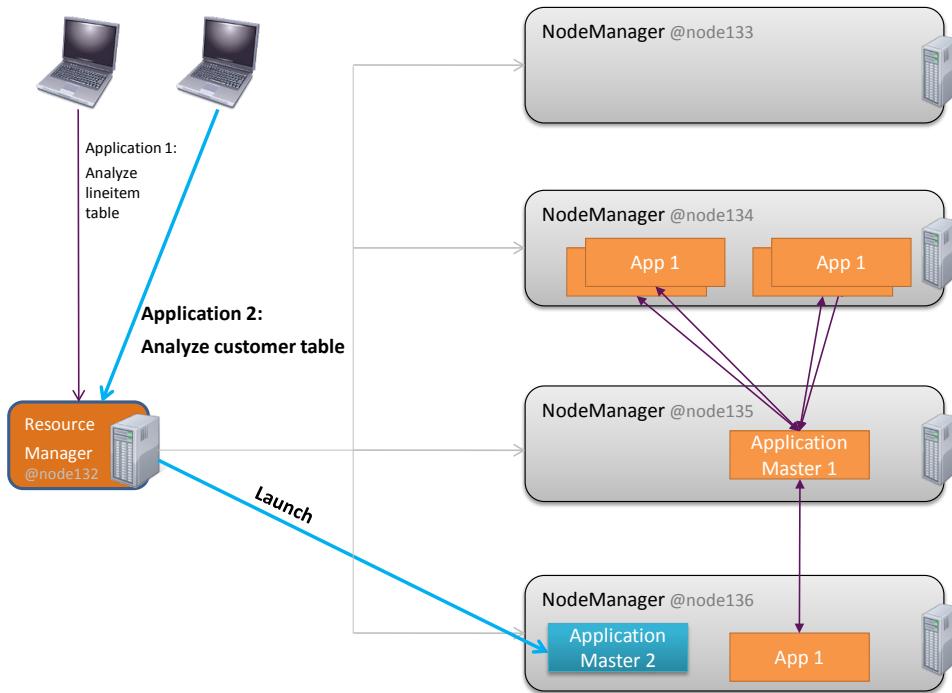
Running an application in YARN (3 of 7)



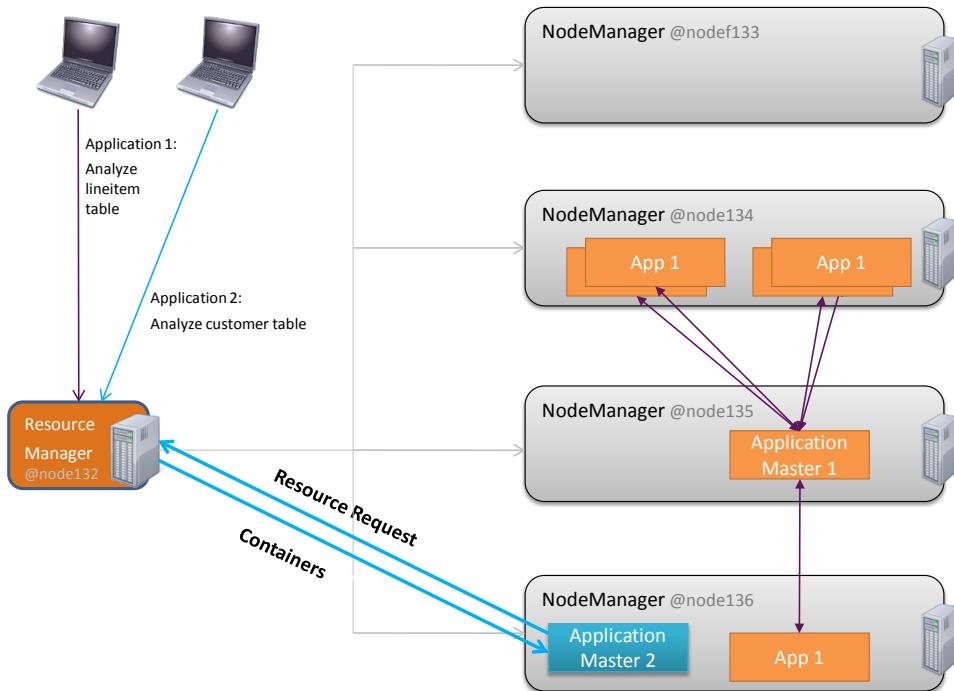
Running an application in YARN (4 of 7)



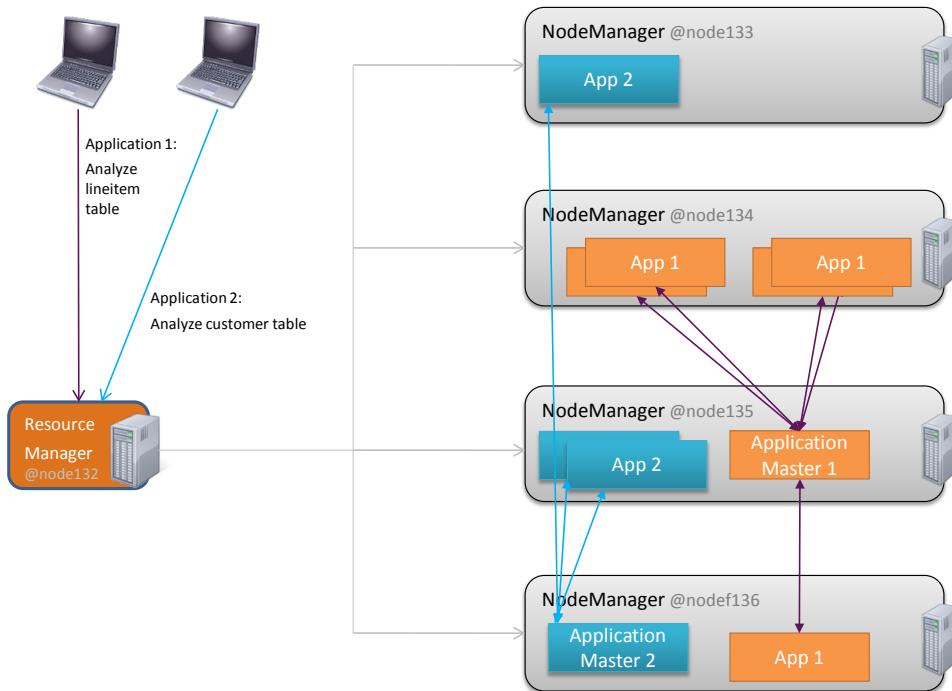
Running an application in YARN (5 of 7)



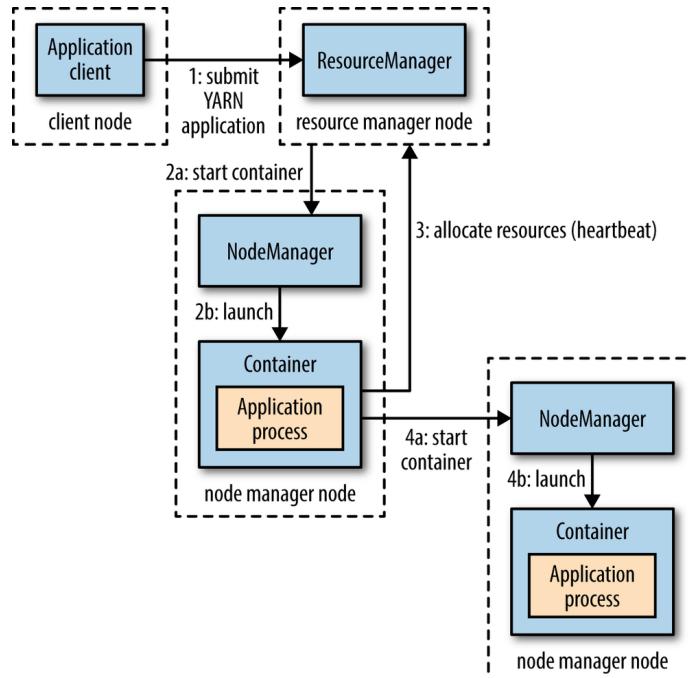
Running an application in YARN (6 of 7)



Running an application in YARN (7 of 7)



How YARN runs an application



MapReduce and Yarn

© Copyright IBM Corporation 2015

How YARN runs an application

To run an application on YARN, a client contacts the resource manager and asks it to run an application master process (step 1). The resource manager then finds a node manager that can launch the application master in a container (steps 2a and 2b). Precisely what the application master does once it is running depends on the application. It could simply run a computation in the container it is running in and return the result to the client. Or it could request more containers from the resource managers (step 3), and use them to run a distributed computation (steps 4a and 4b).

This is well described in: White, T. (2015) *Hadoop: The definitive guide* (4th ed.). Sebastopol, CA: O'Reilly Media, p. 80.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Container Java command line

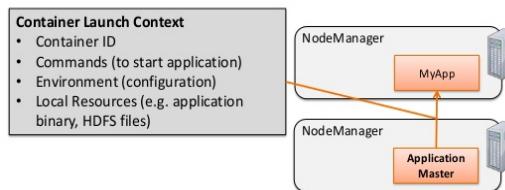
- Container JVM command (generally behind the scenes)

- Launched by "yarn" user with /bin/bash

```
yarn 1251527 1199943 0 14:38 ? 00:00:00 /bin/bash -c
/opt/ibm/biginsights/jdk/bin/java -Djava.net ...
```

- If you count "java" process ids (pids) running with the "yarn" user, you will see 2X

```
00:00:00 /bin/bash -c /opt/ibm/biginsights/jdk/bin/java
00:00:00 /bin/bash -c /opt/ibm/biginsights/jdk/bin/java
...
00:07:48 /opt/ibm/biginsights/jdk/bin/java -Djava.net.pr
00:08:11 /opt/ibm/biginsights/jdk/bin/java -Djava.net.pr
...
```



Container JAVA command line

This is for the more experienced user.

```
yarn 1251527 1199943 0 14:38 ? 00:00:00 /bin/bash -c
/opt/ibm/biginsights/jdk/bin/java -Djava.net.preferIPv4Stack=true -
Dhadoop.metrics.log.level=WARNING -Xmx2000m -Xms1000m -Xmn100m -Xtune:virtualized -
Xshareclasses:name=mrscc_%g,groupAccess,cacheDir=/var/ibm/biginsights/hadoop/tmp,nonFata
l -Xscmx20m -
Xdump:java:file=/var/ibm/biginsights/hadoop/tmp/javacore.%Y%m%d.%H%M%S.%pid.%seq.txt -
Xdump:heap:file=/var/ibm/biginsights/hadoop/tmp/heapdump.%Y%m%d.%H%M%S.%pid.%seq.phd -
Djava.io.tmpdir=/data6/yarn/local/nodemanager-
local/usercache/bigsql/appcache/application_1417731580977_0002/container_1417731580977_0
002_01_000095/tmp -Dlog4j.configuration=container-log4j.properties -
Dyarn.app.container.log.dir=/var/ibm/biginsights/hadoop/yarn/logs/application_1417731580
977_0002/container_1417731580977_0002_01_000095 -Dyarn.app.container.log.filesize=0 -
Dhadoop.root.logger=INFO,CLA org.apache.hadoop.mapred.YarnChild 9.30.75.55 51923
attempt_1417731580977_0002_m_000073_0 95
1>/var/ibm/biginsights/hadoop/yarn/logs/application_1417731580977_0002/container_1417731
580977_0002_01_000095/stdout
2>/var/ibm/biginsights/hadoop/yarn/logs/application_1417731580977_0002/container_1417731
580977_0002_01_000095/stderr
```

Provisioning, management, and monitoring

- The Apache Ambari project is aimed at making Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters. Ambari provides an intuitive, easy-to-use Hadoop management web UI backed by its RESTful APIs.
- Ambari enables System Administrators to:
 - Provision an Hadoop Cluster
 - Ambari provides a step-by-step wizard for installing Hadoop services across any number of hosts.
 - Ambari handles configuration of Hadoop services for the cluster.
 - Manage an Hadoop Cluster
 - Ambari provides central management for starting, stopping, and reconfiguring Hadoop services across the entire cluster.
 - Monitor an Hadoop Cluster
 - Ambari provides a dashboard for monitoring health and status of the Hadoop cluster.
 - Ambari leverages [Ganglia](#) for metrics collection.
 - Ambari leverages [Nagios](#) for system alerting and will send emails when your attention is needed (for example, a node goes down, remaining disk space is low, etc.).
- Ambari enables Application Developers and System Integrators to:
 - Easily integrate Hadoop provisioning, management, and monitoring capabilities to their own applications with the Ambari REST APIs

Provisioning, management, and monitoring

This is just a reminder of the role of Ambari in Hadoop 2.

With Hadoop 2 and YARN there is a great need for provisioning, management, and monitoring because of the greater complexity.

In the final unit you will look at Ambari Slider as a mechanism for dynamically changing requirements at run time for long running jobs.

Spark with Hadoop 2+

- Spark is an alternative in-memory framework to MapReduce
- Supports general workloads as well as streaming, interactive queries and machine learning providing performance gains
- Spark SQL provides APIs that allow SQL queries to be embedded in Scala, Python or Java programs in Spark
- MLLib: Spark optimized library support machine learning functions
- GraphX: API for graphs and parallel computation
- Spark streaming: Write applications to process streaming data in Java or Scala



Spark in Hadoop 2+

Apache Spark is a new, alternative, in-memory framework that is an alternative to MapReduce.

Spark is the subject of the next unit.

Unit summary

- Describe the MapReduce model v1
- List the limitations of Hadoop 1 and MapReduce 1
- Review the Java code required to handle the Mapper class, the Reducer class, and the program driver needed to access MapReduce
- Describe the YARN model
- Compare Hadoop 2/YARN with Hadoop 1

Exercise 1

Running MapReduce and YARN jobs

MapReduce and Yarn

© Copyright IBM Corporation 2015

Exercise 1: Running MapReduce and YARN jobs

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1: Running MapReduce and YARN jobs

Purpose:

You will run Java programs using Hadoop v2, YARN, and related technologies. You will not need to do any Java programming. The Hadoop community provides a number of standard example programs (akin to "Hello, World" for people learning to write C language programs).

In the real world, the sample/example programs provide opportunities to learn the relevant technology, and on a newly installed Hadoop cluster, to exercise the system and the operations environment.

You will start with an existing sample program, selected from among those available, and progress towards compiling a program a Java program from scratch and learning in what type of environment it runs.

The Java program that you will be compiling and running, `WordCount2.java`, resides in the Linux directory `/home/labfiles`.

VM Hostname: `http://ibmclass.localdomain`

User/Password: `biadmin / biadmin`
`root/dalvm3`

Task 1. Run a simple MapReduce job from supplied example programs available with a Hadoop default installation.

The best references for writing MapReduce programs for MRv1, MRv2, and YARN are on the Hadoop website. The URLs relevant for Hadoop r2.7.0 are:

<http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>

<http://hadoop.apache.org/docs/r2.7.0/hadoop-yarn/hadoop-yarn-site/WritingYarnApplications.html>

Use the documentation that corresponds to your current release of Hadoop and HDFS (substitute the appropriate value for r2.7.0, if different).

1. Connect to and login to your lab environment with user **biadmin** and password **biadmin** credentials.
2. In a new terminal window, type `cd` to change to your home directory.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

3. To locate all of the Java jar files in your system with **example** in the file name, type `find / -name "*example*" 2> /dev/null | grep jar`.

There will be files and directories in your Linux file system that you, as biadmin user, will not have access to. The find command will give you errors. Those errors are sent to Linux errout and are eliminated from your console output by using: `2> /dev/null`

```
[hdfs@ibmclass ~]$ find / -name "*example*" 2> /dev/null | grep jar
/usr/share/java/rhino-examples-1.7.jar
/usr/share/java/rhino-examples.jar
/usr/iop/4.0.0.0/oozie/doc/oozie-4.1.0_IBM_5/examples/apps/java-main/lib/oozie-
examples-4.1.0_IBM_5.jar
/usr/iop/4.0.0.0/oozie/doc/oozie-4.1.0_IBM_5/examples/apps/datelist-java-
main/lib/oozie-examples-4.1.0_IBM_5.jar
/usr/iop/4.0.0.0/oozie/doc/oozie-4.1.0_IBM_5/examples/apps/hadoop-el/lib/oozie-
examples-4.1.0_IBM_5.jar
/usr/iop/4.0.0.0/oozie/doc/oozie-4.1.0_IBM_5/examples/apps/custom-main/lib/oozie-
examples-4.1.0_IBM_5.jar
/usr/iop/4.0.0.0/oozie/doc/oozie-4.1.0_IBM_5/examples/apps/demo/lib/oozie-examples-
4.1.0_IBM_5.jar
/usr/iop/4.0.0.0/oozie/doc/oozie-4.1.0_IBM_5/examples/apps/map-reduce/lib/oozie-
examples-4.1.0_IBM_5.jar
/usr/iop/4.0.0.0/oozie/doc/oozie-4.1.0_IBM_5/examples/apps/aggregator/lib/oozie-
examples-4.1.0_IBM_5.jar
/usr/iop/4.0.0.0/oozie/share/lib/hbase/hbase-examples-0.98.8_IBM_4-hadoop2.jar
/usr/iop/4.0.0.0/knox/samples/hadoop-examples.jar
/usr/iop/4.0.0.0/hadoop-mapreduce/hadoop-mapreduce-examples.jar
/usr/iop/4.0.0.0/hadoop-mapreduce/hadoop-mapreduce-examples-2.6.0-IBM-7.jar
/usr/iop/4.0.0.0/spark/lib/spark-examples.jar
/usr/iop/4.0.0.0/spark/lib/spark-examples-1.2.1_IBM_4-hadoop2.6.0-IBM-7.jar
/usr/iop/4.0.0.0/hbase/lib/hbase-examples.jar
/usr/iop/4.0.0.0/hbase/lib/hbase-examples-0.98.8_IBM_4-hadoop2.jar
[hdfs@ibmclass ~]$
```

You can list the contents of a jar file using `jar tf` (`t` = table of contents, `f` = file name follows); you will apply that to one of the jar files that you found with your find-search.

4. To list the contents of the **hadoop-mapreduce-examples.jar** file, type `jar tf /usr/iop/4.0.0.0/hadoop-mapreduce/hadoop-mapreduce-examples.jar`.

The results will show something similar to:

```
[hdfs@ibmclass ~]$ jar -tf /usr/iop/4.0.0.0/hadoop-mapreduce/hadoop-mapreduce-
examples.jar
META-INF/
META-INF/MANIFEST.MF
org/
org/apache/
org/apache/hadoop/
org/apache/hadoop/examples/
org/apache/hadoop/examples/dancing/
org/apache/hadoop/examples/pi/
org/apache/hadoop/examples/pi/math/
org/apache/hadoop/examples/terasort/
org/apache/hadoop/examples/RandomWriter$RandomInputFormat$RandomRecordReader.class
org/apache/hadoop/examples/DBCountPageView$PageviewReducer.class
org/apache/hadoop/examples/DBCountPageView$PageviewRecord.class
org/apache/hadoop/examples/SecondarySort$FirstGroupingComparator.class
org/apache/hadoop/examples/MultiFileWordCount.class
org/apache/hadoop/examples/QuasiMonteCarlo.class
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

```

org/apache/hadoop/examples/BaileyBorweinPlouffe$BbpInputFormat.class
org/apache/hadoop/examples/SecondarySort$IntPair.class
org/apache/hadoop/examples/SecondarySort$Reduce.class
org/apache/hadoop/examples/BaileyBorweinPlouffe$BbpInputFormat$1.class
org/apache/hadoop/examples/BaileyBorweinPlouffe$Fraction.class
org/apache/hadoop/examples/DBCountPageView.class
org/apache/hadoop/examples/WordMean$WordMeanMapper.class
org/apache/hadoop/examples/SecondarySort.class
org/apache/hadoop/examples/RandomTextWriter.class
org/apache/hadoop/examples/QuasiMonteCarlo$HaltonSequence.class
org/apache/hadoop/examples/RandomWriter$RandomInputFormat.class
org/apache/hadoop/examples/Sort.class
org/apache/hadoop/examples/WordStandardDeviation.class
org/apache/hadoop/examples/BaileyBorweinPlouffe$1.class
org/apache/hadoop/examples/dancing/Sudoku$ColumnConstraint.class
org/apache/hadoop/examples/dancing/OneSidedPentomino.class
org/apache/hadoop/examples/dancing/Sudoku$SolutionPrinter.class
org/apache/hadoop/examples/dancing/Pentomino$Point.class
org/apache/hadoop/examples/dancing/Sudoku$RowConstraint.class
org/apache/hadoop/examples/dancing/DistributedPentomino$PentMap.class
org/apache/hadoop/examples/dancing/DistributedPentomino$PentMap$SolutionCatcher.class
org/apache/hadoop/examples/dancing/DancingLinks$Node.class
org/apache/hadoop/examples/dancing/Sudoku$CellConstraint.class
org/apache/hadoop/examples/dancing/DancingLinks$ColumnHeader.class
org/apache/hadoop/examples/dancing/Pentomino$ColumnName.class
org/apache/hadoop/examples/dancing/DancingLinks$SolutionAcceptor.class
org/apache/hadoop/examples/dancing/Pentomino.class
org/apache/hadoop/examples/dancing/DancingLinks.class
org/apache/hadoop/examples/dancing/Sudoku.class
org/apache/hadoop/examples/dancing/Pentomino$SolutionPrinter.class
org/apache/hadoop/examples/dancing/DistributedPentomino.class
org/apache/hadoop/examples/dancing/Sudoku$SquareConstraint.class
org/apache/hadoop/examples/dancing/Pentomino$SolutionCategory.class
org/apache/hadoop/examples/dancing/Sudoku$ColumnName.class
org/apache/hadoop/examples/dancing/Pentomino$Piece.class
org/apache/hadoop/examples/AggregateWordHistogram.class
org/apache/hadoop/examples/DBCountPageView$PageviewMapper.class
org/apache/hadoop/examples/WordMedian$WordMedianReducer.class
org/apache/hadoop/examples/RandomTextWriter$RandomTextMapper.class
org/apache/hadoop/examples/BaileyBorweinPlouffe.class
org/apache/hadoop/examples/WordMean$WordMeanReducer.class
org/apache/hadoop/examples/RandomWriter$Counters.class
org/apache/hadoop/examples/BaileyBorweinPlouffe$BbpSplit.class
org/apache/hadoop/examples/BaileyBorweinPlouffe$BbpReducer$1.class
org/apache/hadoop/examples/WordCount.class
org/apache/hadoop/examples/Join.class
org/apache/hadoop/examples/SecondarySort$IntPair$Comparator.class
org/apache/hadoop/examples/MultiFileWordCount$MapClass.class
org/apache/hadoop/examples/AggregateWordCount$WordCountPlugInClass.class
org/apache/hadoop/examples/WordStandardDeviation$WordStandardDeviationReducer.class
org/apache/hadoop/examples/RandomTextWriter$Counters.class
org/apache/hadoop/examples/MultiFileWordCount$CombineFileLineRecordReader.class
org/apache/hadoop/examples/WordCount$TokenizerMapper.class
org/apache/hadoop/examples/BaileyBorweinPlouffe$BbpMapper.class
org/apache/hadoop/examples/QuasiMonteCarlo$QmcReducer.class
org/apache/hadoop/examples/MultiFileWordCount$WordOffset.class
org/apache/hadoop/examples/pi/DistSum$Machine$AbstractInputFormat.class
org/apache/hadoop/examples/pi/DistSum$MapSide$PartitionInputFormat.class
org/apache/hadoop/examples/pi/DistSum$ReduceSide$SummationInputFormat.class
org/apache/hadoop/examples/pi/Parser.class
org/apache/hadoop/examples/pi/DistSum.class
org/apache/hadoop/examples/pi/DistSum$Machine.class
org/apache/hadoop/examples/pi/SummationWritable.class
org/apache/hadoop/examples/pi/Util$Timer.class
org/apache/hadoop/examples/pi/DistSum$Parameters.class
org/apache/hadoop/examples/pi/DistSum$ReduceSide$PartitionMapper.class

```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

```

org/apache/hadoop/examples/pi/DistBbp.class
org/apache/hadoop/examples/pi/DistSum$1.class
org/apache/hadoop/examples/pi/DistSum$ReduceSide$SummingReducer.class
org/apache/hadoop/examples/pi/DistSum$Machine$AbstractInputFormat$1.class
org/apache/hadoop/examples/pi/DistSum$MapSide$SummingMapper.class
org/apache/hadoop/examples/pi/Combinable.class
org/apache/hadoop/examples/pi/Container.class
org/apache/hadoop/examples/pi/DistSum$Machine$SummationSplit.class
org/apache/hadoop/examples/pi/DistSum$MapSide.class
org/apache/hadoop/examples/pi/DistSum$MixMachine.class
org/apache/hadoop/examples/pi/Util.class
org/apache/hadoop/examples/pi/DistSum$Computation.class
org/apache/hadoop/examples/pi/DistSum$ReduceSide$IndexPartitioner.class
org/apache/hadoop/examples/pi/math/Bellard$Sum$Tail.class
org/apache/hadoop/examples/pi/math/Bellard$Parameter.class
org/apache/hadoop/examples/pi/math/ArithmetricProgression.class
org/apache/hadoop/examples/pi/math/Modular.class
org/apache/hadoop/examples/pi/math/LongLong.class
org/apache/hadoop/examples/pi/math/Summation.class
org/apache/hadoop/examples/pi/math/Montgomery.class
org/apache/hadoop/examples/pi/math/Bellard$Sum$1.class
org/apache/hadoop/examples/pi/math/Bellard$Sum.class
org/apache/hadoop/examples/pi/math/Montgomery$Product.class
org/apache/hadoop/examples/pi/math/Bellard$1.class
org/apache/hadoop/examples/pi/math/Bellard.class
org/apache/hadoop/examples/pi/DistSum$ReduceSide.class
org/apache/hadoop/examples/pi/SummationWritable$ArithmetricProgressionWritable.class
org/apache/hadoop/examples/pi/TaskResult.class
org/apache/hadoop/examples/RandomWriter$RandomMapper.class
org/apache/hadoop/examples/terasort/TeraInputFormat$TeraRecordReader.class
org/apache/hadoop/examples/terasort/TeraChecksum.class
org/apache/hadoop/examples/terasort/TeraOutputFormat$TeraRecordWriter.class
org/apache/hadoop/examples/terasort/TeraValidate$ValidateReducer.class
org/apache/hadoop/examples/terasort/GenSort.class
org/apache/hadoop/examples/terasort/Random16$RandomConstant.class
org/apache/hadoop/examples/terasort/TeraGen$SortGenMapper.class
org/apache/hadoop/examples/terasort/TeraInputFormat$1.class
org/apache/hadoop/examples/terasort/TeraInputFormat$SamplerThreadGroup.class
org/apache/hadoop/examples/terasort/TeraGen$RangeInputFormat$RangeRecordReader.class
org/apache/hadoop/examples/terasort/TeraChecksum$ChecksumReducer.class
org/apache/hadoop/examples/terasort/TeraSort.class
org/apache/hadoop/examples/terasort/Random16.class
org/apache/hadoop/examples/terasort/TeraSort$TotalOrderPartitioner$InnerTrieNode.class
org/apache/hadoop/examples/terasort/TeraSort$SimplePartitioner.class
org/apache/hadoop/examples/terasort/TeraGen.class
org/apache/hadoop/examples/terasort/TeraInputFormat$TextSampler.class
org/apache/hadoop/examples/terasort/TeraGen$Counters.class
org/apache/hadoop/examples/terasort/TeraSort$TotalOrderPartitioner.class
org/apache/hadoop/examples/terasort/TeraOutputFormat.class
org/apache/hadoop/examples/terasort/TeraScheduler$Split.class
org/apache/hadoop/examples/terasort/TeraSort$TotalOrderPartitioner$TrieNode.class
org/apache/hadoop/examples/terasort/TeraValidate$ValidateMapper.class
org/apache/hadoop/examples/terasort/TeraSort$TotalOrderPartitioner$LeafTrieNode.class
org/apache/hadoop/examples/terasort/TeraGen$RangeInputFormat.class
org/apache/hadoop/examples/terasort/TeraScheduler.class
org/apache/hadoop/examples/terasort/TeraChecksum$ChecksumMapper.class
org/apache/hadoop/examples/terasort/TeraScheduler$Host.class
org/apache/hadoop/examples/terasort/TeraValidate.class
org/apache/hadoop/examples/terasort/Unsigned16.class
org/apache/hadoop/examples/terasort/TeraInputFormat.class
org/apache/hadoop/examples/terasort/TeraGen$RangeInputFormat$RangeInputSplit.class
org/apache/hadoop/examples/AggregateWordHistogram$AggregateWordHistogramPlugin.class
org/apache/hadoop/examples/AggregateWordCount.class
org/apache/hadoop/examples/Grep.class
org/apache/hadoop/examples/WordCount$IntSumReducer.class
org/apache/hadoop/examples/WordMean.class

```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

```

org/apache/hadoop/examples/WordStandardDeviation$WordStandardDeviationMapper.class
org/apache/hadoop/examples/SecondarySort$MapClass.class
org/apache/hadoop/examples/BaileyBorweinPlouffe$BbpReducer.class
org/apache/hadoop/examples/RandomWriter.class
org/apache/hadoop/examples/SecondarySort$FirstPartitioner.class
org/apache/hadoop/examples/ExampleDriver.class
org/apache/hadoop/examples/WordMedian$WordMedianMapper.class
org/apache/hadoop/examples/MultiFileWordCount$MyInputFormat.class
org/apache/hadoop/examples/QuasiMonteCarlo$QmcMapper.class
org/apache/hadoop/examples/DBCountPageView$AccessRecord.class
org/apache/hadoop/examples/WordMedian.class
META-INF/maven/
META-INF/maven/org.apache.hadoop/
META-INF/maven/org.apache.hadoop/hadoop-mapreduce-examples/
META-INF/maven/org.apache.hadoop/hadoop-mapreduce-examples/pom.xml
META-INF/maven/org.apache.hadoop/hadoop-mapreduce-examples/pom.properties
[hdfs@ibmclass ~]$

```

This is a long list. You are going to use the WordCount example from this file, but what is the file called? Note also that you can get dates and times for the various classes and programs in this jar file by using `jar tvf` (`v` = verbose).

We can find what executable Java programs are included in the manifest by attempting to run this jar with `hadoop jar`, but not providing any additional and perhaps necessary parameters.

5. Type `hadoop jar /usr/iop/4.0.0.0/hadoop-mapreduce/hadoop-mapreduce-examples.jar`.

```

[hdfs@ibmclass ~]$ hadoop jar /usr/iop/4.0.0.0/hadoop-mapreduce/hadoop-mapreduce-examples.jar
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the words in the input files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the words in the input files.
  bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
  dbcount: An example job that count the pageview counts from a database.
  distbbp: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets
  multifilewc: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino problems.
  pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
  randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.
  randomwriter: A map/reduce program that writes 10GB of random data per node.
  secondarysort: An example defining a secondary sort to the reduce.
  sort: A map/reduce program that sorts the data written by the random writer.
  sudoku: A sudoku solver.
  teragen: Generate data for the terasort
  terasort: Run the terasort
  teravalidate: Checking results of terasort
  wordcount: A map/reduce program that counts the words in the input files.
  wordmean: A map/reduce program that counts the average length of the words in the input files.
  wordmedian: A map/reduce program that counts the median length of the words in the input files.
  wordstandarddeviation: A map/reduce program that counts the standard deviation of the length of the words in the input files.
[hdfs@ibmclass ~]$

```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

The program that you will use is wordcount which is a map/reduce program that counts the words in the input files.

6. To run the program with appropriate parameters, type `hadoop jar /usr/iop/4.0.0.0/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount Gutenberg/Frankenstein.txt wcout`.

```
[biadmin@ibmclass ~]$ hadoop jar /usr/iop/4.0.0.0/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount Gutenberg/Frankenstein.txt wcout
```

```
15/06/04 14:48:45 INFO impl.TimelineClientImpl: Timeline service address: http://ibmclass.localdomain:8188/ws/v1/timeline/
15/06/04 14:48:45 INFO client.RMProxy: Connecting to ResourceManager at ibmclass.localdomain/192.168.244.140:8050
15/06/04 14:48:46 INFO input.FileInputFormat: Total input paths to process : 1
15/06/04 14:48:46 INFO mapreduce.JobSubmitter: number of splits:1
15/06/04 14:48:47 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1433282779965_0002
15/06/04 14:48:48 INFO impl.YarnClientImpl: Submitted application application_1433282779965_0002
15/06/04 14:48:48 INFO mapreduce.Job: The url to track the job: http://ibmclass.localdomain:8088/proxy/application_1433282779965_0002/
15/06/04 14:48:48 INFO mapreduce.Job: Running job: job_1433282779965_0002
15/06/04 14:49:21 INFO mapreduce.Job: Job job_1433282779965_0002 running in uber mode : false
15/06/04 14:49:21 INFO mapreduce.Job: map 0% reduce 0%
15/06/04 14:49:57 INFO mapreduce.Job: map 100% reduce 0%
15/06/04 14:50:11 INFO mapreduce.Job: map 100% reduce 100%
15/06/04 14:50:12 INFO mapreduce.Job: Job job_1433282779965_0002 completed successfully
15/06/04 14:50:12 INFO mapreduce.Job: Counters: 49
    File System Counters
        FILE: Number of bytes read=167616
        FILE: Number of bytes written=564573
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=421641
        HDFS: Number of bytes written=122090
        HDFS: Number of read operations=6
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=33447
        Total time spent by all reduces in occupied slots (ms)=11038
        Total time spent by all map tasks (ms)=33447
        Total time spent by all reduce tasks (ms)=11038
        Total vcore-seconds taken by all map tasks=33447
        Total vcore-seconds taken by all reduce tasks=11038
        Total megabyte-seconds taken by all map tasks=34249728
        Total megabyte-seconds taken by all reduce tasks=11302912
    Map-Reduce Framework
        Map input records=7244
        Map output records=74952
        Map output bytes=717818
        Map output materialized bytes=167616
        Input split bytes=137
        Combine input records=74952
        Combine output records=11603
        Reduce input groups=11603
        Reduce shuffle bytes=167616
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

```

Reduce input records=11603
Reduce output records=11603
Spilled Records=23206
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=291
CPU time spent (ms)=5950
Physical memory (bytes) snapshot=799457280
Virtual memory (bytes) snapshot=3353075712
Total committed heap usage (bytes)=859832320
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=421504
File Output Format Counters
  Bytes Written=122090
[biadmin@ibmclass ~]$
```

There was just one Reduce task:

```

Launched map tasks=1
Launched reduce tasks=1
```

and hence there would be only one output file.

7. To find what file(s) were produced, type **hadoop fs -ls -R**.

```

[biadmin@ibmclass ~]$ hadoop fs -ls -R
drwx---  - biadmin          0 2015-06-04 14:50 .staging
drwxr-xr-x  - biadmin biadmin          0 2015-06-04 11:01 Gutenberg
-rw-r-r-  3 biadmin biadmin    421504 2015-06-04 11:01 Gutenberg/Frankenstein.txt
-rw-r-r-  3 biadmin biadmin    697802 2015-06-04 11:01 Gutenberg/Pride_and_Prejudice.txt
-rw-r-r-  3 biadmin biadmin   757223 2015-06-04 11:01 Gutenberg/Tale_of_Two_Cities.txt
-rw-r-r-  3 biadmin biadmin  281398 2015-06-04 11:01 Gutenberg/The_Prince.txt
drwxr-xr-x  - biadmin biadmin          0 2015-06-04 14:50 wcout
-rw-r-r-  3 biadmin biadmin          0 2015-06-04 14:50 wcout/_SUCCESS
-rw-r-r-  3 biadmin biadmin   122090 2015-06-04 14:50 wcout/part-r-00000
[biadmin@ibmclass ~]$
```

The directory **wcout** (**/user/biadmin/wcout**) in HDFS was created and there are two files in it: **_SUCCESS** and **part-r-00000**.

8. To review the **part-r-0000** file, type `hadoop fs -cat wcout/part-r-00000 | more`.

Scroll through the part-r-0000 file and look at least a dozen or more pages of output into more (press Enter to see more than one page of output).

Here is a sample from several pages down:

```
Concerning      1
Constantinople  1
Constantinople, 1
Continue       1
Continuing     1
Copet.         1
Cornelius      6
Could          6
Coupar,        1
Cousin,        1
Covered        1
Creator.       1
Creator;       1
Cumberland     3
```

While the job was running, you could have watched its progress by using the suggest URL. For the job listing above, this was:

```
15/06/04 14:48:48 INFO mapreduce.Job: The url to track the job:
http://ibmclass.localdomain:8088/proxy/application\_1433282779965\_0002/
```

9. Type **q** to exit from more, and then to remove the directory where your output file was stored (wcout), type `hadoop fs -rm -R wcout`.

It is necessary to remove this directory and all files in it to use the command again without changes. Alternatively you could run the command again, but with a different output directory such as wcout2.

10. You will now run the wordcount program again and, immediately after it starts, you should copy the URL for that run by selecting it, right-clicking, and clicking **Copy**. Paste the URL into a Firefox browser within your lab environment.

Depending on how fast you do this, you will see something like the following, if your MapReduce job is still running:

Or the following, if your MapReduce job has already completed:

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

11. The same job can be run (and monitored) as a YARN job. To do this, type `yarn jar /usr/iop/4.0.0.0/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount Gutenberg/Frankenstein.txt wcout2`.
12. You can also run the job against all four files in the Gutenberg directory. To do this, type `hadoop jar /usr/iop/4.0.0.0/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount Gutenberg/*.txt wcout2`.

Question 1: How many Mappers were used for your run?

Question 2: How many Reducers were used for your run?

Task Type	Total		Complete
	Map	Reduce	
Attempt Type	Failed	Killed	Successful
Maps	0	0	4
Reduces	0	0	1

Results:

You ran Java programs using Hadoop v2, YARN, and related technologies. You started with an existing sample program, selected from among those available, and progressed towards compiling a program a Java program from scratch and learned in what type of environment it runs.

Exercise 2

Creating and coding a simple MapReduce job

Exercise 2: Creating and coding a simple MapReduce job

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 2: Creating and coding a simple MapReduce job

Purpose:

You will compile and run a more complete version of WordCount that has been written specifically for MapReduce2.

Task 1. Compile and run a more complete version of WordCount that has been written specifically for MapReduce2.

The program that you will use - `WordCount2.java` - has been taken from http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v2.0

There is a small error in the Java code ("| " with a space rather than "||" without a space. This has been corrected in the version in /home/labfiles.

This version 2 of WordCount.java is more sophisticated than the one that you have already run, since it allows for splitting on text in addition to and other than whitespace, and also allows you to specify anything that you want to skip when counting words (such as "to", "the", etc.).

There are some limitations; if you are a competent Java programmer, you may want to experiment later with other features. For instance, are all words lowercased when they are tokenized?

Since you are now more familiar with the process of running MR/YARN jobs, the directions provided here will concentrate on the compilation only.

1. To copy the files that you will need to your **biadmin** home directory, type the following commands:

```
cd /home/biadmin/labfiles
cp patternsToSkip run-wc WordCount2.java .
```

2. To move the file `patternsToSkip` to HDFS, type `hadoop fs -put patternsToSkip ..`
3. Type `hadoop classpath`, which provides you with the CLASSPATH environmental variables that you need for compilation.

You will not likely want to type that information manually.

4. Compile WordCount.java with the Hadoop2 API and this CLASSPATH:

```
javac -cp `hadoop classpath` WordCount2.java
```

Note here, the use of the back-quote character to substitute the value of the CLASSPATH set into the compilation as the classpath (-cp) needed for the compilation.

5. Create a Jar file that can be run in the Hadoop2/YARN environment:

```
jar cf WC2.jar *.class
```

```
jar tf WC2.jar
```

6. To run your Jar file successfully, you will need to remove all the class files from your biadmin Linux directory:

```
rm *.class
```

7. You are now ready to run your compiled program with appropriate parameters (see the program logic and the use of these additional parameters on the website listed above). Type the following command on one line:

```
hadoop jar WC2.jar WordCount2 -D wordcount.case.sensitive=false  
./Gutenberg/*.txt ./wc2out -skip ./patternsToSkip
```

This can also be run with "yarn" and the MR job can be monitored as previously.

Question 1: How many Mappers were used for your run?

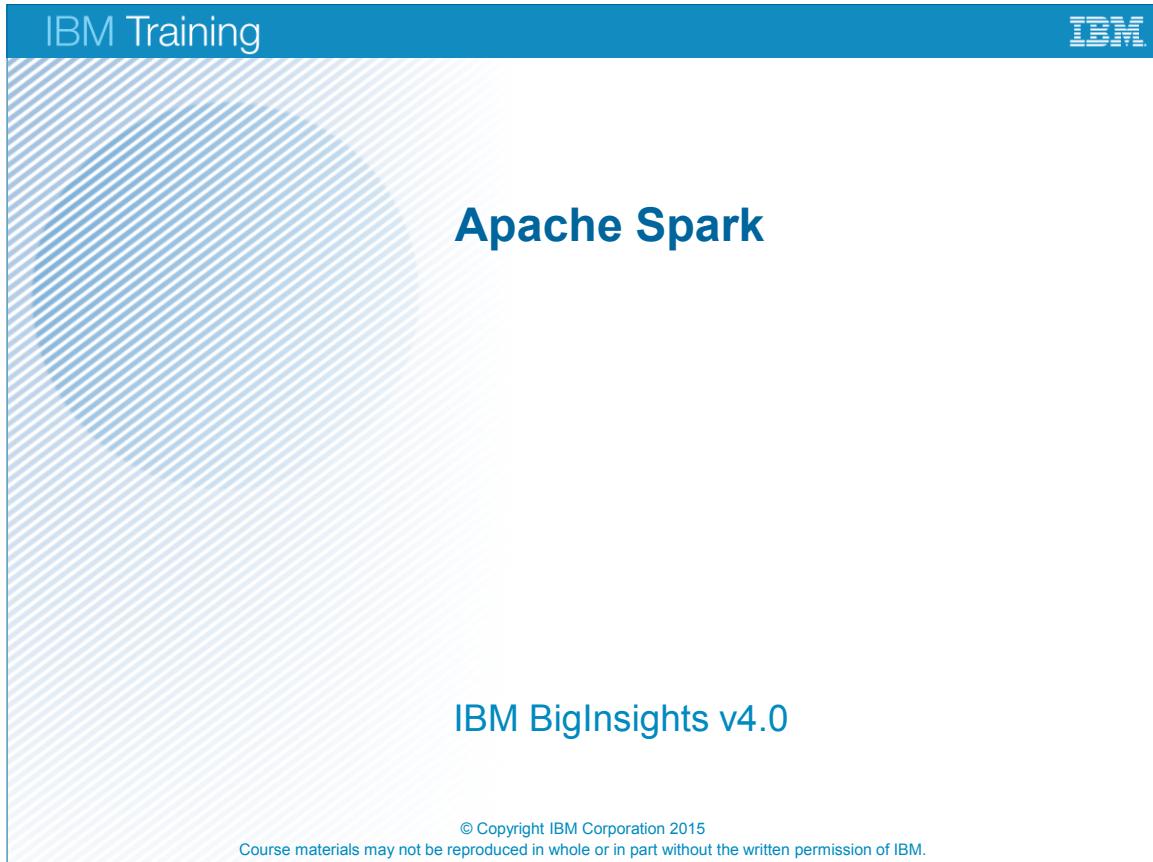
Question 2: How many Reducers were used for your run?

8. Look at your output (in wc2out) and see if it differs from what you generated previously.

Results:

You compiled and ran a more complete version of WordCount that had been written specifically for MapReduce2.

Unit 10 Apache Spark



The slide features a blue header bar with 'IBM Training' on the left and the IBM logo on the right. The main content area has a light gray diagonal striped background. In the center, the text 'Apache Spark' is displayed in a large, bold, blue font. Below it, 'IBM BigInsights v4.0' is shown in a smaller, blue font. At the bottom of the slide, there is a copyright notice: '© Copyright IBM Corporation 2015' and 'Course materials may not be reproduced in whole or in part without the written permission of IBM.'

IBM Training

IBM

Apache Spark

IBM BigInsights v4.0

© Copyright IBM Corporation 2015
Course materials may not be reproduced in whole or in part without the written permission of IBM.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit objectives

- Understand the nature and purpose of Apache Spark in the Hadoop ecosystem
- List and describe the architecture and components of the Spark unified stack
- Describe the role of a Resilient Distributed Dataset (RDD)
- Understand the principles of Spark programming
- List and describe the Spark libraries
- Launch and use Spark's Scala and Python shells

Big data and Spark

- Faster results from analytics has become increasingly important
- Apache Spark is a computing platform designed to be fast and general-purpose, and easy to use

Speed	<ul style="list-style-type: none"> • In-memory computations • Faster than MapReduce for complex applications on disk
Generality	<ul style="list-style-type: none"> • Covers a wide range of workloads on one system • Batch applications (e.g. MapReduce) • Iterative algorithms • Interactive queries and streaming
Ease of use	<ul style="list-style-type: none"> • APIs for Scala, Python, Java • Libraries for SQL, machine learning, streaming, and graph processing • Runs on Hadoop clusters or as a standalone • including the popular MapReduce model

Big data and Spark

There is an explosion of data, and no matter where you look, data is everywhere. You get data from social media such as Twitter feeds, Facebook posts, SMS, and a variety of others. The need to be able to process those data as quickly as possible becomes more important than ever. How can you find out what your customers want and be able to offer it to them right away? You do not want to wait hours for a batch job to complete; you need to have it in minutes or less.

MapReduce has been useful, but the amount of time it takes for the jobs to run is no longer acceptable in many situations. The learning curve to writing a MapReduce job is also difficult as it takes specific programming knowledge and expertise. Also, MapReduce jobs only work for a specific set of use cases. You need something that works for a wider set of use cases.

Apache Spark was designed as a computing platform to be fast, general-purpose, and easy to use. It extends the MapReduce model and takes it to a whole other level.

- The speed comes from the in-memory computations. Applications running in memory allows for a much faster processing and response. Spark is even faster than MapReduce for complex applications on disk.
- This generality covers a wide range of workloads under one system. You can run batch application such as MapReduce type jobs or iterative algorithms that builds upon each other. You can also run interactive queries and process streaming data with your application. In a later slide, you'll see that there are a number of libraries which you can easily use to expand beyond the basic Spark capabilities.
- The ease of use with Spark enables you to quickly pick it up using simple APIs for Scala, Python and Java. As mentioned, there are additional libraries which you can use for SQL, machine learning, streaming, and graph processing. Spark runs on Hadoop clusters such as Hadoop YARN or Apache Mesos, or even as a standalone with its own scheduler.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Ease of use

- To implement the classic **wordcount** in Java MapReduce, you need three classes: the main class that sets up the job, a Mapper, and a Reducer, each about 10 lines long
- For the same **wordcount** program, written in Scala for Spark:

```
val conf = new SparkConf().setAppName("Spark wordcount")
val sc = new SparkContext(conf)
val file = sc.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
    .map(word => (word, 1)).countByKey()
counts.saveAsTextFile("hdfs://...")
```

- With Java, Spark can take advantage of Scala's versatility, flexibility, and its functional programming concepts

Ease of use

Spark supports Scala, Python, and Java programming languages, all now in widespread use among data scientists. The slide shows programming in Scala. Python is widespread among data scientists and in the scientific community, bringing those users on par with Java and Scala developers.

An important aspect of Spark is the ways that it can combine the functionalities of many tools in available in the Hadoop ecosystem to provide a single unifying platform. In addition, the Spark execution model is general enough that a single framework can be used for:

- batch processing operations(similar to that provided by MapReduce),
- stream data processing,
- machine learning,
- SQL-like operations, and
- graph operations.

The result is that many different ways of working with data are available on the same platform, bridging the gap between the work of the classic big data programmer, data engineers and data scientists.

All the same, Spark has its own limitations. There are no universal tools. Thus Spark is not suitable for transaction processing and other ACID types of operations.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Who uses Spark and why?

- Parallel distributed processing, fault tolerance on commodity hardware, scalability, in-memory computing, high level APIs, etc.
- Data scientist
 - Analyze and model the data to obtain insight using ad-hoc analysis
 - Transforming the data into a useable format
 - Statistics, machine learning, SQL
- Data engineers
 - Develop a data processing system or application
 - Inspect and tune their applications
 - Programming with the Spark's API
- Everyone else
 - Ease of use
 - Wide variety of functionality
 - Mature and reliable

Who uses Spark and why?

You may be asking why you would want to use Spark and what you would use it for.

Spark is related to MapReduce in a sense that it expands on Hadoop's capabilities. Like MapReduce, Spark provides parallel distributed processing, fault tolerance on commodity hardware, scalability, etc. Spark adds to the concept with aggressively cached in-memory distributed computing, low latency, high level APIs and stack of high level tools described on the next slide. This saves time and money.

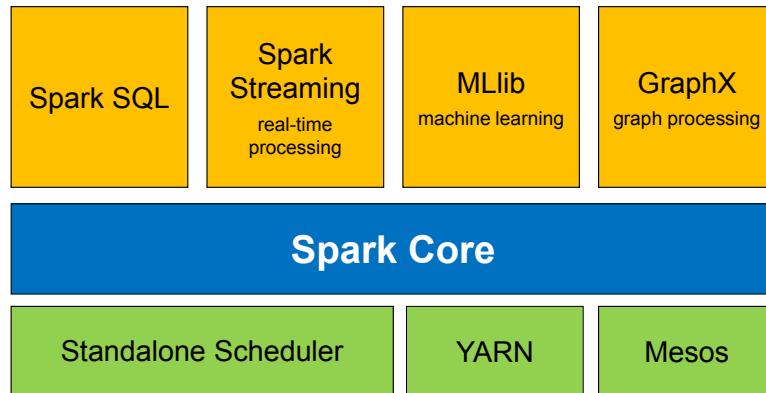
There are two groups that we can consider here who would want to use Spark: Data Scientists and Engineers - and they have overlapping skill sets.

- Data scientists need to analyze and model the data to obtain insight. They need to transform the data into something they can use for data analysis. They will use Spark for its ad-hoc analysis to run interactive queries that will give them results immediately. Data scientists may have experience using SQL, statistics, machine learning and some programming, usually in Python, MatLab or R. Once the data scientists have obtained insights on the data and determined that there is a need develop a production data processing application, a web application, or some system to act upon the insight, the work is handed over to data engineers.
- Data engineers use Spark's programming API to develop a system that implement business use cases. Spark parallelize these applications across the clusters while hiding the complexities of distributed systems programming and fault tolerance. Data engineers can employ Spark to monitor, inspect, and tune applications.

For everyone else, Spark is easy to use with a wide range of functionality. The product is mature and reliable.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Spark unified stack



Spark unified stack

The Spark core is at the center of the Spark Unified Stack. The Spark core is a general-purpose system providing scheduling, distributing, and monitoring of the applications across a cluster.

The Spark core is designed to scale up from one to thousands of nodes. It can run over a variety of cluster managers including Hadoop YARN and Apache Mesos, or more simply, it can run standalone with its own built-in scheduler.

Spark Core contains basic Spark functionalities required for running jobs and needed by other components. The most important of these is the RDD concept, or resilient distributed dataset, the main element of Spark API. RDD is an abstraction of a distributed collection of items with operations and transformations applicable to the dataset. It is resilient because it is capable of rebuilding datasets in case of node failures.

Various add-in components can run on top of the core; these are designed to interoperate closely, letting the users combine them, just like they would any libraries in a software project. The benefit of the Spark Unified Stack is that all the higher layer components will inherit the improvements made at the lower layers. Example: Optimization to the Spark Core will speed up the SQL, the streaming, the machine learning and graph processing libraries as well.

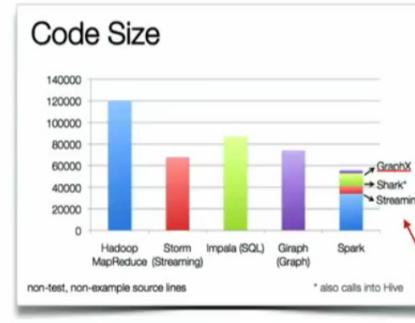
Spark simplifies the picture by providing many of Hadoop ecosystem functions through several purpose-built components. These are Spark Core, Spark SQL, Spark Streaming, Spark MLlib, and Spark GraphX:

- **Spark SQL** is designed to work with the Spark via SQL and HiveQL (a Hive variant of SQL). Spark SQL allows developers to intermix SQL with Spark's programming language supported by Python, Scala, and Java.
- **Spark Streaming** provides processing of live streams of data. The Spark Streaming API closely matches that of the Spark Core's API, making it easy for developers to move between applications that process data stored in memory vs arriving in real-time. It also provides the same degree of fault tolerance, throughput, and scalability that the Spark Core provides.
- **MLlib** is the machine learning library that provides multiple types of machine learning algorithms. These algorithms are designed to scale out across the cluster as well. Supported algorithms include logistic regression, naive Bayes classification, SVM, decision trees, random forests, linear regression, k-means clustering, and others.
- **GraphX** is a graph processing library with APIs to manipulate graphs and performing graph-parallel computations. Graphs are data structures comprised of vertices and edges connecting them. GraphX provides functions for building graphs and implementations of the most important algorithms of the graph theory, like page rank, connected components, shortest paths, and others.

"If you compare the functionalities of Spark components with the tools in the Hadoop ecosystem, you can see that some of the tools are suddenly superfluous. For example, Apache Storm can be replaced by Spark Streaming, Apache Giraph can be replaced by Spark GraphX and Spark MLlib can be used instead of Apache Mahout. Apache Pig, and Apache Sqoop are not really needed anymore, as the same functionalities are covered by Spark Core and Spark SQL. But even if you have legacy Pig workflows and need to run Pig, the Spork project enables you to run Pig on Spark." - Bonači, M., & Zečević, P. (2015). *Spark in action*. Greenwich, CT: Manning Publications. ("Spork" is Apache Pig on Spark, see <https://github.com/sigmoidanalytics/spork>).

Brief history of Spark

- Timeline
 - 2002: MapReduce @ Google
 - 2004: MapReduce paper
 - 2006: Hadoop @ Yahoo
 - 2008: Hadoop Summit
 - 2010: Spark paper
 - 2014: Apache Spark top-level
- MapReduce started a general batch processing paradigm, but had its limitations:
 - difficulty programming in MapReduce
 - batch processing did not fit many use cases
- MR spawned a lot of specialized systems (Storm, Impala, Giraph, etc.)



The State of Spark, and Where We're Going Next
 Matei Zaharia
 Spark Summit (2013)
youtu.be/nU6vO2EJAb4

used as libs, instead of
specialized systems

Brief history of Spark

MapReduce was developed over a decade ago as a fault tolerant framework that ran on commodity systems.

MapReduce started off as a general batch processing system, but there are two major limitations.

- Difficulty in programming directly in MR.
- Batch jobs do not fit many use cases.

This spawned specialized systems to handle the other needed use cases. When you try to combine these third party systems into your applications, there is a lot of overhead.

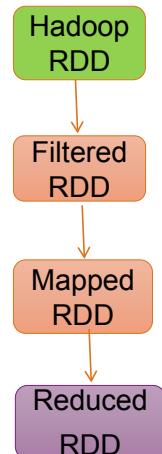
Spark comes out about a decade later with similar framework to run data processing on commodity systems also using a fault tolerant framework.

Taking a look at the code size of some applications on the graph on this slide, you can see that Spark requires a considerable amount less. Even with Spark's libraries, it only adds a small amount of code due to how tightly everything is with little overhead. There is great value to be able to express a wide variety of use cases with few lines of code and the corresponding increase in programmer productivity.

Resilient Distributed Datasets (RDD)

- Spark's primary abstraction: Distributed collection of elements, parallelized across the cluster
- Two types of RDD operations:
 - Transformations
 - Creates a directed acyclic graph (DAG)
 - Lazy evaluations
 - No return value
 - Actions
 - Performs the transformations
 - The action that follows returns a value
- RDD provides fault tolerance
- Has in-memory caching (with overflow to disk)

**Example
RDD flow**



Resilient Distributed Datasets (RDD)

Looking at the core of Spark, Spark's primary core abstraction is called Resilient Distributed Dataset (RDD).

RDD essentially is just a distributed collection of elements that is parallelized across the cluster.

There are two types of RDD operations: transformations and actions.

- Transformations do not return a value. In fact, nothing is evaluated during the definition of transformation statements. Spark just creates the definition of a transformation that will be evaluated later at runtime. This is called this lazy evaluation. The transformation is stored as a directed acyclic graphs (DAG).
- Actions actually perform the evaluation. The action is transformations are performed along with the work that is called for to consume or produce RDDs. Actions return values. For example, you can do a count on a RDD, to get the number of elements within and that value is returned.

The fault tolerance aspect of RDDs allows Spark to reconstruct the transformations used to build the lineage to get back any lost data.

In the example RDD flow shown on the slide, the first step loads the dataset from Hadoop. Successive steps apply transformations on this data such as filter, map, or reduce. Nothing actually happens until an action is called. The DAG is just updated each time until an action is called. This provides fault tolerance; for example, if a node goes offline, all it needs to do when it comes back online is to re-evaluate the graph to where it left off.

In-memory caching is provided with Spark to enable the processing to happen in memory. If the RDD does not fit in memory, it will spill to disk.

Spark jobs and shell

- Spark jobs can be written in Scala, Python, or Java; APIs are available for all three
- Spark shells are provided for Scala (spark-shell) and Python (pyspark)
- Spark's native language is Scala, so it is natural to write Spark applications using Scala

The image contains two side-by-side terminal window screenshots. The left window is titled 'spark@rvm:' and shows Scala shell output. It includes logs from 'spark.SecurityManager' and 'spark.HttpServer', followed by a welcome message for Scala version 1.2.1, and a note about using Scala 2.10.4. The right window is titled 'root@rvm:' and shows Python shell output. It includes logs from 'util.Utils', 'storage.BlockManagerMaster', and 'storage.BlockManagerActor', followed by a welcome message for Python version 2.6.6, and a note about using Python 2.6.6.

```
[spark@rvm ~]$ spark-shell
15/05/18 08:52:34 INFO spark.SecurityManager: Changing view acls to: spark
15/05/18 08:52:34 INFO spark.SecurityManager: Changing modify acls to: spark
15/05/18 08:52:34 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(spark); users with modify permissions: Set(spark)
15/05/18 08:52:34 INFO spark.HttpServer: Starting HTTP Server
15/05/18 08:52:35 INFO server.Server: jetty-8.y-z-SNAPSHOT
15/05/18 08:52:35 INFO server.AbstractConnector: Started SocketConnector@0.0.0.0:3072
15/05/18 08:52:35 INFO util.Utils: Successfully started service 'HTTP class server' on port 3072.
Welcome to
  \_\_/\_\_/\_\_/\_\_/\_\_/\_\_
 / \ / \ / \ / \ / \ / \ / \
version 1.2.1

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_65)
Type in expressions to have them evaluated.
Type :help for more information.
15/05/18 08:52:42 INFO spark.SecurityManager: Changing view acls to: spark
15/05/18 08:52:42 INFO spark.SecurityManager: Changing modify acls to: spark
```

```
File Edit View Search Terminal Help
localhost
15/05/18 09:42:45 INFO util.AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp://sparkDriver@rvm.svl.ibm.com:43469/user/HeartbeatReceiver
15/05/18 09:42:45 INFO netty.NettyBlockTransferService: Server created on 41253
15/05/18 09:42:45 INFO storage.BlockManagerMaster: Trying to register BlockManager
15/05/18 09:42:45 INFO storage.BlockManagerMasterActor: Registering block manager localhost:41253 with 265.4 MB RAM, BlockManagerId<driver>, localhost, 41253)
15/05/18 09:42:45 INFO storage.BlockManagerMaster: Registered BlockManager
15/05/18 09:42:46 WARN shortcircuit.DomainSocketFactory: The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
15/05/18 09:42:46 INFO scheduler.EventLoggingListener: Logging events to hdfs://rvm.svl.ibm.com:8020/lop/apps/4.0.0.0/spark/logs/history-server/local-1431967365
438
Welcome to
  \_\_/\_\_/\_\_/\_\_/\_\_/\_\_
 / \ / \ / \ / \ / \ / \ / \
version 1.2.1

Using Python version 2.6.6 (r266:84292, Nov 21 2013 10:50:32)
SparkContext available as sc.
>>> [REDACTED]
```

Apache Spark

© Copyright IBM Corporation 2015

Spark jobs and shell

Spark jobs can be written in Scala, Python or Java. Spark shells are available for Scala (spark-shell) and Python (pyspark). This course will not teach how to program in each specific language, but will cover how to use them within the context of Spark. It is recommended that you have at least some programming background to understand how to code in any of these.

If you are setting up the Spark cluster yourself, you will have to make sure that you have a compatible version of it. This information can be found on Spark's website. In the lab environment, everything has been set up for you - all you do is launch up the shell and you are ready to go.

Spark itself is written in the Scala language, so it is natural to use Scala to write Spark applications. This course will cover code examples from by Scala, Python, and Java. Java 8 actually supports the functional programming style to include lambdas, which concisely captures the functionality that are executed by the Spark engine. This bridges the gap between Java and Scala for developing applications on Spark. Java 6 is 7 is supported, but would require more work and an additional library to get the same amount of functionality as you would using Scala or Python.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Brief overview of Scala

- Everything is an Object:
 - Primitive types such as numbers or Boolean
 - Functions
- Numbers are objects
 - $1 + 2 * 3 / 4 \rightarrow (1).+(((2).*(3))./(x)))$
 - Where the +, *, / are valid identifiers in Scala
- Functions are objects
 - Pass functions as arguments
 - Store them in variables
 - Return them from other functions
- Function declaration
 - `def functionName ([list of parameters]) : [return type]`

Brief overview of Scala

Everything in Scala is an object. The primitive types defined by Java such as int or boolean are objects in Scala. Functions are objects in Scala and play an important role in how applications are written for Spark.

Numbers are objects. As an example, in the expression that you see here: $1 + 2 * 3 / 4$ actually means that the individual numbers invoke the various identifiers +,-,*,/ with the other numbers passed in as arguments using the dot notation.

Functions are objects. You can pass functions as arguments into another function. You can store them as variables. You can return them from other functions. The function declaration is the function name followed by the list of parameters and then the return type.

If you want to learn more about Scala, check out its website for tutorials and guide. Throughout this course, you will see examples in Scala that will have explanations on what it does. Remember, the focus of this unit is on the context of Spark, and is not intended to teach Scala, Python or Java.

References for learning Scala:

- Horstmann, C. S. (2012). *Scala for the impatient*. Upper Saddle River, NJ: Addison-Wesley Professional
- Odersky, M., Spoon, L., & Venners, B. (2011). *Programming in Scala: A Comprehensive Step-by-Step Guide* (2nd ed.). Walnut Creek, CA: Artima Press

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Scala: anonymous functions, aka Lambda functions

- Lambda, or => syntax: functions without a name created for one-time use to pass to another function
- The left side of the right arrow => is where the argument are placed resides (no arguments in the example)
- Right side of the arrow is the body of the function (here the `println` statement)

```

1. object Timer {
2.   def oncePerSecond(callback: () => Unit) {
3.     while (true) { callback(); Thread sleep 1000 }
4.   }
5.   def timeFlies() {
6.     println("time flies like an arrow...")
7.   }
8.   def main(args: Array[String]) {
9.     oncePerSecond(timeFlies)
10.   }
11. }
```



```

1. object TimerAnonymous {
2.   def oncePerSecond(callback: () => Unit) {
3.     while (true) { callback(); Thread sleep 1000 }
4.   }
5.   def main(args: Array[String]) {
6.     oncePerSecond(() =>
7.       println("time flies like an arrow..."))
8.   }
9. }
```

Scala: anonymous functions, aka Lambda functions

Anonymous functions are very common in Spark applications. Essentially, if the function you need is only going to be required once, there is really no value in naming it. Just use it anonymously on the go and forget about it. For example, if you have a `timeFlies` function and in it, you just print a statement to the console. In another function, `oncePerSecond`, you need to call this `timeFlies` function. Without anonymous functions, you would code it like the top example by defining the `timeFlies` function. Using the anonymous function capability, you just provide the function with arguments, the right arrow, and the body of the function after the right arrow as in the bottom example. Because this is the only place you will be using this function, you do not need to name the function.'

Python's syntax is relatively convenient and easy to work with, but aside from the basic structure of the language Python is also sprinkled with small syntax structures that make certain tasks especially convenient. The `lambda` keyword/function construct is one of these, where the creators call it "syntactical candy."

References:

- <http://docs.scala-lang.org/tutorials/scala-for-java-programmers.html>

Computing wordcount using Lambda functions

- The classic wordcount program can be written with anonymous (Lambda) functions
- Three functions are needed:
 - **Tokenize** each line into words (with space as delimiter)
 - **Map** to produce the <word, 1> key/value pair from each word that is read
 - **Reduce** to aggregate the counts for each word individually (reduceByKey)
- The results are written to HDFS

```
text_file = spark.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" "))
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

- Lambda functions can be used with Scala, Python, and Java v8; this example is Scala

Computing wordcount using Lambda functions

The Lambda or => syntax is a shorthand way to define functions inline in Python and Scala. With Spark you can define anonymous functions separately and then pass the name to Spark. For example, in Python:

```
def hasIBM(line):
    return "IBM" in line;
IBMLines = lines.filter(hasIBM)
```

is functionally equivalent to:

```
grep IBM inputfile
```

A common example is a MapReduce wordcount. You first split up the file by words ("tokenize") and then map each word into a key value pair with the word as the key, and the value of 1. Then you reduce by the key, which adds up all the value of the same key, effectively, counting the number of occurrences of that key. Finally, the counts are written out to a file in HDFS.

References:

- École Polytechnique Fédérale de Lausanne (EPFL). (2012). A tour of Scala: Anonymous function syntax, from <http://www.scala-lang.org/old/node/133>
- Apache Software Foundation,(2015). Spark Examples, from <https://spark.apache.org/examples.html>

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Resilient Distributed Dataset (RDD)

- Fault-tolerant collection of elements that can be operated on in parallel.
- RDDs are immutable
- Three methods for creating RDD
 - Parallelizing an existing collection
 - Referencing a dataset
 - Transformation from an existing RDD
- Two types of RDD operations
 - Transformations
 - Actions
- Dataset from any storage supported by Hadoop
 - HDFS, Cassandra, HBase, Amazon S3, etc.
- Types of files supported:
 - Text files, SequenceFiles, Hadoop InputFormat, etc

Resilient Distributed Dataset (RDD)

Major point of interest here:

- Resilient Distributed Dataset (RDD) is Spark's primary abstraction.
- An RDD is a fault tolerant collection of elements that can be parallelized. In other words, they can be made to be operated on in parallel. They are immutable.
- These are the fundamental primary units of data in Spark.
- When RDDs are created, a direct acyclic graph (DAG) is created. This type of operation is called transformations. Transformations makes updates to that graph, but nothing actually happens until some action is called. Actions are another type of operations. We'll talk more about this shortly. The notion here is that the graphs can be replayed on nodes that need to get back to the state it was before it went offline, thus providing fault tolerance. The elements of the RDD can be operated on in parallel across the cluster. Remember, transformations return a pointer to the RDD created and actions return values that comes from the action.

There are three methods for creating a RDD. You can parallelize an existing collection. This means that the data already resides within Spark and can now be operated on in parallel. As an example, if you have an array of data, you can create a RDD out of it by calling the parallelized method. This method returns a pointer to the RDD. So this new distributed dataset can now be operated upon in parallel throughout the cluster.

The second method to create a RDD, is to reference a dataset. This dataset can come from any storage source supported by Hadoop such as HDFS, Cassandra, HBase, Amazon S3, etc.

The third method to create a RDD is from transforming an existing RDD to create a new RDD. In other words, if you have the array of data that you parallelized earlier, and you want to filter out the records available. A new RDD is created using the filter method.

The final point on this slide: Spark supports text files, SequenceFiles, and any other Hadoop InputFormat.

Creating an RDD

- Launch the Spark shell (requires PATH environment v)
`spark-shell`

- Create some data
`val data = 1 to 10000`

- Parallelize that data (creating the RDD)
`val distData = sc.parallelize(data)`

- Perform additional transformations or invoke an action on it.
`distData.filter(...)`

- Alternatively, create an RDD from an external dataset
`val readmeFile = sc.textFile("Readme.md")`

Creating an RDD

Here is a quick example of how to create an RDD from an existing collection of data. In the examples throughout the course, unless otherwise indicated, you will be using Scala to show how Spark works. In the lab exercises, you will get to work with Python and Java as well.

First, launch the Spark shell. This command is located under the /usr/bin directory.

Once the shell is up (with the prompt "scala>"), create some data with values from 1 to 10,000. Then, create an RDD from that data using the parallelize method from the SparkContext, shown as sc on the slide; this means that the data can now be operated on in parallel.

More will be covered on the SparkContext, the sc object that is invoking the parallelized function later, so for now, just know that when you initialize a shell, the SparkContext, sc, is initialized for you to use.

The parallelize method returns a pointer to the RDD. Remember, transformations operations such as parallelize, only returns a pointer to the RDD. It actually will not create that RDD until some action is invoked on it. With this new RDD, you can perform additional transformations or actions on it such as the filter transformation. This is important with large amounts of data (big data), because you do not want to duplicate the data until needed, and certainly not cache it in memory until needed.

Another way to create an RDD is from an external dataset. In the example here, you are creating a RDD from a text file using the textFile method of the SparkContext object. You will see more examples of how to create RDD throughout this course.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Spark's Scala and Python shells

- Spark's shells provides simple ways to learn the APIs and provide a set of powerful tools to analyze data interactively
- Scala shell:
 - Runs on the Java VM & provides a good way to use existing Java libraries
 - To launch the Scala shell: `./bin/spark-shell`
 - The prompt is: `scala>`
 - To read in a text file: `scala> val textFile = sc.textFile("README.md")`
- Python shell:
 - To launch the Python shell: `./bin/pyspark`
 - The prompt is: `>>>`
 - To read in a text file: `>>> textFile = sc.textFile("README.md")`
 - Two additional variations: IPython & the IPython Notebook
- To quit either shell, use: `Ctrl-D` (the EOF character)

Spark's Scala and Python shells

The Spark shell provides a simple way to learn Spark's API. It is also a powerful tool to analyze data interactively. The Shell is available in either Scala, which runs on the Java VM, or Python. To start up Scala, execute the command `spark-shell` from within the Spark's bin directory. To create a RDD from a text file, invoke the `textFile` method with the `sc` object, which is the `SparkContext`.

To start up the shell for Python, you would execute the `pyspark` command from the same bin directory. Then, invoking the `textFile` command will also create a RDD for that text file.

In the lab exercise later, you will start up either of the shells and run a series of RDD transformations and actions to get a feel of how to work with Spark. Later you will get to dive deeper into RDDs.

IPython offers features such as tab completion. For further details: <http://ipython.org>
IPython Notebook is a web-browser based version of IPython.

RDD basic operations

- Loading a file

```
val lines = sc.textFile("hdfs://data.txt")
```

- Applying transformation

```
val lineLengths = lines.map(s => s.length)
```

- Invoking action

```
val totalLengths = lineLengths.reduce((a,b) => a + b)
```

- View the DAG

lineLengths.toDebugString

```
res5: String =
  MappedRDD[4] at map at <console>:16 (3 partitions)
    MappedRDD[3] at map at <console>:16 (3 partitions)
      FilteredRDD[2] at filter at <console>:14 (3 partitions)
        MappedRDD[1] at textFile at <console>:12 (3 partitions)
          HadoopRDD[0] at textFile at <console>:12 (3 partitions)|
```

RDD basic operations

You have seen how to load a file from an external dataset. This time, however, you are loading a file from the hdfs. Loading the file creates a RDD, which is only a pointer to the file. The dataset is not loaded into memory yet. Nothing will happen until some action is called. The transformation basically updates the direct acyclic graph (DAG).

So the transformation here is saying map each line s, to the length of that line. Then, the action operation is reducing it to get the total length of all the lines. When the action is called, Spark goes through the DAG and applies all the transformation up until that point, followed by the action and then a value is returned back to the caller.

Directed Acyclic Graph (DA)

On this slide, you will see how to view the DAG of any particular RDD. A DAG is essentially a graph of the business logic and does not get executed until an action is called - often called lazy evaluation.

To view the DAG of a RDD after a series of transformation, use the `toDebugString` method as you see here on the slide. It will display the series of transformation that Spark will go through once an action is called. You read it from the bottom up. In the sample DAG shown on the slide, you can see that it starts as a `textFile` and goes through a series of transformation such as `map` and `filter`, followed by more `map` operations. Remember, that it is this behavior that allows for fault tolerance. If a node goes offline and comes back on, all it has to do is just grab a copy of this from a neighboring node and rebuild the graph back to where it was before it went offline.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

What happens when an action is executed? (1 of 8)

```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")

// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))

// Caching
messages.cache()

// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```

Driver

Worker

Worker

Worker

What happens when an action is executed?

In the next several slides, you will see at a high level what happens when an action is executed.

Let's look at the code first. The goal here is to analyze some log files. The first line you load the log from the hadoop file system. The next two lines you filter out the messages within the log errors. Before you invoke some action on it, you tell it to cache the filtered dataset; it doesn't actually cache it yet as nothing has been done up until this point.

Then you do more filters to get specific error messages relating to mysql and php followed by the count action to find out how many errors were related to each of those filters.

What happens when an action is executed? (2 of 8)

```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")

// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))

//Caching
messages.cache()

// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```

Driver

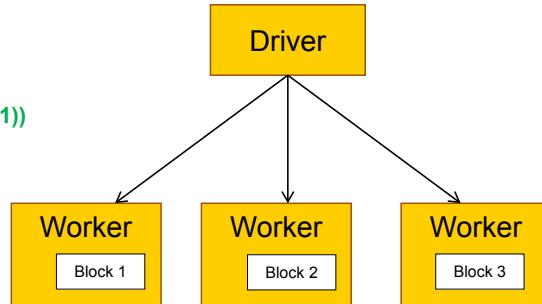


The data is partitioned
into different blocks

In reviewing the steps, the first thing that happens when you load in the text file is the data is partitioned into different blocks across the cluster.

What happens when an action is executed? (3 of 8)

```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Cache
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```

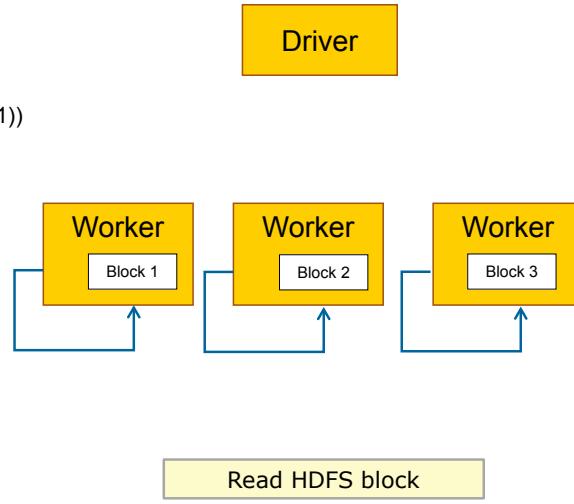


Driver sends the code to be executed on each block

Then the driver sends the code to be executed on each block. In the example, it would be the various transformations and actions that will be sent out to the workers. Actually, it is the executor on each workers that is going to be performing the work on each block. You will see a bit more on executors later.

What happens when an action is executed? (4 of 8)

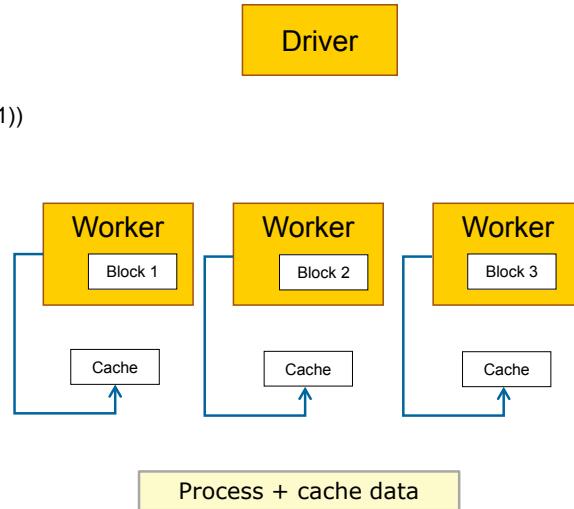
```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



The executors read the HDFS blocks to prepare the data for the operations in parallel.

What happens when an action is executed? (5 of 8)

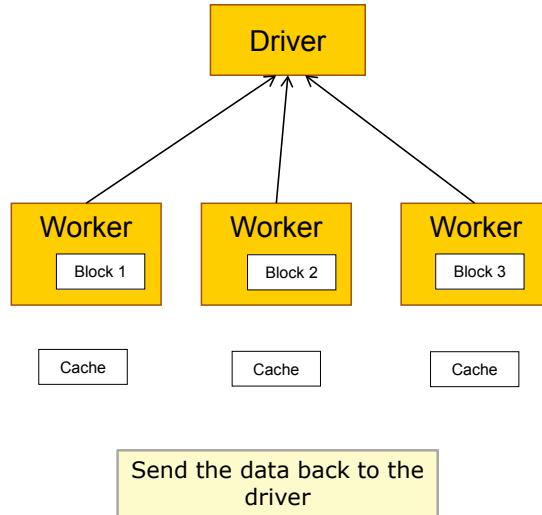
```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



After a series of transformations, you want to cache the results up until that point into memory. A cache is created.

What happens when an action is executed? (6 of 8)

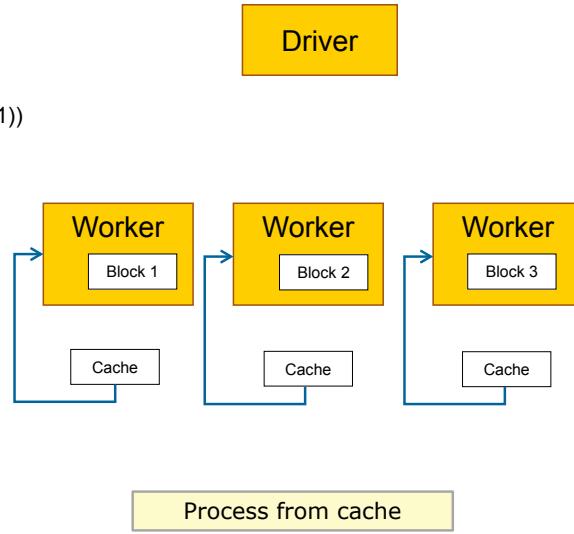
```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



After the first action completes, the results are sent back to the driver. In this case, you are looking for messages that relate to mysql. This is then returned back to the driver.

What happens when an action is executed? (7 of 8)

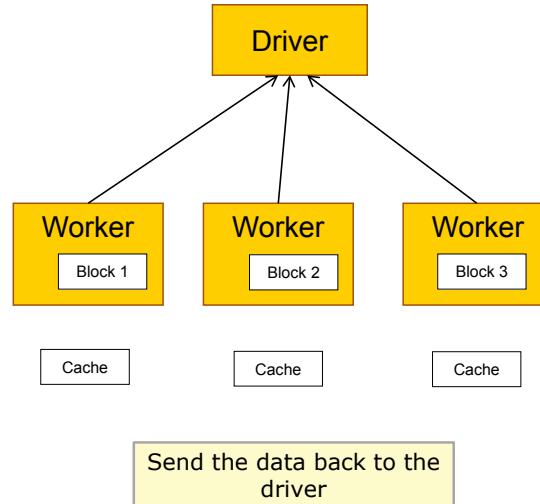
```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



To process the second action, Spark will use the data on the cache; it does not need to go to the HDFS data again. It just reads it from the cache and processes the data from there.

What happens when an action is executed? (8 of 8)

```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



Finally the results are sent back to the driver and you have completed a full cycle.

RDD operations: transformations

- These are some of the transformations available - the full set can be found on Spark's website.
- Transformations are lazy evaluations
- Returns a pointer to the transformed RDD

Transformation	Meaning
map(func)	Return a new dataset formed by passing each element of the source through a function func.
filter(func)	Return a new dataset formed by selecting those elements of the source on which func returns true.
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items. So func should return a Seq rather than a single item
join(otherDataset, [numTasks])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key.
reduceByKey(func)	When called on a dataset of (K, V) pairs, returns a dataset of (K,V) pairs where the values for each key are aggregated using the given reduce function func
sortByKey([ascending],[numTasks])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K,V) pairs sorted by keys in ascending or descending order.

RDD operations: transformations

These are some of the transformations available; the full set can be found on Spark's website. The Spark Programming Guide can be found at <https://spark.apache.org/docs/latest/programming-guide.html> and transformations can be found at <https://spark.apache.org/docs/latest/programming-guide.html#transformations>.

Remember that Transformations are essentially lazy evaluations. Nothing is executed until an action is called. Each transformation function basically updates the graph and when an action is called, the graph is executed. Transformation returns a pointer to the new RDD.

Some of the less obvious are:

- The **flatMap** function is similar to map, but each input can be mapped to 0 or more output items. What this means is that the returned pointer of the func method, should return a sequence of objects, rather than a single item. It would mean that the flatMap would flatten a list of lists for the operations that follows. Basically this would be used for MapReduce operations where you might have a text file and each time a line is read in, you split that line up by spaces to get individual keywords. Each of those lines ultimately is flatten so that you can perform the map operation on it to map each keyword to the value of one.
- The **join** function combines two sets of key value pairs and return a set of keys to a pair of values from the two initial set. For example, you have a K,V pair and a K,W pair. When you join them together, you will get a K, (V,W) set.
- The **reduceByKey** function aggregates on each key by using the given reduce function. This is something you would use in a WordCount to sum up the values for each word to count its occurrences.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

RDD operations: actions

- On the other hand, actions return values

Action	Meaning
collect()	Return all the elements of the dataset as an array of the driver program. This is usually useful after a filter or another operation that returns a sufficiently small subset of data.
count()	Return the number of elements in a dataset.
first()	Return the first element of the dataset
take(n)	Return an array with the first n elements of the dataset.
foreach(func)	Run a function func on each element of the dataset.

RDD operations: actions

Action returns values. Again, you can find more information on Spark's website. The full set of functions are available at <https://spark.apache.org/docs/latest/programming-guide.html#actions>.

The slide shows a subset:

- The collect function returns all the elements of the dataset as an array of the driver program. This is usually useful after a filter or another operation that returns a significantly small subset of data to make sure your filter function works correctly.
- The count function returns the number of elements in a dataset and can also be used to check and test transformations.
- The take(n) function returns an array with the first n elements. Note that this is currently not executed in parallel. The driver computes all the elements.
- The foreach(func) function run a function func on each element of the dataset.

RDD persistence

- Each node stores partitions of the cache that it computes in memory
- Reuses them in other actions on that dataset (or derived datasets)
 - Future actions are much faster (often by more than 10x)
- Two methods for RDD persistence: `persist()`, `cache()`

Storage Level	Meaning
MEMORY_ONLY	Store as deserialized Java objects in the JVM. If the RDD does not fit in memory, part of it will be cached. The other will be recomputed as needed. This is the default. The <code>cache()</code> method uses this.
MEMORY_AND_DISK	Same except also store on disk if it doesn't fit in memory. Read from memory and disk when needed.
MEMORY_ONLY_SER	Store as serialized Java objects (one byte array per partition). Space efficient, but more CPU intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_AND_DISK but stored as serialized objects.
DISK_ONLY	Store only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc	Same as above, but replicate each partition on two cluster nodes
OFF_HEAP (experimental)	Store RDD in serialized format in Tachyon.

RDD persistence

Now to want to get a bit into RDD persistence. You have seen this used already; it is the `cache` function. The `cache` function is actually the default of the `persist` function; `cache()` is essentially just `persist` with `MEMORY_ONLY` storage.

One of the key capability of Spark is its speed through persisting or caching. Each node stores any partitions of the cache and computes it in memory. When a subsequent action is called on the same dataset, or a derived dataset, it uses it from memory instead of having to retrieve it again. Future actions in such cases are often 10 times faster. The first time a RDD is persisted, it is kept in memory on the node. Caching is fault tolerant because if any of the partition is lost, it will automatically be recomputed using the transformations that originally created it.

There are two methods to invoke RDD persistence. `persist()` and `cache()`. The `persist()` method allows you to specify a different storage level of caching. For example, you can choose to persist the data set on disk, persist it in memory but as serialized objects to save space, etc. Again the `cache()` method is just the default way of using persistence by storing deserialized objects in memory.

The table here shows the storage levels and what it means. Basically, you can choose to store in memory or memory and disk. If a partition does not fit in the specified cache location, then it will be recomputed on the fly. You can also decide to serialized the objects before storing this. This is space efficient, but will require the RDD to deserialized before it can be read, so it takes up more CPU workload. There's also the option to replicate each partition on two cluster nodes. Finally, there is an experimental storage level storing the serialized object in Tachyon. This level reduces garbage collection overhead and allows the executors to be smaller and to share a pool of memory. You can read more about this on Spark's website.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Best practices for which storage level to choose

- The default storage level (MEMORY_ONLY) is the best
- Otherwise MEMORY_ONLY_SER and a fast serialization library
- Don't spill to disk unless the functions that computed your datasets are expensive, or they filter a large amount of the data - recomputing a partition may be as fast as reading it from disk.
- Use the replicated storage levels if you want fast fault recovery (such as if using Spark to serve requests from a web application)
- All the storage levels provide full fault tolerance by recomputing lost data, but the replicated ones let you continue running tasks on the RDD without waiting to recompute a lost partition
- The experimental OFF_HEAP mode has several advantages:
 - Allows multiple executors to share the same pool of memory in Tachyon
 - Reduces garbage collection costs
 - Cached data is not lost if individual executors crash.

Best practices for which storage level to choose

There are a lot of rules on this page; use them as a reference when you have to decide the type of storage level. There are tradeoffs between the different storage levels. You should analyze your current situation to decide which level works best. You can find this information on Spark's website.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

The primary rules are:

- If the RDDs fit comfortably with the default storage level (MEMORY_ONLY), leave them that way - the most CPU-efficient option, allowing operations on the RDDs to run as fast as possible. Basically if your RDD fits within the default storage level, by all means, use that. It is the fastest option to fully take advantage of Spark's design.
- Otherwise use MEMORY_ONLY_SER and a fast serialization library to make objects more space-efficient, but still reasonably fast to access.
- Do not spill to disk unless the functions that compute your datasets are expensive or it requires a large amount of space.
- If you want fast recovery, use the replicated storage levels. Note that all levels of storage are fully fault tolerant, but would still require the recomputing of the data. If you have a replicated copy, you can continue to work while Spark is recomputing a lost partition.
- In environments with high amounts of memory or multiple applications, the experimental OFF_HEAP mode has several advantages. Use Tachyon if your environment has high amounts of memory or multiple applications. It allows you to share the same pool of memory and significantly reduces garbage collection costs. Also, the cached data is not lost if the individual executors crash.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Shared variables and key-value pairs

- When a function is passed from the driver to a worker, normally a separate copy of the variables are used ("pass by value").
- Two types of variables:
 - Broadcast variables
 - Read-only copy on each machine
 - Distribute broadcast variables using efficient broadcast algorithms
 - Accumulators
 - Variables added through an associative operation
 - Implement counters and sums
 - Only the driver can read the accumulators value
 - Numeric types accumulators. Extend for new types.

Scala: key-value pairs

```
val pair = ('a', 'b')
pair._1 // will return 'a'
pair._2 // will return 'b'
```

Python: key-value pairs

```
pair = ('a', 'b')
pair[0] # will return 'a'
pair[1] # will return 'b'
```

Java: key-value pairs

```
Tuple2 pair = new Tuple2('a', 'b');
pair._1 // will return 'a'
pair._2 // will return 'b'
```

Shared variables and key-value pairs

On this page and the next, you will review Spark shared variables and the type of operations you can do on key-value pairs.

Spark provides two limited types of shared variables for common usage patterns: broadcast variables and accumulators. Normally, when a function is passed from the driver to a worker, a separate copy of the variables are used for each worker. Broadcast variables allow each machine to work with a read-only variable cached on each machine. Spark attempts to distribute broadcast variables using efficient algorithms. As an example, broadcast variables can be used to give every node a copy of a large dataset efficiently.

The other shared variables are accumulators. These are used for counters in sums that works well in parallel. These variables can only be added through an associated operation. Only the driver can read the accumulators value, not the tasks. The tasks can only add to it. Spark supports numeric types but programmers can add support for new types. As an example, you can use accumulator variables to implement counters or sums, as in MapReduce.

Last, but not least, key-value pairs are available in Scala, Python and Java. In Scala, you create a key-value pair RDD by typing `val pair = ('a', 'b')`. To access each element, invoke the `._` notation. This is not zero-index, so the `._1` will return the value in the first index and `._2` will return the value in the second index. Java is also very similar to Scala where it is not zero-index. You create the `Tuple2` object in Java to create a key-value pair. In Python, it is a zero-index notation, so the value of the first index resides in index 0 and the second index is 1.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Programming with key-value pairs

- There are special operations available on RDDs of key-value pairs
 - Grouping or aggregating elements by a key
- Tuple2 objects created by writing (a, b)
 - Must import org.apache.spark.SparkContext._
- PairRDDFunctions contains key-value pair operations
 - reduceByKey((a, b) => a + b)
- Custom objects as key in key-value pair requires a custom equals() method with a matching hashCode() method.
- Example:

```
val textFile = sc.textFile("...")  
val readmeCount = textFile.flatMap(line =>  
    line.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
```

Programming with key-value pairs

There are special operations available to RDDs of key-value pairs. In an application, you must remember to import the **SparkContext** package to use **PairRDDFunctions** such as **reduceByKey**.

The most common ones are those that perform grouping or aggregating by a key. RDDs containing the Tuple2 object represents the key-value pairs. Tuple2 objects are simple created by writing (a, b) as long as you import the library to enable Spark's implicit conversion.

If you have custom objects as the key inside your key-value pair, remember that you will need to provide your own equals() method to do the comparison as well as a matching hashCode() method.

In the example, you have a **textFile** that is just a normal RDD. You then perform some transformations on it and it creates a PairRDD which allows it to invoke the **reduceByKey** method that is part of the PairRDDFunctions API.

Programming with Spark

- You have reviewed accessing Spark with interactive shells:
 - spark-shell (for Scala)
 - pyspark (for Python)
- Next you will review programming with Spark with:
 - Scala
 - Python
 - Java
- Compatible versions of software are needed
 - Spark 1.3.1 uses Scala 2.10. To write applications in Scala, you will need to use a compatible Scala version (e.g. 2.10.X)
 - Spark 1.x works with Python 2.6 or higher (but not yet with Python 3)
 - Spark 1.x works with Java 6 and higher - and Java 8 supports lambda expressions

Programming with Spark

Compatibility of Spark with various versions of the programming languages is important.

Note also that as new releases of the Open Data Platform are released, you should revisit the issue of compatibility of languages to work with the new versions of Spark.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

SparkContext

- The SparkContext is the main entry point for Spark functionality; it represents the connection to a Spark cluster
- Use the SparkContext to create RDDs, accumulators, and broadcast variables on that cluster
- With the Spark shell, the SparkContext, sc, is automatically initialized for you to use
- But in a Spark program, you need to add code to import some classes and implicit conversions into your program:

```
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext._  
import org.apache.spark.SparkConf
```

SparkContext

The **SparkContext** is the main entry point to everything Spark. It can be used to create RDDs and shared variables on the cluster. When you start up the Spark Shell, the SparkContext is automatically initialized for you with the variable sc. For a Spark application, you must first import some classes and implicit conversions and then create the SparkContext object.

The three import statements for Scala are shown on the slide.

Linking with Spark: Scala

- Spark applications require certain dependencies.
- Needs a compatible Scala version to write applications.
 - For example, Spark 1.1.1 uses Scala 2.10.
- To write a Spark application, you need to add a Maven dependency on Spark.
 - Spark is available through Maven Central at:

```
groupId = org.apache.spark
artifactId = spark-core_2.10
version = 1.1.1
```

- To access a HDFS cluster, you need to add a dependency on hadoop-client for your version of HDFS

```
groupId = org.apache.hadoop
artifactId = hadoop-client
version = <your-hdfs-version>
```

Linking with Spark: Scala

Each Spark application you create requires certain dependencies. Over the next three slides you will review how to link to those dependencies depending on which programming language you decide to use.

To link with Spark using Scala, you must have a compatible version of Scala with the Spark you choose to use. For example, Spark 1.1.1 uses Scala 2.10, so make sure that you have Scala 2.10 if you wish to write applications for Spark 1.1.1.

To write a Spark application, you must add a Maven dependency on Spark. The information is shown on the slide here. If you wish to access a Hadoop cluster, you need to add a dependency to that as well.

In the lab environment, this will already be set up for you. The information on this page is important if you want to set up a Spark stand-alone environment or your own Spark cluster.

Initializing Spark: Scala

- Build a SparkConf object that contains information about your application


```
val conf = new SparkConf().setAppName(appName).setMaster(master)
```
- The appName parameter → Name for your application to show on the cluster UI
- The master parameter → is a Spark, Mesos, or YARN cluster URL (or a special "local" string to run in local mode)
 - In testing, you can pass "local" to run Spark.
 - local[16] will allocate 16 cores
 - In production mode, do not hardcode master in the program. Launch with spark-submit and provide it there.
- Then, you need to create the SparkContext object.

```
new SparkContext(conf)
```

Initializing Spark: Scala

Once you have the dependencies established, the first thing is to do in your Spark application before you can initialize Spark is to build a SparkConf object. This object contains information about your application.

For example,

```
val conf = new SparkConf().setAppName(appName).setMaster(master)
```

You set the application name and tell it which is the master node. The master parameter can be a standalone Spark distribution, Mesos, or a YARN cluster URL. You can also decide to use the local keyword string to run it in local mode. In fact, you can run local[16] to specify the number of cores to allocate for that particular job or Spark shell as 16.

For production mode, you would not want to hardcode the master path in your program. Instead, launch it as an argument to the spark-submit command.

Once you have the SparkConf all set up, you pass it as a parameter to the SparkContext constructor to create it.

Linking with Spark: Python

- Spark 1.x works with Python 2.6 or higher (but not yet with Python 3)
- Uses the standard CPython interpreter, so C libraries like NumPy can be used.
- To run Spark applications in Python, use the bin/spark-submit script located in the Spark directory.
 - Load Spark's Java/Scala libraries
 - Allow you to submit applications to a cluster
- If you want to access HDFS, you need to use a build of PySpark linking to your version of HDFS.
- Import some Spark classes

```
from pyspark import SparkContext, SparkConf
```

Linking with Spark: Python

Spark 1.2.1 works with Python 2.6 or higher, but not Python 3 yet. It uses the standard CPython interpreter, so C libraries like NumPy can be used.

Check which version of Spark you have when you enter an environment that uses it.

To run Spark applications in Python, use the bin/spark-submit script located in the Spark's home directory. This script will load the Spark's Java/Scala libraries and allow you to submit applications to a cluster. If you want to use HDFS, you will have to link to it as well. In the lab environment, you will not need to do this as Spark is bundled with it. You also need to import some Spark classes shown here.

From the Apache Spark website: "It would be nice to have Python 3 support in PySpark, provided that we can do it in a way that maintains backwards-compatibility with Python 2.6." (<https://issues.apache.org/jira/browse/SPARK-4897>).

Initializing Spark: Python

- Build a SparkConf object that contains information about your application

```
conf = SparkConf().setAppName(appName).setMaster(master)
```

- The appName parameter → Name for your application to show on the cluster UI
- The master parameter → is a Spark, Mesos, or YARN cluster URL (or a special "local" string to run in local mode)
 - In production mode, do not hardcode master in the program. Launch with spark-submit and provide it there.
 - In testing, you can pass "local" to run Spark.
- Then, you need to create the SparkContext object.

```
sc = SparkContext(conf=conf)
```

Initializing Spark: Python

Here's the information for Python. It is pretty much the same information as Scala. The syntax here is slightly different, otherwise, you are required to set up a SparkConf object to pass as a parameter to the SparkContext object. You are also recommended to pass the master parameter as an argument to the spark-submit operation.

Linking with Spark: Java

- Spark 1.1.1 works with Java 6 and higher.
 - Java 8 supports lambda expressions
- Add a dependency on Spark
 - Available through Maven Central at:

```
groupId = org.apache.spark
artifactId = spark-core_2.10
version = 1.1.1
```
- If you want to access an HDFS cluster, you must add the dependency as well.


```
groupId = org.apache.hadoop
artifactId = hadoop-client
version = <your-hdfs-version>
```
- Import some Spark classes

```
import org.apache.spark.api.java.JavaSparkContext
import org.apache.spark.api.java.JavaRDD
import org.apache.spark.SparkConf
```

Linking with Spark: Java

Spark 1.2.1 works with Java 6 and higher. If you are using Java 8, Spark supports lambda expressions for concisely writing functions. Otherwise, you can use the `org.apache.spark.api.java.function` package with older Java versions.

As with Scala, you need to a dependency on Spark, which is available through Maven Central. If you wish to access an HDFS cluster, you must add the dependency there as well. Last, but not least, you need to import some Spark classes.

Initializing Spark: Java

- Build a `SparkConf` object that contains information about your application

```
SparkConf conf = new SparkConf().setAppName(appName).setMaster(master)
```

- The `appName` parameter → Name for your application to show on the cluster UI
- The `master` parameter → is a Spark, Mesos, or YARN cluster URL (or a special "local" string to run in local mode)
 - In production mode, do not hardcode `master` in the program. Launch with `spark-submit` and provide it there.
 - In testing, you can pass "local" to run Spark.
- Then, you will need to create the `JavaSparkContext` object.

```
JavaSparkContext sc = new JavaSparkContext(conf);
```

Initializing Spark: Java

Here is the same information for Java. Following the same idea, you need to create the `SparkConf` object and pass that to the `SparkContext`, which in this case, would be a `JavaSparkContext`. Remember, when you imported statements in the program, you imported the `JavaSparkContext` libraries.

Passing functions to Spark

- Spark's API relies on heavily passing functions in the driver program to run on the cluster
- Three methods
 - Anonymous function syntax
 $(x: \text{ Int}) \Rightarrow x + 1$
 - Static methods in a global singleton object

```
object MyFunctions {
    def func1 (s: String): String = {...}
}
myRdd.map(MyFunctions.func1)
```
 - Passing by reference, to avoid sending the entire object, consider copying the function to a local variable
 $\text{val field} = \text{"Hello"}$
 - Avoid:
 $\text{def doStuff(rdd: RDD[String]):RDD[String] = \{rdd.map(x => field + x)\}}$
 - Consider:
 $\text{def doStuff(rdd: RDD[String]):RDD[String] = \{$
 $\text{val field_ = this.field}$
 $\text{rdd.map(x => field_ + x)\}}$

Passing functions to Spark

Passing functions to Spark is important to understand as you begin to think about the business logic of your application.

The design of Spark's API relies heavily on passing functions in the driver program to run on the cluster. When a job is executed, the Spark driver needs to tell its worker how to process the data.

There are three methods that you can use to pass functions.

- The first method to do this is using an anonymous function syntax. You saw briefly what an anonymous function is in the first lesson. This is useful for short pieces of code. For example, here we define the anonymous function that takes in a parameter `x` of type `Int` and add one to it. Essentially, anonymous functions are useful for one-time use of the function. In other words, you do not need to explicitly define the function to use it. You define as you go. Again, the left side of the `=>` are the parameters or the argument. The right side of the `=>` is the body of the function.
- Another method to pass functions around Spark is to use static methods in a global singleton object. This means that you can create a global object, in the example, it is the object `MyFunctions`. Inside that object, you basically define the function `func1`. When the driver requires that function, it only needs to send out the object - the worker will be able to access it. In this case, when the driver sends out the instructions to the worker, it just has to send out the singleton object.
- It is possible to pass reference to a method in a class instance, as opposed to a singleton object. This would require sending the object that contains the class along with the method. To avoid this consider copying it to a local variable within the function instead of accessing it externally.

Example, say you have a field with the string `Hello`. You want to avoid calling that directly inside a function as shown on the slide as `x => field + x`.

Instead, assign it to a local variable so that only the reference is passed along and not the entire object shown `val field_ = this.field`

For an example such as this, it may seem trivial, but imagine if the field object is not a simple text `Hello`, but is something much larger, say a large log file. In that case, passing by reference will have greater value by saving a lot of storage by not having to pass the entire file.

Programming the business logic

- Spark's API available in Scala, Java, or Python.
- Create the RDD from an external dataset or from an existing RDD.
- Transformations and actions to process the data.
- Use RDD persistence to improve performance
- Use broadcast variables or accumulators for specific use cases

```

package org.apache.spark.examples

import org.apache.spark._

object HdfsTest {

  /** Usage: HdfsTest [file] */
  def main(args: Array[String]) {
    if (args.length < 1) {
      System.err.println("Usage: HdfsTest <file>")
      System.exit(1)
    }
    val sparkConf = new SparkConf().setAppName("HdfsTest")
    val sc = new SparkContext(sparkConf)
    val file = sc.textFile(args(0))
    val mapped = file.map(s => s.length).cache()
    for (iter <- 1 to 10) {
      val start = System.currentTimeMillis()
      for (x <- mapped) { x + 2 }
      val end = System.currentTimeMillis()
      println("Iteration " + iter + " took " + (end - start) + " ms")
    }
    sc.stop()
  }
}

```

Programming the business logic

At this point, you should know how to link dependencies with Spark and also know how to initialize the SparkContext. I also touched a little bit on passing functions with Spark to give you a better view of how you can program your business logic. This course will not focus too much on how to program business logics, but there are examples available for you to see how it is done. The purpose is to show you how you can create an application using a simple, but effective examples which demonstrates Spark's capabilities.

Once you have the beginning of your application ready by creating the SparkContext object, you can start to program in the business logic using Spark's API available in Scala, Java, or Python. You create the RDD from an external dataset or from an existing RDD. You use transformations and actions to compute the business logic. You can take advantage of RDD persistence, broadcast variables and/or accumulators to improve the performance of your jobs.

Here's a sample Scala application. You have your import statement. After the beginning of the object, you see that the SparkConf is created with the application name. Then a SparkContext is created. The several lines of code after is creating the RDD from a text file and then performing the Hdfs test on it to see how long the iteration through the file takes. Finally, at the end, you stop the SparkContext by calling the stop() function.

Again, just a simple example to show how you would create a Spark application. You will get to practice this in the exercise.

Running Spark examples

- Spark samples available in the examples directory
- Run the examples: `./bin/run-example SparkPi`
where SparkPi is the name of the sample application
- In Python: `./bin/spark-submit examples/src/main/python/pi.py`

LocalMeans.scala	[SPARK-2434] [MLLIB] vwarning messages that point users to original ML...	als.py	[SPARK-1701] [PySpark] remove slice terminology fr...
LocalR.scala	[SPARK-4047] - Generate runtime warnings for example implementation o...	avro_inputformat.py	SPARK-2626 [DOCS] Stop SparkContext in all exam...
LocalPi.scala	SPARK-1462: Examples of ML algorithms are using deprecated APIs	cassandra_inputformat.py	[SPARK-3361] Expand PEP 8 checks to include EC2
LogQuery.scala	SPARK-2626 [DOCS] Stop SparkContext in all examples	cassandra_outputformat.py	[SPARK-3361] Expand PEP 8 checks to include EC2
MultiBroadcastTest.scala	SPARK-1565, update examples to be used with spark-submit script.	hbase_inputformat.py	[SPARK-3361] Expand PEP 8 checks to include EC2
SimpleSkewedGroupByTest.scala	SPARK-1565, update examples to be used with spark-submit script.		
SkewedGroupByTest.scala	SPARK-1565, update examples to be used with spark-submit script.		
SparkALS.scala	ml [SPARK-5704] [SQL] [PySt... mllib [SPARK-5879] [MLLIB] upd...	hbase_outputformat.py	[SPARK-3361] Expand PEP 8 checks to include EC2
SparkKfIdfLR.scala	sql [SPARK-5704] [SQL] [PySt... streaming Revise formatting of previo...	kmeans.py	[SPARK-2850] [SPARK-2626] [mllib] MLlib stats exar...
SparkKMeans.scala	JavaHdfsLR.java [SPARK-4047] - Generate r...	logistic_regression.py	[SPARK-2850] [SPARK-2626] [mllib] MLlib stats exar...
SparkLR.scala	JavaLogQuery.java SPARK-1565, update exar...	pagerank.py	[SPARK-4047] - Generate runtime warnings for exam...
SparkPageRank.scala	JavaPageRank.java [SPARK-4047] - Generate r...	parquet_inputformat.py	SPARK-2626 [DOCS] Stop SparkContext in all exam...
SparkPi.scala	JavaSparkPi.java SPARK-2626 [DOCS] Stop ...	pi.py	[SPARK-1701] [PySpark] remove slice terminology fr...
SparkTC.scala	JavaStatusTrackerDemo.java [SPARK-2321] Several pro...	sort.py	[SPARK-2850] [SPARK-2626] [mllib] MLlib stats exar...
	JavaTC.java SPARK-1565, update exar...	sql.py	[SPARK-5704] [SQL] [PySpark] createDataFrame fro...
	JavaWordCount.java SPARK-1565, update exar...	status_api_demo.py	[SPARK-4172] [PySpark] Progress API in Python

Apache Spark

© Copyright IBM Corporation 2015

Running Spark examples

There are many example programs available that shows the various usage of Spark. Depending on your programming language preference, there are examples in all three languages that work with Spark. You can view the source code of the examples on the Spark website, or on Github, or within the Spark distribution itself. The slide is a partial overview.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

For a full list of the examples available in GitHub:

- Scala:
<https://github.com/apache/spark/tree/master/examples/src/main/scala/org/apache/spark/examples>
- Python:
<https://github.com/apache/spark/tree/master/examples/src/main/python>
- Java:
<https://github.com/apache/spark/tree/master/examples/src/main/java/org/apache/spark/examples>
- Spark Streaming:
<https://github.com/apache/spark/tree/master/examples/src/main/scala/org/apache/spark/examples/streaming>
- Java Streaming:
<https://github.com/apache/spark/tree/master/examples/src/main/java/org/apache/spark/examples/streaming>

The slide lists the steps to run some of these examples. To run Scala or Java examples, you would execute the run-example script under the Spark's bin directory. So for example, to run the SparkPi application, execute run-example SparkPi, where SparkPi would be the name of the application. Substitute that with a different application name to run that other applications.

To run the sample Python applications, use the spark-submit command and provide the path to the application.

Create Spark standalone applications - Scala

```
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

Import statements

SparkConf and
SparkContext

Transformations
+ Actions

Create Spark standalone applications: Scala

This is an example application using Scala. Similarly programs can be written in Python or Java.

The application shown here counts the number of lines with 'a' and the number of lines with 'b' - you need to replace the YOUR_SPARK_HOME with the directory where Spark is installed.

Unlike the Spark shell, you have to initialize the SparkContext in a program. First you must create a SparkConf to set up your application's name. Then you create the SparkContext by passing in the SparkConf object. Next, you create the RDD by loading in the textFile, and then caching the RDD. Since we will be apply a couple of transformation on it, caching will help speed up the process, especially if the logData RDD is large. Finally, you get the values of the RDD by executing the count action on it. End the program by printing it out onto the console.

Run standalone applications

- Define the dependencies using any system build mechanism (Ant, SBT, Maven, Gradle)
- Example:
 - Scala → simple.sbt
 - Java → pom.xml
 - Python → --py-files argument (not needed for SimpleApp.py)
- Create the typical directory structure with the files

Scala using SBT :
 ./simple.sbt
 ./src
 ./src/main
 ./src/main/scala
 ./src/main/scala/SimpleApp.scala

Java using Maven:
 ./pom.xml
 ./src
 ./src/main
 ./src/main/java
 ./src/main/java/SimpleApp.java

- Create a JAR package containing the application's code.
- Use spark-submit to run the program

Run standalone applications

At this point, you should know how to create a Spark application using any of the supported programming language. Now you get to explore how to run the application. You will need to first define the dependencies. Then you have to package the application together using system build tools such as Ant, sbt, or Maven. The examples here show how you would do it using various tools. You can use any tool for any of the programming languages. For Scala, the example is shown using sbt, so you would have a simple.sbt file. In Java, the example shows using Maven so you would have the pom.xml file. In Python, if you need to have dependencies that requires third party libraries, then you can use the -py-files argument to handle that.

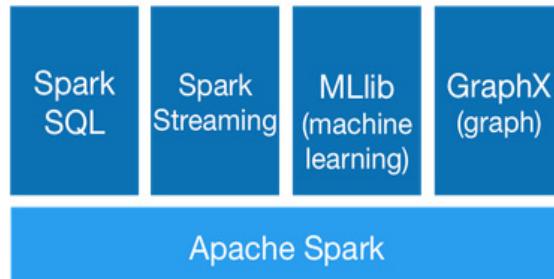
Again, shown here are examples of what a typical directory structure would look like for the tool that you choose.

Finally, once you have the JAR packaged created, run the spark-submit to execute the application:

- Scala: sbt
- Java: mvn
- Python: submit-spark

Spark libraries

- Extension of the core Spark API.
- Improvements made to the core are passed to these libraries.
- Little overhead to use with the Spark core



spark.apache.org

Spark libraries

Spark comes with libraries that you can utilize for specific use cases. These libraries are an extension of the Spark Core API. Any improvements made to the core will automatically take effect with these libraries. One of the big benefits of Spark is that there is little overhead to use these libraries with Spark as they are tightly integrated. The rest of this lesson will cover on a high level, each of these libraries and their capabilities. The main focus will be on Scala with specific callouts to Java or Python if there are major differences.

The four libraries are Spark SQL, Spark Streaming, Mllib, and GraphX. The remainder of this course will cover these libraries.

Spark SQL

- Allows relational queries expressed in
 - SQL
 - HiveQL
 - Scala
- SchemaRDD
 - Row objects
 - Schema
 - Created from:
 - Existing RDD
 - Parquet file
 - JSON dataset
 - HiveQL against Apache Hive
- Supports Scala, Java, and Python

Spark SQL

Spark SQL allows you to write relational queries that are expressed in either SQL, HiveQL, or Scala to be executed using Spark. Spark SQL has a new RDD called the SchemaRDD. The SchemaRDD consists of rows objects and a schema that describes the type of data in each column in the row. You can think of this as a table in a traditional relational database.

You create a SchemaRDD from existing RDDs, a Parquet file, a JSON dataset, or using HiveQL to query against the data stored in Hive. Spark SQL is currently an alpha component, so some APIs may change in future releases.

Spark SQL supports Scala, Java and Python.

Spark streaming

- Scalable, high-throughput, fault-tolerant stream processing of live data streams
- Receives live input data and divides into small batches which are processed and returned as batches
- DStream - sequence of RDD
- Currently supports Scala and Java
- Basic Python support available in Spark 1.2+



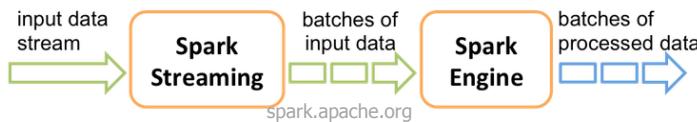
Spark streaming

Spark streaming gives you the capability to process live streaming data in small batches. Utilizing Spark's core, Spark Streaming is scalable, high-throughput and fault-tolerant. You write Stream programs with DStreams, which is a sequence of RDDs from a stream of data. There are various data sources that Spark Streaming receives from including, Kafka, Flume, HDFS, Kinesis, or Twitter. It pushes data out to HDFS, databases, or some sort of dashboard.

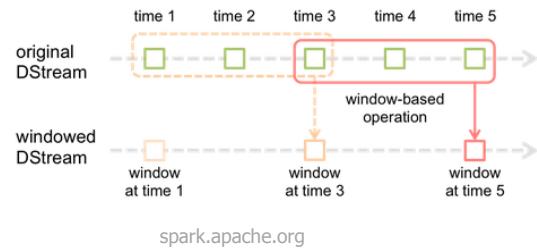
Spark Streaming supports Scala, Java and Python. Python was actually introduced with Spark 1.2. Python has all the transformations that Scala and Java have with DStreams, but it can only support text data types. Support for other sources such as Kafka and Flume will be available in future releases for Python.

Spark Streaming: Internals

- The input stream (DStream) goes into Spark Streaming
- The data is broken up into batches that are fed into the Spark engine for processing
- The final results are generated as a stream of batches



- Sliding window operations
 - Windowed computations
 - Window length
 - Sliding interval
 - `reduceByKeyAndWindow`



Spark Streaming - Internals

Here's a quick view of how Spark Streaming works. First the input stream comes in to Spark Streaming. That data stream is broken up into batches of data that are fed into the Spark engine for processing. Once the data has been processed, it is sent out in batches.

Spark Stream support sliding window operations. In a windowed computation, every time the window slides over a source of DStream, the source RDDs that falls within the window are combined and operated upon to produce the resulting RDD.

There are two parameters for a sliding window:

- The **window length** is the duration of the window
- The **sliding interval** is the interval in which the window operation is performed.

Both of these parameters must be in multiples of the batch interval of the source DStream.

In the second diagram, the window length is 3 and the sliding interval is 2. To put it into perspective, maybe you want to generate word counts over last 30 seconds of data, every 10 seconds. To do this, you would apply the reduceByKeyAndWindow operation on the pairs of DStream of (Word,1) pairs over the last 30 seconds of data.

Doing wordcount in this manner is provided as an example program

NetworkWordCount, available on GitHub at

<https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/streaming/NetworkWordCount.scala>

MLlib

- MLlib for machine learning library - under active development
- Provides, currently, the following common algorithm and utilities
 - Classification
 - Regression
 - Clustering
 - Collaborative filtering
 - Dimensionality reduction

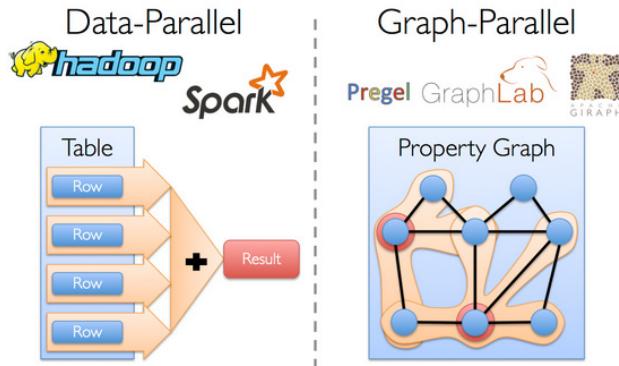
MLib

Here is a really short overview of the machine learning library. The MLlib library contains algorithms and utilities for classification, regression, clustering, collaborative filtering and dimensionality reduction. Essentially, you would use this for specific machine learning use cases that requires these algorithms.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

GraphX

- GraphX for graph processing
 - Graphs and graph parallel computation
 - Social networks and language modeling



<https://spark.apache.org/docs/latest/graphx-programming-guide.html#overview>

Apache Spark

© Copyright IBM Corporation 2015

GraphX

The GraphX is a library that sits on top of the Spark Core. It is basically a graph processing library which can be used for social networks and language modeling. Graph data and the requirement for graph parallel systems is becoming more common, which is why the GraphX library was developed. Specific scenarios would not be efficient if it is processed using the data-parallel model. A need for the graph-parallel model is introduced with new graph-parallel systems like Giraph and GraphLab to efficiently execute graph algorithms much faster than general data-parallel systems.

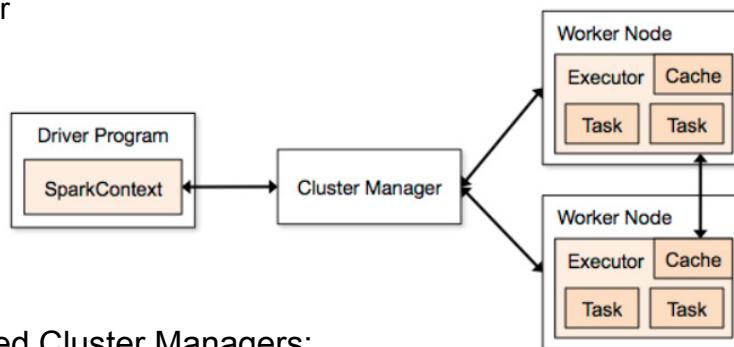
There are new inherent challenges that come with graph computations, such as constructing the graph, modifying its structure, or expressing computations that span several graphs. As such, it is often necessary to move between table and graph views depending on the objective of the application and the business requirements.

The goal of GraphX is to optimize the process by making it easier to view data both as a graph and as collections, such as RDD, without data movement or duplication.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Spark cluster overview

- Components
 - Driver
 - Cluster Master
 - Executors



- Three supported Cluster Managers:
 - Standalone
 - Apache Mesos
 - Hadoop YARN

Spark cluster overview

There are three main components of a Spark cluster. You have the driver, where the `SparkContext` is located within the main program. To run on a cluster, you would need some sort of cluster manager. This could be either Spark's standalone cluster manager, or Mesos or Yarn. Then you have your worker nodes where the executors reside. The executors are the processes that run computations and store the data for the application. The `SparkContext` sends the application, defined as JAR or Python files to each executor. Finally, it sends the tasks for each executor to run.

Several things to understand this architecture.

- Each application gets its own executor processes. The executor stays up for the entire duration that the application is running. The benefit of this is that the applications are isolated from each other, on both the scheduling side, and running on different JVMs. However, this means that you cannot share data across applications. You would need to externalize the data if you wish to share data between different applications, instances of `SparkContext`.
- Spark applications don't care about the underlying cluster manager. As long as it can acquire executors and communicate with each other, it can run on any cluster manager.
- Because the driver program schedules tasks on the cluster, it should run close to the worker nodes on the same local network. If you like to send remote requests to the cluster, it is better to use a RPC and have it submit operations from nearby.
- There are currently three supported cluster managers. Spark comes with a standalone manager that you can use to get up and running. You can use Apache Mesos, a general cluster manager that can run and service Hadoop jobs. Finally, you can also use Hadoop YARN, the resource manager in Hadoop 2. In the lab exercise, you will be using BigInsights with Yarn to run your Spark applications.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Spark monitoring

- Three ways to monitor Spark applications
 1. Web UI
 - Port 4040 (lab exercise on port 8088)
 - Available for the duration of the application
 2. Metrics
 - Based on the Coda Hale Metrics Library
 - Report to a variety of sinks (HTTP, JMX, and CSV)
/conf/metrics.properties
 3. External instrumentations
 - Ganglia
 - OS profiling tools (dstat, iostat, iotop)
 - JVM utilities (jstack, jmap, jstat, jconsole)

Spark monitoring

There are three ways to monitor Spark applications. The first way is the Web UI. The default port is 4040. The port in the lab environment is 8088. The information on this UI is available for the duration of the application. If you want to see the information after the fact, set the spark.eventLog.enabled to true before starting the application. The information will then be persisted to storage as well.

The **Web UI** has the following information.

- A list of scheduler stages and tasks.
- A summary of RDD sizes and memory usage.
- Environmental information and information about the running executors.

To view the history of an application after it has running, you can start up the history server. The history server can be configured on the amount of memory allocated for it, the various JVM options, the public address for the server, and a number of properties.

Metrics are a second way to monitor Spark applications. The metric system is based on the Coda Hale Metrics Library. You can customize it so that it reports to a variety of sinks such as CSV. You can configure the metrics system in the metrics.properties file under the conf directory.

In addition, you can also use external instrumentations to monitor Spark. Ganglia is used to view overall cluster utilization and resource bottlenecks. Various OS profiling tools and JVM utilities can also be used for monitoring Spark.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit summary

- Understand the nature and purpose of Apache Spark in the Hadoop ecosystem
- List and describe the architecture and components of the Spark unified stack
- Describe the role of a Resilient Distributed Dataset (RDD)
- Understand the principles of Spark programming
- List and describe the Spark libraries
- Launch and use Spark's Scala and Python shells

Unit summary

Additional information and exercises are available in the BigDataUniversity.com (BDU) course *Spark Fundamentals* available at <http://bigdatauniversity.com/bdu-wp/bdu-course/spark-fundamentals>. That course gets into further details on most of the topics covered here, especially Spark configuration, monitoring, and tuning.

Big Data University has been chosen by IBM as one of the issuers of badges as part of the IBM Open Badge program.

Big Data University leverages the services of Pearson VUE Acclaim to assist in the administration of the IBM Open Badge program. By enrolling into this course, you agree to Big Data University sharing your details with Pearson VUE Acclaim for the strict use of issuing your badge upon completion of the badge criteria. The BDU course comes with a final quiz where the minimum passing mark for the course is 60%, where the final test is worth 100% of the course mark.

The BDU course uses a IBM Quickstart Edition (QSE) v4 environment.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1

Working with a Spark RDD with Scala

Exercise 1: Working with a Spark RDD with Scala

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1: Working with a Spark RDD with Scala

Purpose:

In this Exercise you will learn to use some of the fundamental aspects of running Spark in the Open Data Platform environment.

Additional information on Spark and additional lab exercises using Spark (with Scala, Python, and Java) are available in the BigDataUniversity.com (BDU) course *Spark Fundamentals* available at <http://bigdatauniversity.com/bdu-wp/bdu-course/spark-fundamentals>.

Good locations for additional information on Scala and introductory exercises in Spark with Scala and Python can be found at:

Scala Community: <http://www.scala-lang.org>

Quick Start: <https://spark.apache.org/docs/latest/quick-start.html>

Blogs, e.g.: <http://blog.ajduke.in/2013/05/31/various-ways-to-run-scala-code>

Note:

You may need verify that your hostname and IP address are setup correctly as noted in earlier units. Note that if you shut down your lab environment, this verification of hostname and IP address should be repeated. This is particularly important if you find that you cannot connect to Spark. Resetting the IP values may require that you reboot the VMware Image (as root: `reboot`).

Task 1. Connect to the VMware Image & to the Spark server.

1. Connect to and login to your lab environment with user **biadmin** and password **biadmin** credentials.
2. In a new terminal window, type `cd` to change to your home directory.
3. To set an environmental variable `$SPARK_HOME`, type `export SPARK_HOME=/usr/iop/current/spark-client`.

4. To start the Spark shell, type `$SPARK_HOME/bin/spark-shell`.

```
[biadmin@ibmclass Desktop]$ cd  
[biadmin@ibmclass ~]$ $SPARK_HOME/bin/spark-shell  
15/06/05 11:06:04 INFO spark.SecurityManager: Changing view acls to: biadmin  
15/06/05 11:06:04 INFO spark.SecurityManager: Changing modify acls to: biadmin  
15/06/05 11:06:04 INFO spark.SecurityManager: SecurityManager: authentication  
disabled; ui acls disabled; users with view permissions: Set(biadmin); users with  
modify permissions: Set(biadmin)  
15/06/05 11:06:04 INFO spark.HttpServer: Starting HTTP Server  
15/06/05 11:06:04 INFO server.Server: jetty-8.y.z-SNAPSHOT  
15/06/05 11:06:04 INFO server.AbstractConnector: Started  
SocketConnector@0.0.0.0:34520  
15/06/05 11:06:04 INFO util.Utils: Successfully started service 'HTTP class server'  
on port 34520.  
Welcome to  
  
version 1.2.1  
  
Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_65)  
Type in expressions to have them evaluated.  
Type :help for more information.  
15/06/05 11:06:07 INFO spark.SecurityManager: Changing view acls to: biadmin  
15/06/05 11:06:07 INFO spark.SecurityManager: Changing modify acls to: biadmin  
...  
15/06/05 11:06:10 INFO scheduler.EventLoggingListener: Logging events to  
hdfs://ibmclass.localdomain:8020/iop/apps/4.0.0.0/spark/logs/history-server/local-  
1433520369438  
15/06/05 11:06:10 INFO repl.SparkILoop: Created spark context..  
Spark context available as sc.  
  
scala>
```

You now have a Scala prompt where you can enter Scala interactively.

Note that the Spark context is available as sc.

To exit from Scala at any time, you type `sys.exit` and press **Enter** (the official approach) or use Ctrl-D.

The tab key provides code completion.

5. Type `sc.` (the period is needed!), and then press **Tab**.

```
scala> sc.
accumulable          accumulableCollection
accumulator          addFile
addJar               addSparkListener
appName              applicationId
asInstanceOf         binaryFiles
binaryRecords        broadcast
cancelAllJobs        cancelJobGroup
clearCallSite        clearFiles
clearJars            clearJobGroup
defaultMinPartitions defaultMinSplits
defaultParallelism   emptyRDD
files                getAllPools
getCheckpointDir    getConf
getExecutorMemoryStatus getExecutorStorageStatus
getLocalProperty     getPersistentRDDs
getPoolForName       getRDDStorageInfo
getSchedulingMode   hadoopConfiguration
hadoopFile           hadoopRDD
initLocalProperties  isInstanceOf
isLocal              jars
killExecutor         killExecutors
makeRDD              master
metricsSystem         newAPIHadoopFile
newAPIHadoopRDD      objectFile
parallelize          requestExecutors
runApproximateJob   runJob
sequenceFile         setCallSite
setCheckpointDir    setJobDescription
setJobGroup          setLocalProperty
sparkUser            startTime
statusTracker        stop
submitJob            tachyonFolderName
textFile              toString
union                version
wholeTextFiles
```

Note, if you type `sc` and press **Tab**, without the period after `sc`, you will get an abbreviated output, since only three keywords start with `sc`, whereas a lot of functionality is provided by the Spark context ("sc.").

```
scala> sc
sc      scala    schema
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Task 2. Load data into an RDD and perform transformations and actions on that data.

- To do an RDD transformation by reading in a file that was previously loaded to HDFS, type the following:

```
val pp = sc.textFile("Gutenberg/Pride_and_Prejudice.txt")

scala> val pp = sc.textFile("Gutenberg/Pride_and_Prejudice.txt")
15/06/05 12:05:48 INFO storage.MemoryStore: ensureFreeSpace(274073) called with
curMem=0, maxMem=278302556
15/06/05 12:05:48 INFO storage.MemoryStore: Block broadcast_0 stored as values in
memory (estimated size 267.6 KB, free 265.1 MB)
15/06/05 12:05:49 INFO storage.MemoryStore: ensureFreeSpace(41821) called with
curMem=274073, maxMem=278302556
15/06/05 12:05:49 INFO storage.MemoryStore: Block broadcast_0_piece0 stored as bytes
in memory (estimated size 40.8 KB, free 265.1 MB)
15/06/05 12:05:49 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory
on localhost:46250 (size: 40.8 KB, free: 265.4 MB)
15/06/05 12:05:49 INFO storage.BlockManagerMaster: Updated info of block
broadcast_0_piece0
15/06/05 12:05:49 INFO spark.SparkContext: Created broadcast 0 from textFile at
<console>:12
pp: org.apache.spark.rdd.RDD[String] = Gutenberg/Pride_and_Prejudice.txt MappedRDD[1]
at textFile at <console>:12
```

The result is a pointer to the file. The file is not actually read at this time, as is evidenced by noting that you do not get any errors if you misspell the file name. Now **pp** is a pointer to the RDD.

We can perform some RDD actions on this data. One simple action is to count the number of items (lines, records) in the RDD.

- To count the number of items in the RDD, type **pp.count()**.

```
scala> pp.count()
15/06/05 12:02:27 INFO mapred.FileInputFormat: Total input paths to process : 1
15/06/05 12:02:27 INFO spark.SparkContext: Starting job: count at <console>:15
15/06/05 12:02:27 INFO scheduler.DAGScheduler: Got job 0 (count at <console>:15) with
2 output partitions (allowLocal=false)
15/06/05 12:02:27 INFO scheduler.DAGScheduler: Final stage: Stage 0(count at
<console>:15)
15/06/05 12:02:27 INFO scheduler.DAGScheduler: Parents of final stage: List()
15/06/05 12:02:27 INFO scheduler.DAGScheduler: Missing parents: List()
15/06/05 12:02:27 INFO scheduler.DAGScheduler: Submitting Stage 0
(Gutenberg/Pride_and_Prejudice.txt MappedRDD[7] at textFile at <console>:12), which
has no missing parents
15/06/05 12:02:27 INFO storage.MemoryStore: ensureFreeSpace(2536) called with
curMem=1263720, maxMem=278302556

...
15/06/05 12:02:28 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks
have all completed, from pool
15/06/05 12:02:28 INFO scheduler.DAGScheduler: Stage 0 (count at <console>:15)
finished in 0.600 s
res2: Long = 13030

scala> 15/06/05 12:02:28 INFO scheduler.DAGScheduler: Job 0 finished: count at
<console>:15, took 0.958171 s
```

The number of lines in the file is 13030.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

3. In a new terminal window, to verify the number of lines in the file, use the Linux command `wc` on the original file that we uploaded to HDFS:

```
wc -l /home/biadmin/labfiles/Pr*
```

```
[biadmin@ibmclass ~]$ wc -l /home/labfiles/Pr*
13030 /home/labfiles/Pride_and_Prejudice.txt
[biadmin@ibmclass ~]$
```

4. Restart the Spark Shell, and then to read the first record of the RDD, type `pp.first()`.

```
scala> pp.first()
15/06/05 12:40:14 INFO mapred.FileInputFormat: Total input paths to process : 1
15/06/05 12:40:14 INFO spark.SparkContext: Starting job: first at <console>:15
15/06/05 12:40:14 INFO scheduler.DAGScheduler: Got job 1 (first at <console>:15) with
1 output partitions (allowLocal=true)
15/06/05 12:40:14 INFO scheduler.DAGScheduler: Final stage: Stage 1(first at
<console>:15)
15/06/05 12:40:14 INFO scheduler.DAGScheduler: Parents of final stage: List()
15/06/05 12:40:14 INFO scheduler.DAGScheduler: Missing parents: List()
15/06/05 12:40:14 INFO scheduler.DAGScheduler: Submitting Stage 1
(Gutenberg/Pride_and_Prejudice.txt MappedRDD[3] at textFile at <console>:12), which
has no missing parents
15/06/05 12:40:14 INFO storage.MemoryStore: ensureFreeSpace(2560) called with
curMem=631860, maxMem=278302556
15/06/05 12:40:14 INFO storage.MemoryStore: Block broadcast_3 stored as values in
memory (estimated size 2.5 KB, free 264.8 MB)
15/06/05 12:40:14 INFO storage.MemoryStore: ensureFreeSpace(1901) called with
curMem=634420, maxMem=278302556
15/06/05 12:40:14 INFO storage.MemoryStore: Block broadcast_3_piece0 stored as bytes
in memory (estimated size 1901.0 B, free 264.8 MB)
15/06/05 12:40:14 INFO storage.BlockManagerInfo: Added broadcast_3_piece0 in memory
on localhost:46250 (size: 1901.0 B, free: 265.3 MB)
15/06/05 12:40:14 INFO storage.BlockManagerMaster: Updated info of block
broadcast_3_piece0
15/06/05 12:40:14 INFO spark.SparkContext: Created broadcast 3 from broadcast at
DAGScheduler.scala:838
15/06/05 12:40:14 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from Stage
1 (Gutenberg/Pride_and_Prejudice.txt MappedRDD[3] at textFile at <console>:12)
15/06/05 12:40:14 INFO scheduler.TaskSchedulerImpl: Adding task set 1.0 with 1 tasks
15/06/05 12:40:14 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 1.0 (TID
1, localhost, ANY, 1343 bytes)
15/06/05 12:40:14 INFO executor.Executor: Running task 0.0 in stage 1.0 (TID 1)
15/06/05 12:40:14 INFO rdd.HadoopRDD: Input split:
hdfs://ibmclass.localdomain:8020/user/biadmin/Gutenberg/Pride_and_Prejudice.txt:0+348
901
15/06/05 12:40:14 INFO mapred.LineRecordReader: Found UTF-8 BOM and skipped it
15/06/05 12:40:14 INFO executor.Executor: Finished task 0.0 in stage 1.0 (TID 1).
1796 bytes result sent to driver
15/06/05 12:40:14 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0 (TID
1) in 21 ms on localhost (1/1)
15/06/05 12:40:14 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks
have all completed, from pool
15/06/05 12:40:14 INFO scheduler.DAGScheduler: Stage 1 (first at <console>:15)
finished in 0.021 s
15/06/05 12:40:14 INFO scheduler.DAGScheduler: Job 1 finished: first at <console>:15,
took 0.030293 s
res2: String = PRIDE AND PREJUDICE

scala>
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

The first actual line in the file has the string: *PRIDE AND PREJUDICE*. This string is the title of the book.

Scala, Python, and Java are each substantive languages. It is not our goal to teach you the complete Scala language in this unit, but merely to introduce you to it.

Scala is an interpreted language, like Java, and has a compiler scalac just as Java has its compiler javac.

The next stage in your learning should be to take the free BigDataUniversity (BDU) course on Spark, which has programming exercises in Scala and Python that carry on from what you have learned here. The BDU course uses a free, downloadable VMware Image based on BigInsights v4 and Open Data Platform software. From there you would progress to one of the textbooks on learning Scala, but be warned that the best ones are often over 500 pages long.

Close all open windows.

Results:

You learned to use some of the fundamental aspects of running Spark in the Open Data Platform environment.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit 11 Coordination, Management, and Governance

IBM Training



Coordination, Management, and Governance

IBM BigInsights v4.0

© Copyright IBM Corporation 2015
Course materials may not be reproduced in whole or in part without the written permission of IBM.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit objectives

- Understand the challenges posed by distributed applications and how ZooKeeper is designed to handle them
- Explain the role of ZooKeeper within the Apache Hadoop infrastructure and the realm of Big Data management
- Explore generic use cases and some real-world scenarios for ZooKeeper
- Define the ZooKeeper services that are used to manage distributed systems
- Explore and use the ZooKeeper CLI to interact with ZooKeeper services
- Understand how Apache Slider works in conjunction with YARN to deploy distributed applications and to monitor them
- Explain how Apache Knox provides peripheral security services to an Hadoop cluster

Unit objectives

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Topic: ZooKeeper

Coordination, Management, and Governance

© Copyright IBM Corporation 2015

Topic: ZooKeeper

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

ZooKeeper

- ZooKeeper provides support for writing distributed applications in the Hadoop ecosystem
- ZooKeeper addresses the issue of partial failure
 - Partial failure is intrinsic to distributed systems
 - ZK provides a set of tools to build distributed applications that can safely handle partial failures
- Zookeeper has the following characteristics:
 - simple
 - expressive
 - highly available
 - facilitates loosely coupled interactions
 - is a library
- Apache ZooKeeper is an open source server that enables highly reliable distributed coordination



ZooKeeper

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

Information on Zookeeper can be found at:

- **Primary website:** <http://zookeeper.apache.org>
- **Wiki:** <https://cwiki.apache.org/confluence/display/ZOOKEEPER/Index>
- **Documentation:** <http://zookeeper.apache.org/doc/trunk>
- **FAQ:** <https://cwiki.apache.org/confluence/display/ZOOKEEPER/FAQ>
- Chapter 21 (pp. 601-40) in: White, T. (2015). *Hadoop: The definitive guide* (4th ed.). Sebastopol, CA: O'Reilly Media.

The standard documentation includes the following sections:

- ZooKeeper Overview
- Developers
- Administrators & Operators
- Contributors to Zookeeper
- Miscellaneous FAQ

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Distributed systems

- Multiple software components on multiple computers, but run as a single system
- Computers can be physically close (local network), or geographically distant (WAN)
- The goal of distributed computing is to make such a network work as a single computer
- Distributed systems offer many benefits over centralized systems
 - **Scalability:** System can easily be expanded by adding more machines as needed
 - **Redundancy:** Several machines can provide the same services, so if one is unavailable: work does not stop, smaller machines can be used, redundancy not prohibitively expensive

Distributed systems

A distributed computer system consists of multiple software components that are on multiple computers, but run as a single system. The computers that are in a distributed system can be physically close together and connected by a local network, or they can be geographically distant and connected by a wide area network. The goal of distributed computing is to make such a network work as a single computer.

Distributed systems offer many benefits over centralized systems. One benefit is that of Scalability. Systems can easily be expanded by adding more machines as needed. Another major benefit of distributed systems is Redundancy. Several machines can provide the same services, so if one is unavailable, work does not stop. Additionally, because many smaller machines can be used, this redundancy does not need to be prohibitively expensive.

What is ZooKeeper?

- Distributed applications require coordination
 - Develop your own service (a lot of work)
 - Or, use robust pre-existing coordination services (ZooKeeper)
- Distributed open-source centralized coordination service for:
 - Maintaining configuration information: Sharing config info across all "nodes"
 - Naming: Find a machine in a cluster of 1,000s of servers, Naming Service
 - Providing distributed synchronization: Locks, Barriers, Queues, etc.
 - Providing group services: Leader election and more
- The ZooKeeper service is distributed, reliable, fast, and simple

What is ZooKeeper?

Distributed applications require coordination. Coordination services are notoriously hard to get right. They are especially prone to errors such as race conditions and deadlock. The motivation behind ZooKeeper is to relieve distributed applications the responsibility of implementing coordination services from scratch.

You could develop your own coordination service, however that can take a lot of work and is not a trivial task. The alternative is using a robust coordination service that is already available for use. ZooKeeper is just that - a distributed centralized coordination service that is open-source and ready for use.

ZooKeeper is a distributed, open-source coordination service for distributed applications. It exposes a simple set of primitives that distributed applications can build upon to implement higher level services for synchronization, configuration maintenance, and groups and naming. It is designed to be easy to program to, and uses a data model styled after the familiar directory tree structure of file systems. It runs in Java and has bindings for both Java and C.

ZooKeeper is simple. ZooKeeper allows distributed processes to coordinate with each other through a shared hierachal namespace which is organized similarly to a standard file system. The name space consists of data registers, called znodes, in ZooKeeper parlance, and these are similar to files and directories. Unlike a typical file system, which is designed for storage, ZooKeeper data is kept in-memory, which means ZooKeeper can achieve high throughput and low latency numbers.

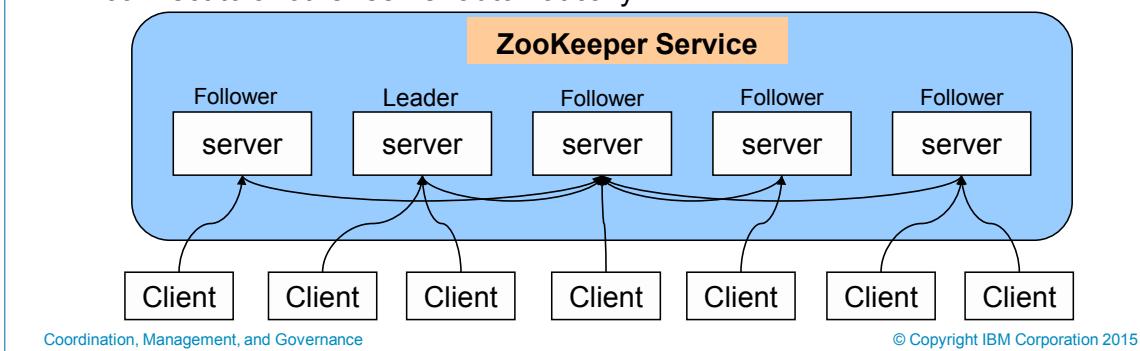
The ZooKeeper implementation puts a premium on high performance, highly available, strictly ordered access. The performance aspects of ZooKeeper means it can be used in large, distributed systems. The reliability aspects keep it from being a single point of failure. The strict ordering means that sophisticated synchronization primitives can be implemented at the client. The ZooKeeper service can be used to help you tackle many of the common challenges distributed applications face.

- ZooKeeper can be used to maintain configuration information. For example you can store configuration data in ZooKeeper and share that data across all nodes in your distributed setup.
- ZooKeeper can be used for naming. An example is using it as a naming service, allowing one node to find a specific machine in a cluster of thousands of servers.
- ZooKeeper can be used to solve the problem of distributed synchronization, providing the building blocks for Locks, Barriers, and Queues.
- ZooKeeper can also be used for group services such as leader election and more.
- ZooKeeper provides the building blocks for all of these scenarios and is distributed, reliable and fast, while still being relatively simple to work with!

Reference: <http://zookeeper.apache.org/doc/trunk/zookeeperOver.html>

ZooKeeper service: replicated mode

- ZooKeeper runs on cluster of machines, Ensemble, providing high availability and consistency
 - Requires majority of servers; 7 node ensemble can lose 3 nodes
 - Writes go through leader; requires majority consensus
- Servers that make up the ZooKeeper service know about each other
 - Maintain an in-memory image of state
 - Clients connect to only one server, but If they loose the connection, can connect to another server automatically



ZooKeeper service: replicated mode

In a distributed ZooKeeper implementation, there are multiple servers. This is known as ZooKeeper's Replicated Mode. One server is elected as the leader and all additional servers are followers. If the ZooKeeper leader fails, then a new leader is elected.

All ZooKeeper servers must know about each other. Each server maintains an in-memory image of the overall state as well as transaction logs and snapshots in persistent storage. Clients connect to just a single server, however, when a client is started, it can provide a list of servers. In that way, if the connection to server for that client fails, the client connects to the next server in its list. Since each server maintains the same information, the client is able to continue to function without interruption.

A ZooKeeper client can perform a read operation from any server in the ensemble, however a write operation must go through the ZooKeeper leader and requires a majority consensus to succeed.

Who does what? Reads are satisfied by followers, writes are committed by the leader.

The processing sequence is:

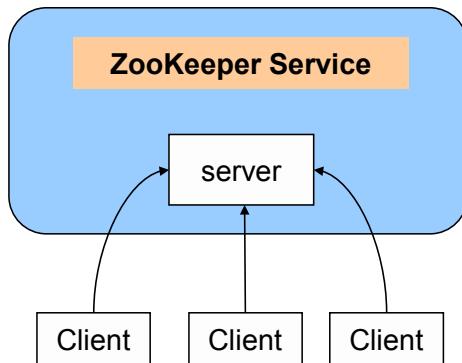
- ZooKeeper Service is replicated over a set of machines
- All machines store a copy of the data (in memory)
- A leader is elected on service startup
- Clients only connect to a single ZooKeeper server & maintains a TCP connection.
- Client can read from any Zookeeper server, writes go through the leader & and this needs a majority consensus.

The image in the slide above is based on:

<https://cwiki.apache.org/confluence/display/ZOOKEEPER/ProjectDescription>

ZooKeeper service: Standalone mode

- Single ZooKeeper server
- Good for testing/learning
- Lacks benefits of Replicated Mode; no guarantee of high-availability or resilience



ZooKeeper service: Standalone mode

ZooKeeper can also be run in Standalone mode. In this mode only a single ZooKeeper server exists. All clients connect to the ZooKeeper service via this one server. You lose the benefits of replicated mode when using standalone mode. Trading high-availability and resilience for a simpler environment can be useful for testing and learning purposes.

Consistency guarantees

- Sequential consistency
 - Client updates are applied in order they are received/sent
- Atomicity
 - Updates succeed or fail; partial updates are not allowed
- Single system image
 - Client sees same view of ZooKeeper service regardless of server
- Reliability
 - If update succeeds then it persists
- Timeliness
 - Client view of system is guaranteed up-to-date within a time bound
 - Generally within tens of seconds
 - If client does not see system changes within time bound, then service-outage

Consistency guarantees

ZooKeeper provides five consistency guarantees:

1. Sequential Consistency: updates from a client to the ZooKeeper service are applied in the order they are sent.
2. Atomicity: updates in ZooKeeper either succeed or fail. Partial updates are not allowed.
3. Single System Image: a client will see the same view of the ZooKeeper service regardless of the server in the ensemble that it is connected to.
4. Reliability: if an update succeeds in ZooKeeper then it will persist and not be rolled back. The update will only be overwritten when another client performs a new update.
5. Timeliness: a client's view of the system is guaranteed to be up-to-date within a certain time bound, generally within tens of seconds. If a client does not see system changes within that time bound, then the client assumes a service outage and will connect to a different server in the ensemble.

What ZooKeeper does not guarantee

- Simultaneously consistent cross-client views
- Different clients will not always have identical views of ZooKeeper data at every instance in time.
 - ZooKeeper provides the sync() method
 - Forces a ZooKeeper ensemble server to catch up with leader

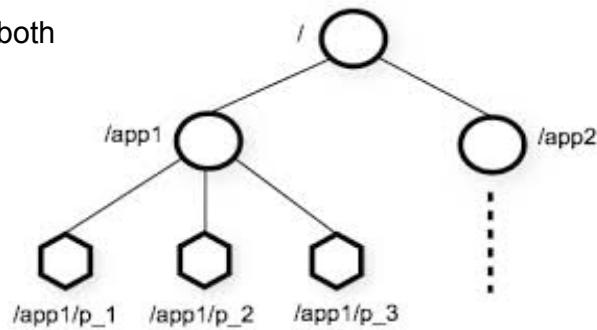
What ZooKeeper does not guarantee

In the previous slide we looked at the consistency guarantees that ZooKeeper makes. It is important to understand that ZooKeeper does *not* make a *Simultaneously Consistent Cross-Client View* guarantee. This means that ZooKeeper does not guarantee that different clients will have identical views of ZooKeeper data at every instance in time.

Network delays and other factors may make it possible for one client to perform an update before another client is notified of the change. The way you can handle this is using the sync() method that ZooKeeper provides. The sync method forces a ZooKeeper ensemble server to catch up with the leader.

ZooKeeper structure: data model

- Distributed processes coordinate through shared hierachal namespaces
 - These are organized very similarly to standard UNIX and Linux file systems
- A namespace consists of data registers
 - Called ZNodes
 - Similar to files and directories
 - ZNode holds data, children, or both
- ZNode types
 - Persistent: lasts until deleted
 - Ephemeral: lasts for the duration of the session, cannot have children
 - Sequence: provides unique numbering



Source : <http://zookeeper.apache.org>

ZooKeeper structure: data model

The distributed processes using ZooKeeper coordinate with each other through shared hierarchical namespaces. These namespaces are organized similarly to file systems in UNIX or Linux.

Each namespace has a root node and can have one or more child nodes. Since the term node has so many different connotations, ZooKeeper refers to each of these nodes as ZNodes.

As previously stated, data can be stored in a ZNode.

When data is written to or read from a ZNode, all of the data is either written or read. Also, there is an access controlled list (also known as an ACL) that is associated with each ZNode. This allows control over who can create, read, update, and delete a Znode.

ZNode operations

- ZNodes are the main entity that a programmer or administrator uses
- Access is also available through a ZooKeeper Shell

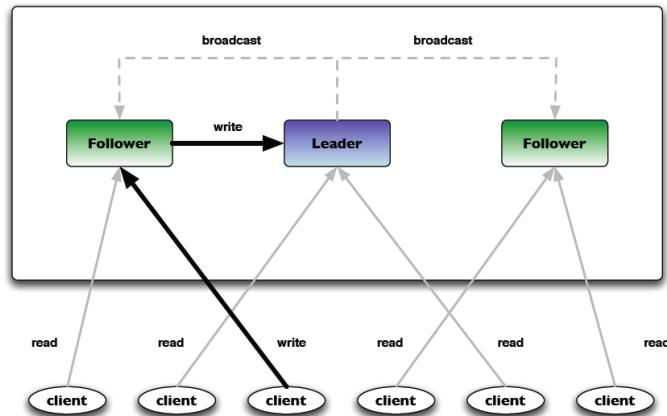
Operation	Type
create	Write
delete	Write
exists	Read
getChildren	Read
getData	Read
setData	Write
getACL	Read
setACL	Write
sync	Read

ZNode watches

- Clients can set watches on ZNodes:
 - NodeChildrenChanged
 - NodeCreated
 - NodeDataChanged
 - NodeDeleted
- Changes to a ZNode trigger the watch and ZooKeeper sends the client a notification.
- Watches are one-time triggers.
- Watches are always ordered.
- Client sees watched event before new ZNode data.
- Client should handle cases of latency between getting the event and sending a new request to get a watch

ZNode reads and writes

- Read requests are processed locally at the ZooKeeper server to which the client is currently connected
- Write requests are forwarded to the leader and go through majority consensus before a response is generated



ZooKeeper generic use cases

- Configuration Management
 - Cluster member nodes bootstrapping configuration from a centralized source in unattended way
 - Easier, simpler deployment/provisioning
- Distributed Cluster Management
 - Node join/leave
 - Node statuses in real time
- Naming service such as DNS
- Distributed synchronization: locks, barriers, queues
- Leader election in a distributed system
- Centralized and highly reliable (simple) data registry

ZooKeeper's role in the Hadoop infrastructure

- HBase
 - Uses ZooKeeper for master election, server lease management, bootstrapping, and coordination between servers
- Hadoop and MapReduce
 - Uses ZooKeeper to aid in high availability of Resource Manager
- Flume
 - Uses ZooKeeper for configuration purposes in recent releases

ZooKeeper's role in the Hadoop infrastructure

As new versions of Hadoop are released, ZooKeeper is being used more and more in the Hadoop infrastructure. Some of the uses are:

- HBase uses ZooKeeper for master election, server lease management, bootstrapping, and coordination between servers.
- Later versions of Hadoop are using ZooKeeper to provide high availability for YARN's Resource Manager.
- Apache Flume has been using ZooKeeper for configuration purposes in recent releases.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Real world use cases for ZooKeeper

- Rackspace
 - "The Email and Apps team uses ZooKeeper to coordinate sharding and responsibility changes in a distributed e-mail client that pulls and indexes data for search. ZooKeeper also provides distributed locking for connections to prevent a cluster from overwhelming servers."
- Twitter
 - Service Discovery
- Vast.com
 - "Used internally as a part of sharding services, distributed synchronization of data/index updates, configuration management and failover support"
- Yahoo
 - "ZooKeeper is used for a myriad of services inside Yahoo! for doing leader election, configuration management, sharding, locking, group membership etc."
- Zynga
 - "Used for a variety of services including configuration management, leader election, sharding and more ..."

Real world use cases for ZooKeeper

A variety of companies are using ZooKeeper with their distributed applications. Twitter, Yahoo and many others are using ZooKeeper for different purposes such as configuration management, sharding, locking and more.

Sources: *Apache ZooKeeper wiki and blog.twitter.com*

Topic: Slider

Coordination, Management, and Governance

© Copyright IBM Corporation 2015

Topic 2: Slider

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Slider

- Apache Slider is a YARN application to deploy existing distributed applications on YARN, monitor them and make them larger or smaller as desired, even while the application is running.
- Some of the features are:
 - Allows users to create on-demand applications in a YARN cluster
 - Allow different users/applications to run different versions of the application.
 - Allow users to configure different application instances differently
 - Stop/restart application instances as needed
 - Expand/shrink application instances as needed
- The Slider tool is a Java command line application.

Development model: Apache incubator project

License: Apache

Adoption: HDP, Pivotal, BigInsights

Contributors: 20+

Community activity: Vibrant (80+ issues/month)

Established: 2014

Key backers: Hortonworks

URL: slider.incubator.apache.org

Slider

Apache Slider is a YARN application to deploy existing distributed applications on YARN, monitor them and make them larger or smaller as desired, even while the application is running.

Applications can be stopped then started; the distribution of the deployed application across the YARN cluster is persisted -enabling a best-effort placement close to the previous locations. Applications which remember the previous placement of data (such as HBase) can exhibit fast start-up times from this feature.

YARN itself monitors the health of "YARN containers" hosting parts of the deployed application -it notifies the Slider manager application of container failure. Slider then asks YARN for a new container, into which Slider deploys a replacement for the failed component. As a result, Slider can keep the size of managed applications consistent with the specified configuration, even in the face of failures of servers in the cluster, as well as parts of the application itself.

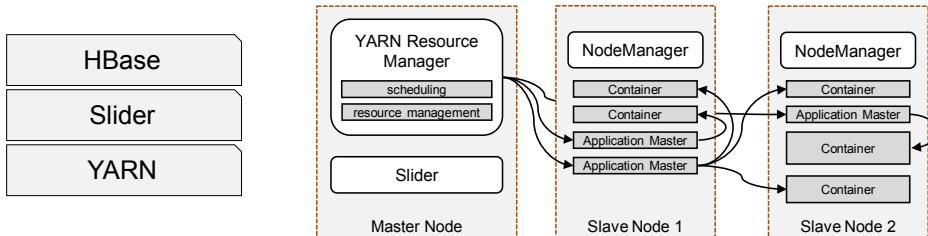
Apache Slider (incubating) is an effort undergoing incubation at The Apache Software Foundation (ASF), sponsored by the name of Apache TLP sponsor. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Apache Slider: enable long running services on YARN

- YARN resource management and scheduling works well for batch workloads, but not for interactive or real-time data processing services
- Apache Slider extends YARN to support long-running distributed services on an Hadoop cluster
 - Dynamic expansion or contraction of services
 - Dedicated monitoring
 - Supports restart after process failure
- Live Long and Process (LLAP)

**Live Long
And Process!**



Role of Slider in the Hadoop ecosystem (1 of 2)

- Applications can be stopped then started
 - The distribution of the deployed application across the YARN cluster is persisted
 - This enables best-effort placement close to the previous locations
 - Applications which remember the previous placement of data (such as HBase) can exhibit fast start-up times from this feature.
- YARN itself monitors the health of "YARN containers" hosting parts of the deployed application
 - YARN notifies the Slider manager application of container failure
 - Slider then asks YARN for a new container, into which Slider deploys a replacement for the failed component, keeping the size of managed applications consistent with the specified configuration
- The tool persists the information as JSON documents in HDFS.

Role of Slider in the Hadoop ecosystem

Applications can be stopped then started; the distribution of the deployed application across the YARN cluster is persisted -enabling a best-effort placement close to the previous locations. Applications which remember the previous placement of data (such as HBase) can exhibit fast start-up times from this feature.

YARN itself monitors the health of "YARN containers" hosting parts of the deployed application -it notifies the Slider manager application of container failure. Slider then asks YARN for a new container, into which Slider deploys a replacement for the failed component. As a result, Slider can keep the size of managed applications consistent with the specified configuration, even in the face of failures of servers in the cluster, as well as parts of the application itself.

Role of Slider in the Hadoop ecosystem (2 of 2)

- Once the cluster has been started:
 - The cluster can be made to grow or shrink using Slider commands
 - The cluster can also be stopped and later restarted
- Slider implements all its functionality through YARN APIs and the existing application shell scripts
- The goal of the application was to have minimal code changes and impact on existing applications

Some of the features of Slider are:

- Allows users to create on-demand applications in a YARN cluster
- Allow different users/applications to run different versions of the application.
- Allow users to configure different application instances differently
- Stop / Restart application instances as needed
- Expand / shrink application instances as needed

The Slider tool is a Java command-line application.

Once the cluster has been started, the cluster can be made to grow or shrink using the Slider commands. The cluster can also be stopped and later restarted.

Slider implements all its functionality through YARN APIs and the existing application shell scripts. The goal of the application was to have minimal code changes and as of this writing, it has required few changes

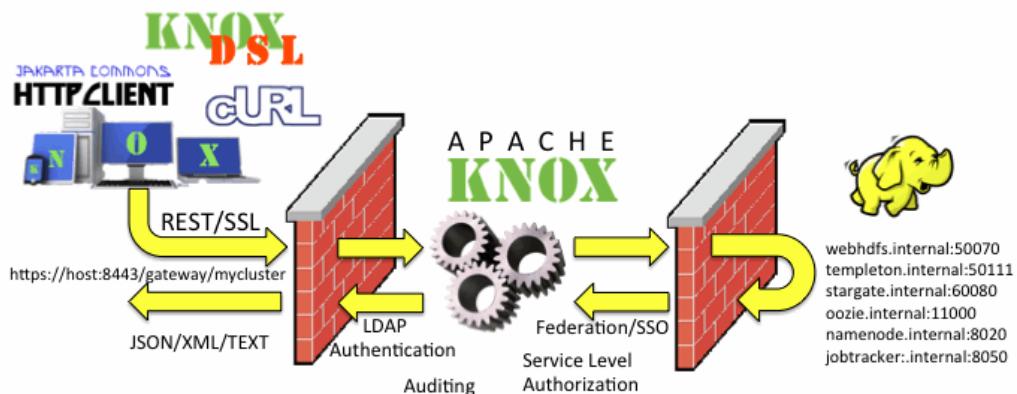
Topic: Knox

Topic: Knox

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Knox

- The Apache Knox Gateway is an extensible reverse proxy framework for securely exposing REST APIs and HTTP based services at a perimeter
- Different types of REST access supported: HTTP(S) client, cURL, Knox Shell (DSL), SSL, ...



Coordination, Management, and Governance

© Copyright IBM Corporation 2015

Knox

The Apache Knox Gateway is a REST API Gateway for interacting with Hadoop clusters. The Knox Gateway provides a single access point for all REST interactions with Hadoop clusters. In this capacity, the Knox Gateway is able to provide valuable functionality to aid in the control, integration, monitoring and automation of critical administrative and analytical needs of the enterprise.

- Authentication (LDAP and Active Directory Authentication Provider)
- Federation/SSO (HTTP Header Based Identity Federation)
- Authorization (Service Level Authorization)
- Auditing

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

While there are a number of benefits for unsecured Hadoop clusters, the Knox Gateway also complements the kerberos secured cluster quite nicely. Coupled with proper network isolation of a Kerberos secured Hadoop cluster, the Knox Gateway provides the enterprise with a solution that:

- Integrates well with enterprise identity management solutions
- Protects the details of the Hadoop cluster deployment (hosts and ports are hidden from endusers)
- Simplifies the number of services that clients need to interact with

Reference: <https://knox.apache.org> (including the diagram above).

Knox runs in a firewall DMZ between the external clients and the Apache Hadoop components.

Additional information:

- **cURL** is a computer software project providing a library and command line tool for transferring data using various protocols. The cURL project produces two products, **libcurl** and **cURL**. The name is a recursive acronym that stands for Curl URL Request Library.
- The Knox Wiki (<https://cwiki.apache.org/confluence/display/KNOX/Examples>) provides examples of the use of the Knox Shell DSL including an example that submits the familiar WordCount Java MapReduce job to the Hadoop cluster via the gateway using the Knox Shell DSL. However, in this case, the job is submitted via an Oozie workflow, showing several different ways of doing this.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Knox security

- Knox provides perimeter security for Hadoop clusters with the following advantages:

Advantage	Description
Simplified access	Extends Hadoop's REST/HTTP services by encapsulating Kerberos within the cluster
Enhanced security	Exposes Hadoop's REST/HTTP services without revealing network details, with SSL provided out of box
Centralized control	Centrally enforces REST API security and route requests to multiple Hadoop clusters
Enterprise integration	Supports LDAP, Active Directory, SSO, SAML, and other authentication systems

- Any fully secure Hadoop cluster needs Kerberos, but Kerberos requires a client-side library and complex client side configuration
- By encapsulating Kerberos, Knox eliminates the need for client software and client configuration thereby simplifying the access model

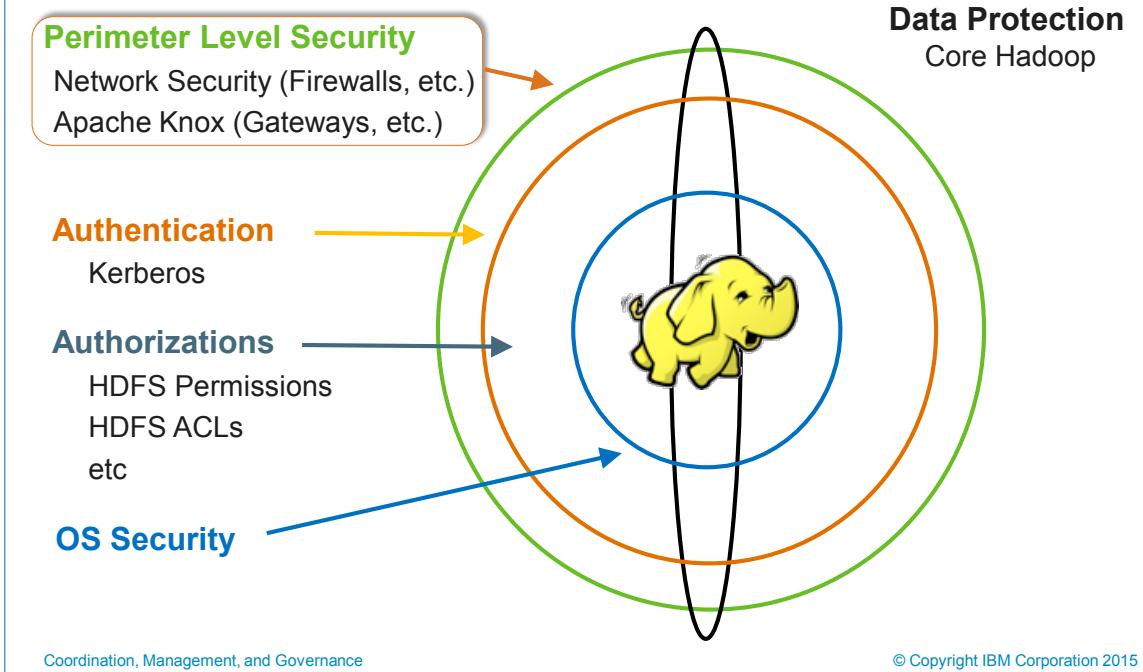
Knox security

REST (Representational State Transfer) is an architectural style for networked hypermedia applications. It is primarily used to build Web services that are lightweight, maintainable, and scalable. A service based on REST is called a RESTful service. REST is not dependent on any protocol, but almost every RESTful service uses HTTP as its underlying protocol.

The client and service talk to each other via messages. Clients send a request to the server, and the server replies with a response. Apart from the actual data, these messages also contain some metadata about the message.

Reference: <http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069>

Apache Knox is just one ring of an overall security system



Apache Knox is just one ring of an overall security system

Apache Knox provides perimeter security.

Other levels of security, including Access Control Lists (ACLs) and file level protections are needed at other levels within the Hadoop cluster.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit summary

- Understand the challenges posed by distributed applications and how ZooKeeper is designed to handle them
- Explain the role of ZooKeeper within the Apache Hadoop infrastructure and the realm of Big Data management
- Explore generic use cases and some real-world scenarios for ZooKeeper
- Define the ZooKeeper services that are used to manage distributed systems
- Explore and use the ZooKeeper CLI to interact with ZooKeeper services
- Understand how Apache Slider works in conjunction with YARN to deploy distributed applications and to monitor them
- Explain how Apache Knox provides peripheral security services to an Hadoop cluster

Unit summary

This is the unit summary for all three topics: ZooKeeper, Slider, and Knox.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1

Explore ZooKeeper

Coordination, Management, and Governance

© Copyright IBM Corporation 2015

Exercise 1: Explore Zookeeper

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1: Explore ZooKeeper

Purpose:

You will connect to ZooKeeper and explore the ZooKeeper files.

Task 1. Connect to ZooKeeper and explore the ZooKeeper files.

The major reference for Apache ZooKeeper can be found on the Apache ZooKeeper website:

- <http://zookeeper.apache.org>
- <http://zookeeper.apache.org/doc/trunk/> (Documentation)

Additional information can be found on the ZooKeeper wiki:

- <https://cwiki.apache.org/confluence/display/ZOOKEEPER/Index>
- <https://cwiki.apache.org/confluence/display/ZOOKEEPER/Tutorial> (Quick Tutorial)

A full course *Developing Distributed Applications Using ZooKeeper*, updated for BigInsights v4 and the Open Data Platform initiative is available on the [BigDataUniversity.com \(BDU\)](http://bigdatauniversity.com/courses/course/view.php?id=547) at bigdatauniversity.com/courses/course/view.php?id=547.

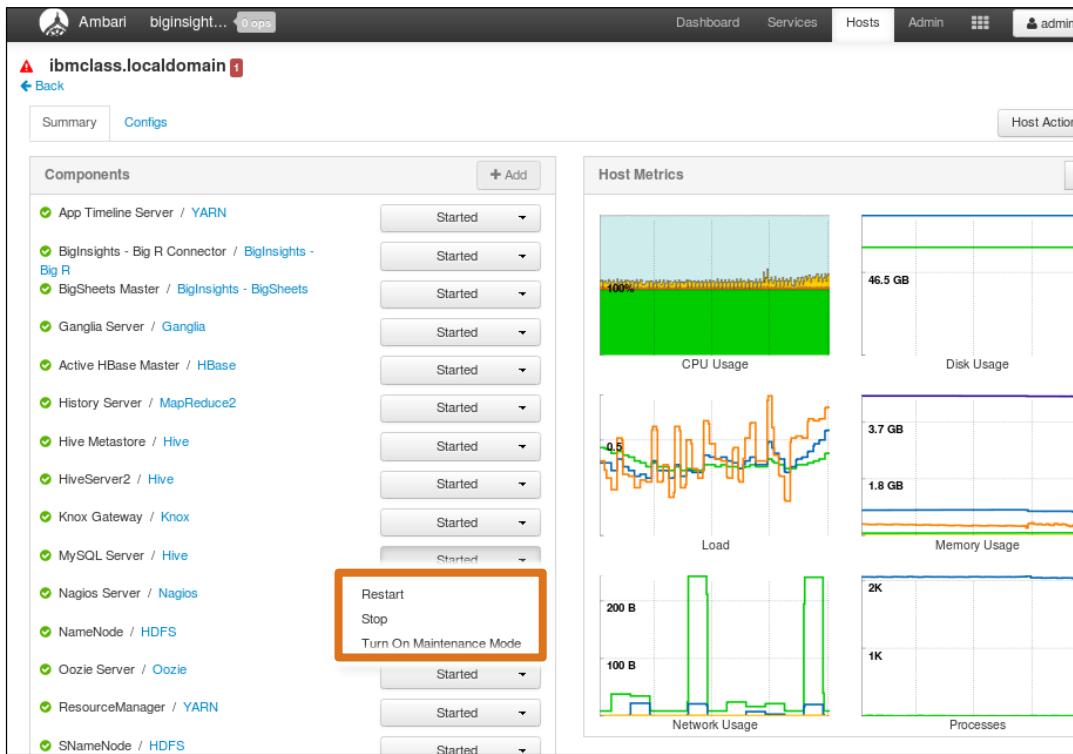
1. Connect to and login to your lab environment with user **biadmin** and password **biadmin** credentials.
2. Launch **Firefox**, and then if necessary, navigate to the **Ambari** login page, <http://localhost:8080>, logging in as **admin/admin**.
3. Click **ZooKeeper** in the left pane to see the Summary information.

The screenshot shows the Ambari interface with the sidebar menu expanded. The 'ZooKeeper' service is highlighted in the sidebar. In the main panel, there are two tabs: 'Summary' and 'Configs'. The 'Summary' tab is selected, displaying the status of the ZooKeeper Server and Client. It shows 'ZooKeeper Server' as 'Started' and 'ZooKeeper Client' as '1 ZooKeeper Client Installed'.

You will see more information about the ZooKeeper Server.

4. Click **ZooKeeper Server** in the **Summary** pane.

This will bring up a detailed list of individual services where from the individual drop-downs you are able to Start, Stop, Restart, Turn on Maintenance Mode.



5. In the **Components** list, beside ZooKeeper, expand the drop down and click **Start** (if stopped) or **Restart** (if currently running).
6. Click **OK** to confirm, and after the background operation is complete, click **OK** to close the Background Operations Running window.
7. Minimize the Ambari Web Console browser (Firefox).

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Task 2. Investigate the ZooKeeper environment.

You will now use a terminal window to navigate to the ZooKeeper home directory on the Linux file system and investigate the directories and files that you will find there.

1. To open a terminal window, right-click your desktop, and then click **Open in Terminal**.
2. Execute the following commands to change directory in the Linux file system to where you will find details of the **ZooKeeper Client**, and list the files that you find there:

```
cd /usr/iop/current/zookeeper-client
pwd
ls -l
[biadmin@ibmclass Desktop]$ cd /usr/iop/current/zookeeper-client
[biadmin@ibmclass zookeeper-client]$ pwd
/usr/iop/current/zookeeper-client
[biadmin@ibmclass zookeeper-client]$ ls -l
total 1328
drwxr-xr-x. 2 root root    4096 Apr 15 12:43 bin
lrwxrwxrwx. 1 root root      19 Apr 15 12:43 conf -> /etc/zookeeper/conf
drwxr-xr-x. 3 root root    4096 Apr 15 12:43 contrib
drwxr-xr-x. 6 root root    4096 Apr 15 12:43 doc
drwxr-xr-x. 2 root root    4096 Apr 15 12:43 lib
drwxr-xr-x. 3 root root    4096 Apr 15 12:43 man
-rw-r--r--. 1 root root 1337298 Mar 28 00:31 zookeeper-3.4.6_IBM_1.jar
lrwxrwxrwx. 1 root root     25 Apr 15 12:43 zookeeper.jar -> zookeeper-
3.4.6_IBM_1.jar
[biadmin@ibmclass zookeeper-client]$
```

Note the names of the various directories. These follow, in general, Linux directory naming conventions:

- **bin**: Executables to start/stop/interact with ZooKeeper
- **conf**: ZooKeeper and log configuration files
- **contrib**: Utilities to help integrate ZooKeeper into other systems: rest, fuse, perl, and python libraries.
- **docs**: ZooKeeper documentation

3. Execute the following commands to set your environmental variable ZOOKEEPER_HOME to the current directory (as you may need it again in this window), change directory to the **bin** subdirectory, and start the **ZooKeeper Client Command Line Interface (CLI)**:

```
export ZOOKEEPER_HOME=`pwd`  
cd bin  
. ./zkCli.sh
```

```
[biadmin@ibmclass zookeeper-client]$ export ZOOKEEPER_HOME=`pwd`  
[biadmin@ibmclass zookeeper-client]$ cd bin  
[biadmin@ibmclass bin]$ ./zkCli.sh  
Connecting to localhost:2181  
2015-06-06 16:22:47,926 - INFO [main:Environment@100] - Client  
environment:zookeeper.version=3.4.6-IBM_1--1, built on 03/28/2015 04:29 GMT  
2015-06-06 16:22:47,929 - INFO [main:Environment@100] - Client  
environment:host.name=ibmclass.localdomain  
2015-06-06 16:22:47,929 - INFO [main:Environment@100] - Client  
environment:java.version=1.7.0_75  
2015-06-06 16:22:47,933 - INFO [main:Environment@100] - Client  
environment:java.vendor=Oracle Corporation  
2015-06-06 16:22:47,934 - INFO [main:Environment@100] - Client  
environment:java.home=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.75.x86_64/jre  
2015-06-06 16:22:47,934 - INFO [main:Environment@100] - Client  
environment:java.class.path=/usr/iop/current/zookeeper-  
client/bin/../../build/classes:/usr/iop/current/zookeeper-  
client/bin/../../build/lib/*.jar:/usr/iop/current/zookeeper-client/bin/../../lib/slf4j-  
log4j12-1.6.1.jar:/usr/iop/current/zookeeper-client/bin/../../lib/slf4j-api-  
1.6.1.jar:/usr/iop/current/zookeeper-client/bin/../../lib/netty-  
3.7.0.Final.jar:/usr/iop/current/zookeeper-client/bin/../../lib/log4j-  
1.2.17.jar:/usr/iop/current/zookeeper-client/bin/../../lib/jline-  
0.9.94.jar:/usr/iop/current/zookeeper-client/bin/../../zookeeper-  
3.4.6_IBM_1.jar:/usr/iop/current/zookeeper-  
client/bin/../../src/java/lib/*.jar:/usr/iop/current/zookeeper-  
client/bin/../../conf:/usr/share/zookeeper/*  
2015-06-06 16:22:47,934 - INFO [main:Environment@100] - Client  
environment:java.library.path=/usr/java/packages/lib/amd64:/usr/lib64:/lib:/us  
r/lib  
2015-06-06 16:22:47,934 - INFO [main:Environment@100] - Client  
environment:java.io.tmpdir=/tmp  
2015-06-06 16:22:47,934 - INFO [main:Environment@100] - Client  
environment:java.compiler=<NA>  
2015-06-06 16:22:47,934 - INFO [main:Environment@100] - Client  
environment:os.name=Linux  
2015-06-06 16:22:47,934 - INFO [main:Environment@100] - Client  
environment:os.arch=amd64  
2015-06-06 16:22:47,934 - INFO [main:Environment@100] - Client  
environment:os.version=2.6.32-504.el6.x86_64  
2015-06-06 16:22:47,935 - INFO [main:Environment@100] - Client  
environment:user.name=biadmin  
2015-06-06 16:22:47,935 - INFO [main:Environment@100] - Client  
environment:user.home=/home/biadmin  
2015-06-06 16:22:47,935 - INFO [main:Environment@100] - Client  
environment:user.dir=/usr/iop/4.0.0.0/zookeeper/bin  
2015-06-06 16:22:47,937 - INFO [main:ZooKeeper@438] - Initiating client connection,  
connectString=localhost:2181 sessionTimeout=30000  
watcher=org.apache.zookeeper.ZooKeeperMain$MyWatcher@53187f60  
Welcome to ZooKeeper!  
2015-06-06 16:22:48,115 - INFO [main-  
SendThread(localhost.localdomain:2181):ClientCnxn$SendThread@975] - Opening socket  
connection to server localhost.localdomain/127.0.0.1:2181. Will not attempt to  
authenticate using SASL (unknown error)  
JLine support is enabled
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

```

2015-06-06 16:22:48,150 - INFO  [main-
SendThread(localhost.localdomain:2181):ClientCnxn$SendThread@852] - Socket connection
established to localhost.localdomain/127.0.0.1:2181, initiating session
[zk: localhost:2181(CONNECTING) 0] 2015-06-06 16:22:48,200 - INFO  [main-
SendThread(localhost.localdomain:2181):ClientCnxn$SendThread@1235] - Session
establishment complete on server localhost.localdomain/127.0.0.1:2181, sessionid =
0x14dcaa8f2000000, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null

[zk: localhost:2181(CONNECTED) 0]

```

You will probably have to press **Enter** to get this last line as your ongoing prompt.

4. Type **help** to get a list of available commands for the ZK CLI.

```

zk: localhost:2181(CONNECTED) 0] help
ZooKeeper -server host:port cmd args
  connect host:port
  get path [watch]
  ls path [watch]
  set path data [version]
  rmr path
  delquota [-n|-b] path
  quit
  printwatches on|off
  create [-s] [-e] path data acl
  stat path [watch]
  close
  ls2 path [watch]
  history
  listquota path
  setAcl path acl
  getAcl path
  sync path
  redo cmdno
  addauth scheme auth
  delete path [version]
  setquota -n|-b val path
[zk: localhost:2181(CONNECTED) 1]

```

5. Type **ls ** in the ZooKeeper CLI prompt.

This tells ZK to list the ZNodes at the top level of the ZooKeeper node hierarchy. For this we need a slash ("\\") after the ls command:

```

[zk: localhost:2181(CONNECTED) 1] ls /
[hiveserver2, hbase-unsecure, zookeeper]
[zk: localhost:2181(CONNECTED) 2]

```

ZooKeeper returns [hiveserver2, hbase-unsecure, zookeeper]. (your results may be slightly different). This means there are three ZNodes at the top of the node hierarchy. The second node listed is used by hbase.

Notice that each interaction in this session with the ZK CLI is numbered: 1, 2, and so on.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

6. To see the subnodes of the hbase node, type `ls /hbase-unsecure`.

```
[zk: localhost:2181(CONNECTED) 2] ls /hbase-unsecure
[meta-region-server, rolllog-proc, backup-masters, table, draining, region-in-
transition, table-lock, running, master, namespace, hbaseid, online-snapshot,
replication, splitWAL, recovering-regions, rs]
[zk: localhost:2181(CONNECTED) 3]
```

7. Type `get /hbase-unsecure` to view the data and metadata stored in any of the nodes.

You should look at the hbase node:

```
[zk: localhost:2181(CONNECTED) 3] get /hbase-unsecure
cZxid = 0xb
ctime = Wed Apr 15 12:12:39 GMT-05:00 2015
mZxid = 0xb
mtime = Wed Apr 15 12:12:39 GMT-05:00 2015
pZxid = 0x723
cversion = 72
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 0
numChildren = 16
[zk: localhost:2181(CONNECTED) 4]
```

The meaning of the data / metadata fields for this response is shown in this table:

Field / Data	Description
Blank line	(Optional) line of text that is actual data stored in this ZNode
cZxid = 0xb	The zxid (ZooKeeper Transaction Id) of the change that caused this znode to be created
ctime = Wed Apr 15 12:12:39 GMT-05:00 2015	The time when this znode was created
mZxid = 0xb	The zxid of the change that last modified this znode
mtime = Wed Apr 15 12:12:39 GMT-05:00 2015	The time when this znode was last modified.
pZxid = 0x723	The zxid of the change that last modified children of this znode.
cversion = 72	The number of changes to the children of this znode.
dataVersion = 0	The number of changes to the data of this znode.
aclVersion = 0	The number of changes to the ACL of this znode.
ephemeralOwner = 0x0	The session id of the owner of this znode if the znode is an ephemeral node. If it is not an ephemeral node, it will be zero
dataLength = 0	The length of the data field of this znode (zero in this case, since blank)
numChildren = 16	The number of children of this znode.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

There are a number of other commands that can be applied against this ZNode and other ZNodes of the ZooKeeper Server. For further details check the documentation.

Extra exercises are available in the BDU course *Developing Distributed Applications Using ZooKeeper* and excellent documentation can be found on <http://zookeeper.apache.org>.

8. Close all open windows.

Results:

You connected to ZooKeeper and explored the ZooKeeper files.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit 12 Data Movement

IBM Training



Data Movement

IBM BigInsights v4.0

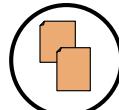
© Copyright IBM Corporation 2015
Course materials may not be reproduced in whole or in part without the written permission of IBM.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit objectives

- List some of the load scenarios that are applicable to Hadoop
- Understand how to load data at rest
- Understand how to load data in motion
- Understand how to load data from common sources such as a data warehouse, relational database, web server, or database logs
- Explain what Sqoop is and how it works
- Describe how Sqoop can be used to import data from relational systems into Hadoop and export data from Hadoop into relational systems
- Explain what Flume is and how it works
- Describe the components of the Flume configuration file

Load scenarios



Data at rest



Data in motion



Data from web server and database logs



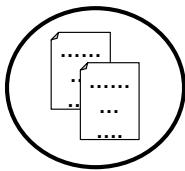
Data from a data warehouse

Load scenarios

You will several load scenarios in this unit:

- Data at rest
- Data in motion
- Data from web server and database logs
- Data from a data warehouse

Data at rest



Data at rest

- Data is already generated into a file or directories.
- No additional updates will be done to the file and will be transferred over as is

Standard HDFS shell commands - using CLI or a graphical interface

Use the `hadoop fs -copyFromLocal` or `-put` commands

Name	Size	Block Size	Permission	Owner
books.csv	420.0 B	128.0 MB	rwx-r--r-	bigadmin

```

1. J. K. Rowling,The Sorcerer's Stone,1997
2. J. K. Rowling,Harry Potter and the Chamber of Secrets,1998
3. J. K. Rowling,The Prisoner of Azkaban,1999
4. J. K. Rowling,Harry Potter and the Goblet of Fire,2000
5. J. K. Rowling,Harry Potter and the Order of the Phoenix,2003
6. J. K. Rowling,Harry Potter and the Half-Blood Prince,2005
7. J. K. Rowling,Harry Potter and the Deathly Hallows,2007
8. David Baldacci,The Devil's Advocate,2000
9. David Baldacci,The Collector,2000
10. David Baldacci,The Painted House,2000
11. David Baldacci,Divine Justice,2000
12. David Baldacci,One False Step,2001
13. David Baldacci,The War of the Worlds,2002
14. David Baldacci,Hour Game,2004
15. David Baldacci,The Last Run,2007
16. David Baldacci,First Family,2009

```

Data Movement

© Copyright IBM Corporation 2015

Data at rest

Data at rest is already in a file in some directory. No additional updates are planned on this data and it can be transferred as is. This can be accomplished using standard HDFS shell commands, cp or copyFromLocal, or using the BigInsights web console.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Data in motion



Data in motion

Lots of servers generating log files or other data

For example:

- a file that is getting appended
- a file that is being updated

Data from multiple locations that needs to be merged into one single source

For example:

- merge all log files from one directory into one file

Data in motion

What about when data is in motion?

First of all, what is meant by data in motion. This is data that is continually updated. New data might be added regularly to these data sources, data might be appended to a file, or logs might be getting merged into one log. You need to have the capability of merging the files before copying them into Hadoop.

Examples of data in motion include:

- Data from a web server such as WebSphere Application Server or Apache Server
- Data in server logs, or application logs, for example site browsing histories

Load solution using Flume



Flume is a distributed service for collecting data and persisting it in a centralized store

- Can be used to collect data from sources and transfer to other agents
- A number of supported sources
- A number of supported sinks
- Flume agents are placed in source and target locations

Load solution using Flume

Flume is a three tiered distributed service for data collection and possibly processing of the data that consists of logical nodes. The first tier, or agent tier, has Flume agents installed at the sources of the data. These agents then send their data to the second tier, or collector tier. The collectors aggregate the data and in turn forward the data to the final storage tier such as HDFS. Each logical node has a source and a sink. The source tells from where to collect data and the sink specifies to where to send the data. Decorators can be optionally configured that allow for some simple data processing as the data is passed through.

There is also the concept of a physical node as well. A physical node corresponds to a single Java process running on one machine in a single JVM. Usually there is only one physical node on a machine. Each physical node can host multiple logical nodes.

Solution if data is from a data warehouse or RDBMS



Data from a Data Warehouse or RDBMS

1. Use standard database export commands, export the tables into comma delimited files & and then use Hadoop commands to import the data HDFS
2. Sqoop to load directly from relational systems
3. Data from DB2 and Netezza, using proprietary stored procedures and the like
4. Data from DB2, Netezza, Teradata, and other database servers into Big SQL using Big SQL Load

Solution if data is from a data warehouse or RDBMS

When moving data from a data warehouse, (or any RDBMS), you export the data into an operating system file in CSV or other format and then use Hadoop commands to import the data.

If you are working with PureData System for Analytics (Netezza) system or DB2, you may find proprietary stored procedures that allow reading/writing between database tables and HDFS - depending on the particular version of IBM BigInsights.

Data from a web server



Data from a Web Server

- If the source is from a web server such as WebSphere
- Data is typically web server logs or applications logs
- Logs are constantly appended

Data from a web server

Flume is a great tool for collecting data from a web server. Another option would be to use Java Management Extension (JMX) commands.

Topic: Sqoop

Data Movement

© Copyright IBM Corporation 2015

Topic: Sqoop

After completing this topic, you should be able to:

- Explain the use of Sqoop to:
 - Import data from relational database tables into HDFS
 - Export data from HDFS into relational database tables
- Describe the basic Sqoop commands:
 - Import statement
 - Export statement

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Sqoop

- Transfers data between Hadoop and relational databases
 - Uses JDBC
 - Must copy the JDBC driver JAR files for any relational databases to \$SQOOP_HOME/lib
- Uses the database to describe the schema of the data
- Uses MapReduce to import and export the data
 - Import process creates a Java class
 - Can encapsulate one row of the imported table
 - The source code of the class is provided to you
- Can help you to quickly develop MapReduce applications that use HDFS-stored records
- The options import and export work in relation to HDFS (and the opposite of the relation to the database itself)

Sqoop

The sqoop binary is **/usr/bin/sqoop**

If accumulo is needed and installed, \$ACCUMULO_HOME should be set.

Sqoop connection

- Database connection requirements are the same for import and export
- Specify as command line options:
 - JDBC connection string
 - Username
 - Password
 - Table
 - Target directory
 - Number of Mappers
- Or, put the options into a file for execution

```
sqoop import|export \
  --connect jdbc:db2://your.db2.com:50000/yourDB \
  --username db2user --password yourpassword \
  ...
```

Sqoop connection

The example shows using a DB2 database connection. Sqoop works with all databases that have a JDBC connection.

In the exercise later, you will export data from a MySQL database.

Sqoop import

- Imports data from relational tables into HDFS
 - Each row in the table becomes a separate record in HDFS
- Data can be stored
 - Text files
 - Binary files
 - Into HBase
 - Into Hive
- Imported data
 - Can be all rows of a table
 - Can limit the rows and columns
 - Can specify your own query to access relational data
- Target directory (--target-dir)
 - Specifies the directory in HDFS in which to place the data
 - If omitted, the name of the directory is the name of the table

Sqoop import

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Sqoop import examples

- Import all rows from a table

```
sqoop import --connect jdbc:db2://your.db2.com:50000/yourDB \
--username db2user --password db2password --table db2table \
--target-dir sqoopdata
```

- Addition parameters, as needed:

```
--split-by tbl_primarykey
--columns "empno,empname,salary"
--where "salary > 40000"
--query 'SELECT e.empno, e.empname, d.deptname FROM employee e JOIN
department d on (e.deptnum = d.deptnum)'
--as-textfile
--as-avrodatafile
--as-sequencefile
```

Sqoop exports

- Exports a set of files from HDFS to a relation database system
 - Table must already exist
 - Records are parsed based upon user's specifications
- Default mode is insert
 - Inserts rows into the table
- Update mode
 - Generates update statements
 - Replaces existing rows in the table
 - Does not generate an upsert
 - Missing rows are not inserted
 - Not detected as an error
- Call mode
 - Makes a stored procedure call for each record
- --export-dir
 - Specifies the directory in HDFS from which to read the data

Sqoop exports

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Sqoop export examples

- Basic export from files in a directory to a table

```
sqoop export --connect jdbc:db2://your.db2.com:50000/yourDB \
--username db2user --password db2password --table employee \
--export-dir /employeedata/processed
```

- Example calling a stored procedure

```
sqoop export --connect jdbc:db2://your.db2.com:50000/yourDB \
--username db2user --password db2password --call empproc \
--export-dir /employeedata/processed
```

- Example updating a table

```
sqoop export --connect jdbc:db2://your.db2.com:50000/yourDB \
--username db2user --password db2password --table employee \
--update_key empno --export-dir /employeedata/processed
```

Additional export information

- Parsing data
 - Default is comma separated fields with newline separated records
 - Can provide input arguments that override the default
 - If the records to be exports loaded into HDFS using the import command
 - The original generated Java class can be used to read the data
- Transactions
 - Sqoop uses multi-row insert syntax
 - Inserts up to 100 rows per statement
 - Commits work every 100 inserts
 - Commit every 10,000 rows
 - Each export map task operates as a separate transaction

Topic: Flume

Data Movement

© Copyright IBM Corporation 2015

Topic: Flume

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Working with Flume

- After completing this topic, you should be able to:
 - Describe Flume and it is used with Hadoop
 - Explain how Flume works



Flume



"A flume is an open artificial water channel, in the form of a gravity chute, that leads water from a diversion dam or weir completely aside a natural flow."

"Often, the flume is an elevated box structure (typically wood) that follows the natural contours of the land. These have been extensively used in hydraulic mining and working placer deposits for gold, tin, and other heavy minerals."

"They are also used in the transportation of logs in the logging industry, electric power generation, and to power various mill operations by the use of a waterwheel."

- Wikipedia



Data Movement

© Copyright IBM Corporation 2015

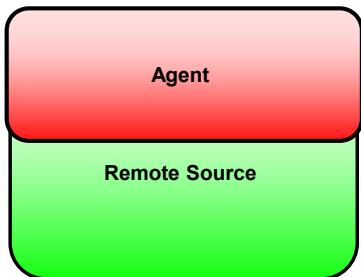
Flume

Flume has agents, installed at a source, which only know how to read data and pass it on to a sink. A source can be a file, or it might be another flume agent. Likewise the sink might be, among other things, HDFS or another flume agent. The final agent in the chain is the collector.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

How Flume works (1 of 3)

- It is built on the concept of flows
- Flows might have different batching or reliability setup
- Flows are comprised of nodes chained together

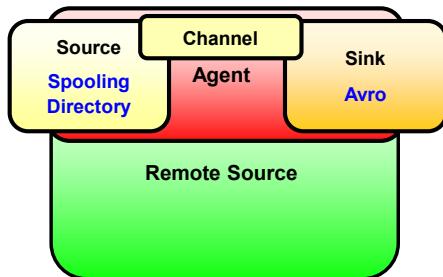


How Flume works

Any number of Flume agents can be chained together. You start by installing a Flume agent where the data originates. This is the agent tier.

How Flume works (2 of 3)

- It is built on the concept of flows
- Flows might have different batching or reliability setup
- Flows are comprised of nodes chained together
 - Each node receives data as "source", stores it in a channel, and sends it via a "sink"



Data Movement

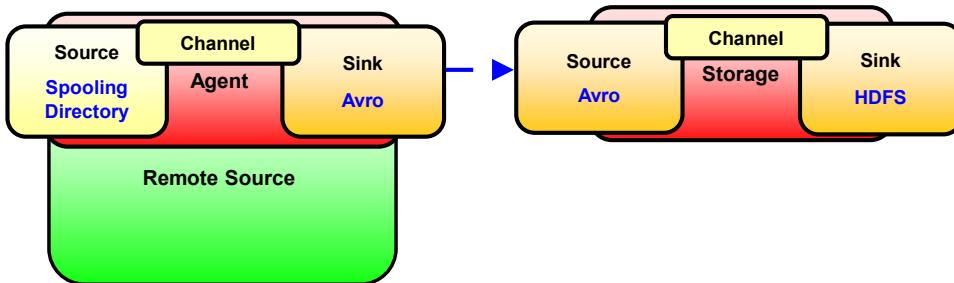
© Copyright IBM Corporation 2015

Every logical node has both a source and a sink. You define the sink to continually watch the tail of some file.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

How Flume works (3 of 3)

- It is built on the concept of flows
- Flows might have different batching or reliability setup
- Flows are comprised of nodes chained together
 - Each node receives data as "source", stores it in a channel, and sends it via a "sink"



Data Movement

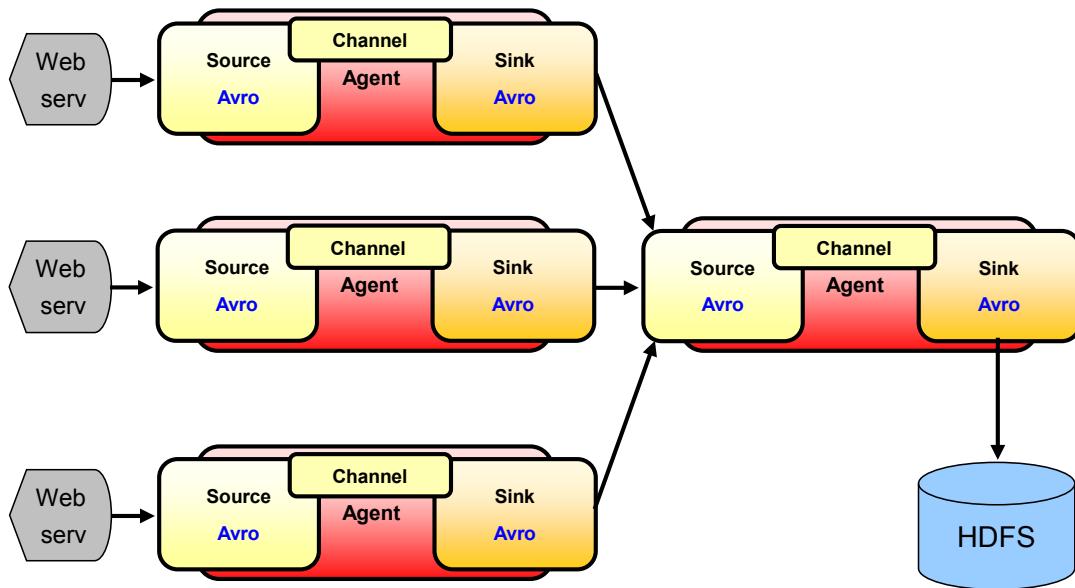
© Copyright IBM Corporation 2015

After capturing data, the logical node at the agent tier sends the data to a logical node at the collector tier. This logical node also has a source and a sink defined. The source is reading from the logical node at the agent tier.

At the collector tier, you will want to do data consolidation. Decorators are defined to accomplish that. You can do some additional data processing by writing your own decorator using a Java-based plug-in API.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Consolidation



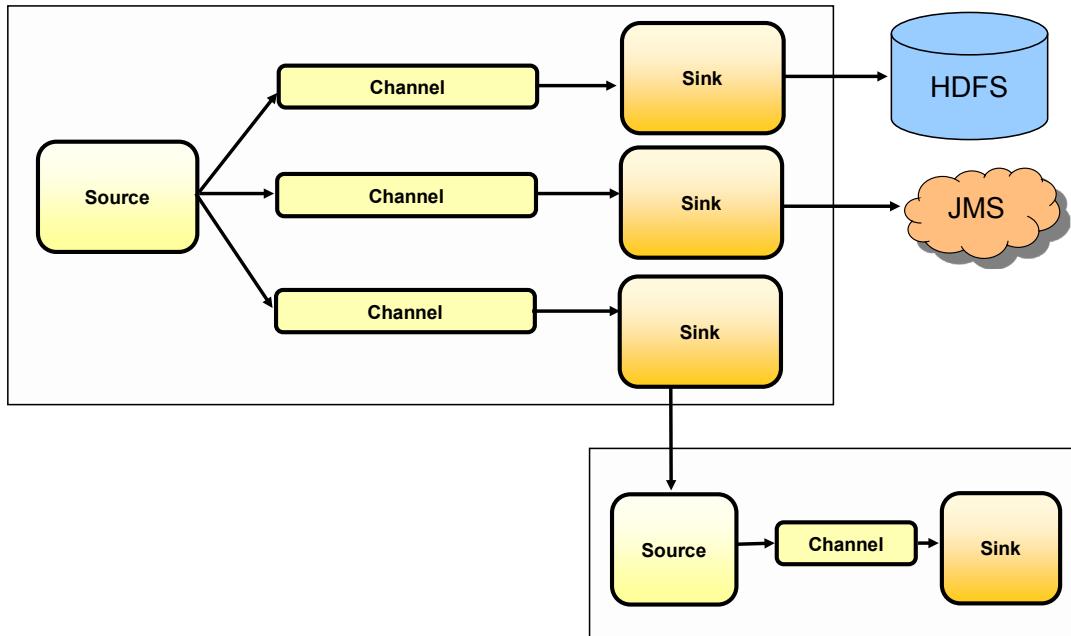
Data Movement

© Copyright IBM Corporation 2015

Consolidation

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Replicating and multiplexing



Data Movement

© Copyright IBM Corporation 2015

Replicating and multiplexing

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Configuration considerations of Flume

- Configuration topics:
 - List the Flume configuration components
 - Describe how interceptors can be used to modify or drop events
 - Explain how sources and sinks are linked using channels
 - Describe how to start a Flume agent

Configuration considerations of Flume

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Configuring Flume

- Flume components are defined in a configuration file
 - Multiple agents running on the same node can be defined in the same configuration file
 - The configuration file resembles a Java properties format
- For each agent, you define the components
 - The source(s)
 - The sink(s)
 - The channel(s)
- Then you define
 - The properties for each component
 - The relationships between the components

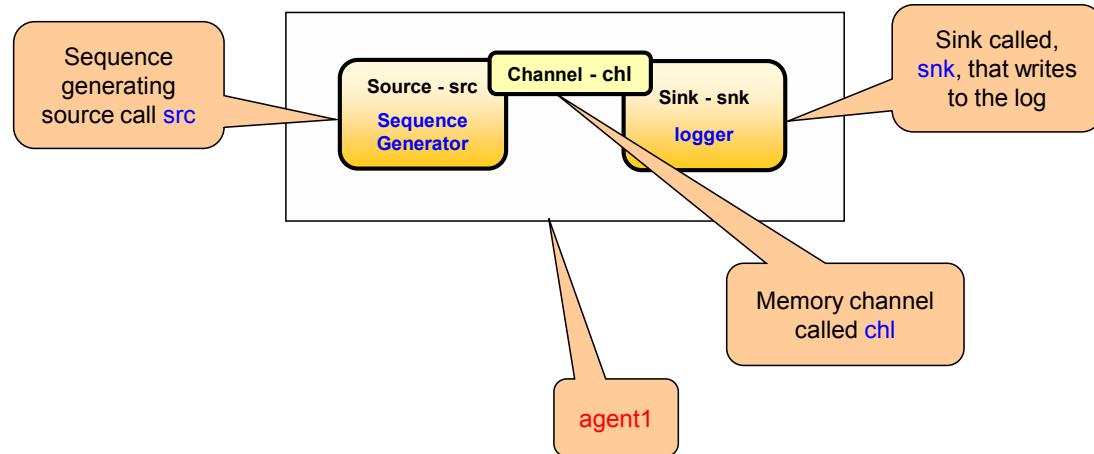
Configuring Flume

Here is an example configuration file (from <https://flume.apache.org/FlumeUserGuide.html>), describing a single-node Flume deployment. This configuration lets a user generate events and subsequently logs them to the console.

```
# example.conf: A single-node Flume configuration
# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# Describe/configure the source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444
# Describe the sink
a1.sinks.k1.type = logger
# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Configuration example



Configuration example

```
# Name components on the agent
agent1.sources = src
agent1.sinks = snk
agent1.channels = chl
# Component properties
agent1.sources.src.type = seq
agent1.sinks.snk.type = logger
agent1.channels.chl.type = memory
agent1.channels.chl.capacity = 100
# Bind the components
agent1.sources.src.channels = chl
agent1.sinks.snk.channel = chl
```

Flume sources

- Avro source: Listens on Avro port
- Exec source: Runs a specified UNIX command on start-up
- Spooling Directory source: Reads data from files in a spooling directory
- NetCat source: Listens on a given port and turns each line of text into an event
- Sequence Generator source: Continuously generates events with a counter
- Syslog source: Reads syslog data
- HTTP source: Accepts events by HTTP POST and GET
- JSONHandler source: Handles events in JSON format
- Legacy sources: Receives events from a Flume 0.9.4 agent
- Scribe source: Accept events from Scribe
- Custom source: Implement your own source interface

Flume sources

Avro source

- Listens on Avro port
- Receives events from external Avro client streams

Exec source

- Runs a specified UNIX command on start-up

Spooling Directory source

- Reads data from files in a spooling directory

NetCat source

- Listens on a given port and turns each line of text into an event

Sequence Generator source

- Continuously generates events with a counter

Syslog source

- Reads syslog data

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

HTTP source

- Accepts events by HTTP POST and GET

JSONHandler source

- Handles events in JSON format

Legacy sources

- Receives events from a Flume 0.9.4 agent

Scribe source

- Accept events from Scribe

Custom source

- Implement your own source interface

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Interceptors

- Give Flume the ability to modify or drop events in flight
- Interceptors are defined in the source configuration
- Predefined interceptors:
 - Timestamp: Inserts the time in milliseconds into the event header
 - Host: Inserts the hostname or ip address of the host into the event header
 - Static: Appends a static header with static values to all events
 - Regex Filtering: Filter events by applying regex expressions to the event body
 - Regex Extractor: Extracts regex matched groups from the body of an event and places the matched groups in the header
 - Custom: You can write your own
- Any source can have multiple interceptors defined, where the defined order is the same order in which interceptors are invoked

Flume sinks

- HDFS sink: Writes events to HDFS
- Logger sink: Logs event at INFO level
- Avro sink: Used with Flume's tiered collection support
- IRC sink: Sends messages to configured IRC destinations
- File Roll sink: Stores events on the local file system
- Null sink: Discards events
- HBaseSink: Writes data to HBase
- ElasticSearchSink: Writes data to ElasticSearch
- Custom sink: Implement your own sink interface

Flume channels

- Memory channel
 - Events are stored in an in-memory queue
- JDBC channel
 - Events are stored in a database
 - Currently supports imbedded Derby
- File channel
 - Stores events in a file
- Custom channel
 - Implement your own channel interface

Flume channel selectors

- Replicating channel selector
 - Default
 - Event is written to each sink
- Multiplexing channel selector
 - An event can be delivered to a subset of sinks
- Custom channel selector
 - Allows you to implement your own channel selector

Configuration details: components

- Defining components

```
<Agent>.sources = <SourceName> [<SourceName> ... ]
```

```
<Agent>.sinks = <SinkName> [<SinkName> ... ]
```

```
<Agent>.channels = <ChannelName> [<ChannelName> ...]
```

Configuration details: components

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Configuration details: properties

- Defining properties for
 - Sources
 - Sinks
 - Channels
-
- <Agent>.sources.<SourceName>.<someProperty> = <someValue>
 - <Agent>.sinks.<SinkName>.<someProperty> = <someValue>
 - <Agent>.channels.<ChannelName>.<someProperty> = <someValue>

Example:

```
agent1.sinks.hdfsSink.type = hdfs
```

Configuration details: bindings

- Bind a source to a sink via a channel

```
<Agent>.sources.<SourceName>.channels = <ChannelName>  
<Agent>.sinks.<SinkName>.channels = <ChannelName>
```

Example:

```
agent1.sources.spoolDirSrc.channels = memChannel
```

Flume example configuration file

```
#Component definition
agent2.sources = spoolDirSrc
agent2.channels = memoryChannel
agent2.sinks = hdfsSink
# For each one of the sources, the type is defined
agent2.sources.spoolDirSrc.type = spooldir
agent2.sources.spoolDirSrc.spoolDir = /home/biadmin/spooldata
agent2.sources.spoolDirSrc.interceptors = ts
agent2.sources.spoolDirSrc.interceptors.ts.type = timestamp
# The channel can be defined as follows.
agent2.sources.spoolDirSrc.channels = memoryChannel
# Each sink's type must be defined
agent2.sinks.hdfsSink.type = hdfs
agent2.sinks.hdfsSink.hdfs.path = hdfs://ibmclass:9000/user/biadmin/flume/%y-%m-%d/%H%M
agent2.sinks.hdfsSink.hdfs.filePrefix = Log
agent2.sinks.hdfsSink.hdfs.writeFormat = Text
agent2.sinks.hdfsSink.hdfs.fileType = DataStream
#Specify the channel the sink should use
agent2.sinks.hdfsSink.channel = memoryChannel
# Each channel's type is defined.
agent2.channels.memoryChannel.type = memory
# Other config values specific to each type of channel(sink or source)
agent2.channels.memoryChannel.capacity = 100
```

Working with an agent

- Agents are invoked from the command line

```
flume-ng agent -n agent_name -c conf  
-f conf/flume.conf -D flume.root.logger=INFO,console
```

- Parameters

- n name of the agent defined in the configuration file
 - c the configuration directory
 - Normal location of the configuration file
 - Location of flume-env.sh
 - If in the configuration directory, will be sourced when Flume start
 - Location of log4j.properties
 - f the configuration file
 - D Java option that overrides a value in the log4j.properties
- Agents are terminated via the kill process
 - ps -ef | grep flume
 - kill -9 pid-of-agents

Working with an agent

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit summary

- List some of the load scenarios that are applicable to Hadoop
- Understand how to load data at rest
- Understand how to load data in motion
- Understand how to load data from common sources such as a data warehouse, relational database, web server, or database logs
- Explain what Sqoop is and how it works
- Describe how Sqoop can be used to import data from relational systems into Hadoop and export data from Hadoop into relational systems
- Explain what Flume is and how it works
- Describe the components of the Flume configuration file

Exercise 1

Moving data into HDFS with Sqoop

Exercise 1: Moving data into HDFS with Sqoop

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1: Moving data into HDFS with Sqoop

Purpose:

You will learn how to move data into an HDFS cluster from a relational database. The relational database that will be used is MySQL since that is already installed in an ODP cluster.

You will connect into the MySQL database and create some data records in internal database storage format.

Sqoop will be used to import that data into HDFS (file movements are in relation to Hadoop and HDFS).

Additional exercises are available in the BigDataUniversity.com course on Moving Data into Hadoop (<http://bigdatauniversity.com/bdu-wp/bdu-course/moving-data-into-hadoop>).

Task 1. Login to your lab environment and connect to the MySQL database.

If you have problems connecting to the MySQL database, it may be for one of several reasons:

- The MySQL database may be not running. Appropriate commands to check status, start/stop, restart the MySQL server (needs root / administrative privileges) are:

```
sudo service mysqld status
sudo service mysqld start
sudo service mysqld stop
sudo service mysqld restart
```

- The database driver may be out-of-date for your version of MySQL:

You can download the latest MySQL Connector/J driver for Linux from <http://dev.mysql.com/downloads/connector/j> and, after unpacking the tar file (tar xvf), place the mysql-connector-java-*.*.*-bin.jar file into /usr/iop/4.0.0.0/sqoop/lib directory.

- As noted previously, you may need to verify that your hostname and IP address are setup correctly, and make changes if they are needed as noted in earlier units. Note that if you have shut down your lab environment anytime, or overnight, this verification of hostname and IP address should be repeated.

1. Connect to your lab environment and login as **biadmin**.
2. In a new terminal window, type **cd** to change to your home directory, and then type **mysql** to start a MySQL command session.

```
[biadmin@ibmclass Desktop]$ cd
[biadmin@ibmclass ~]$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

You will receive a MySQL prompt: **mysql>**

3. To see what databases are currently available in your MySQL server, type

```
mysql> SHOW DATABASES;
```

```
mysql> SHOW DATABASES;
+-----+
| Database      |
+-----+
| information_schema |
| test          |
+-----+
2 rows in set (0.02 sec)
```

```
mysql>
```

Note: SQL (and hence MySQL) commands are case-insensitive. The convention you are using here is that SQL keywords (reserved words) are written in Upper-Case and words which are not keywords are written here in lower-case.

The string "mysql>" is the prompt, and should not be entered.

MySQL commands are terminated by a semi-colon (";").

4. To connect to the test database, type **mysql> USE test;**.
5. Type the following command to create a table called **mytable** in this database:

```
mysql> CREATE TABLE test.mytable (id INT,
name VARCHAR(20));
```

6. Type the following command to insert data into your table:

```
mysql> INSERT INTO test.mytable VALUES
(1,'one'),(2,'two'),(3,'three'),(4,'four'),
(5,'five'),(6,'six'),(7,'seven');
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

7. To verify that your data was actually stored into the database table, type:

```
mysql> SELECT * FROM test.mytable;

mysql> USE test;
Database changed
mysql> CREATE TABLE test.mytable (id INT, name VARCHAR(20));
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO test.mytable VALUES (1,'one'),(2,'two'),(3,'three'),(4,'four'),
-> (5,'five'),(6,'six'),(7,'seven');
Query OK, 7 rows affected (0.00 sec)
Records: 7  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test.mytable;
+----+----+
| id | name |
+----+----+
| 1 | one |
| 2 | two |
| 3 | three |
| 4 | four |
| 5 | five |
| 6 | six |
| 7 | seven |
+----+----+
7 rows in set (0.00 sec)

mysql>
```

8. Type `quit;` since you will not need the database again for awhile.

Task 2. Import data from the database into HDFS.

Sqoop wants to know how many mappers it should employ. Normally, Sqoop would look at the primary key column to figure out how to split the data across the mappers. Since there was not a primary key defined for this table, you will have to help out.

You could use the *split-by <column-name>* parameter to tell Sqoop which column should be used in place of the primary key column. But since you only have one node in your Hadoop cluster, there is no sense in running more than just a single mapper. Do this by specifying the *-m 1* parameter.

1. Import all rows from **mytable** into the scooper directory in Hadoop using the following command:

The sqoop statement is (in one line):

```
sqoop import --connect jdbc:mysql://localhost/test --table mytable
--target-dir scooper -m 1
```

```
[biadmin@ibmclass ~]$ sqoop import --connect jdbc:mysql://localhost/test --table mytable
--target-dir scooper -m 1
Warning: /usr/iop/4.0.0.0/sqoop.../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
15/06/05 18:18:59 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5_IBM_2
15/06/05 18:18:59 ERROR sqoop.ConnFactory: Could not load ManagerFactory
com.ibm.biginights.ie.sqoop.BIConnectionFactory (not found)
15/06/05 18:18:59 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
15/06/05 18:18:59 INFO tool.CodeGenTool: Beginning code generation
15/06/05 18:18:59 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM
`mytable` AS t LIMIT 1
15/06/05 18:18:59 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM
`mytable` AS t LIMIT 1
15/06/05 18:18:59 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is
/usr/iop/4.0.0.0/hadoop
Note: /tmp/sqoop-biadmin/compile/edc36d0be586f056b14ba5b6adc97663/mytable.java uses or
overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
15/06/05 18:19:01 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-
biadmin/compile/edc36d0be586f056b14ba5b6adc97663/mytable.jar
15/06/05 18:19:01 WARN manager.MySQLManager: It looks like you are importing from mysql.
15/06/05 18:19:01 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
15/06/05 18:19:01 WARN manager.MySQLManager: option to exercise a MySQL-specific fast
path.
15/06/05 18:19:01 INFO manager.MySQLManager: Setting zero DATETIME behavior to
convertToNull (mysql)
15/06/05 18:19:01 INFO mapreduce.ImportJobBase: Beginning import of mytable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/hadoop/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/zookeeper/lib/slf4j-log4j12-
1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
15/06/05 18:19:03 INFO impl.TimelineClientImpl: Timeline service address:
http://ibmclass.localdomain:8188/ws/v1/timeline/
15/06/05 18:19:03 INFO client.RMProxy: Connecting to ResourceManager at
ibmclass.localdomain/192.168.244.141:8050
15/06/05 18:19:04 INFO db.DBInputFormat: Using read committed transaction isolation
15/06/05 18:19:04 INFO mapreduce.JobSubmitter: number of splits:1
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

```

15/06/05 18:19:04 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1433542652323_0002
15/06/05 18:19:05 INFO impl.YarnClientImpl: Submitted application
application_1433542652323_0002
15/06/05 18:19:05 INFO mapreduce.Job: The url to track the job:
http://ibmclass.localdomain:8088/proxy/application_1433542652323_0002/
15/06/05 18:19:05 INFO mapreduce.Job: Running job: job_1433542652323_0002
15/06/05 18:19:12 INFO mapreduce.Job: Job job_1433542652323_0002 running in uber mode :
false
15/06/05 18:19:12 INFO mapreduce.Job: map 0% reduce 0%
15/06/05 18:19:17 INFO mapreduce.Job: map 100% reduce 0%
15/06/05 18:19:18 INFO mapreduce.Job: Job job_1433542652323_0002 completed successfully
15/06/05 18:19:18 INFO mapreduce.Job: Counters: 30
    File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=122722
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=87
        HDFS: Number of bytes written=48
        HDFS: Number of read operations=4
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=1
        Other local map tasks=1
        Total time spent by all maps in occupied slots (ms)=3136
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=3136
        Total vcore-seconds taken by all map tasks=3136
        Total megabyte-seconds taken by all map tasks=3211264
    Map-Reduce Framework
        Map input records=7
        Map output records=7
        Input split bytes=87
        Spilled Records=0
        Failed Shuffles=0
        Merged Map outputs=0
        GC time elapsed (ms)=40
        CPU time spent (ms)=910
        Physical memory (bytes) snapshot=197353472
        Virtual memory (bytes) snapshot=1661890560
        Total committed heap usage (bytes)=230686720
    File Input Format Counters
        Bytes Read=0
    File Output Format Counters
        Bytes Written=48
15/06/05 18:19:18 INFO mapreduce.ImportJobBase: Transferred 48 bytes in 16.2116 seconds
(2.9609 bytes/sec)
15/06/05 18:19:18 INFO mapreduce.ImportJobBase: Retrieved 7 records.
[biadmin@ibmclass ~]$
```

Note that a MapReduce job was created and run. There was one Mapper and no Reducers. There is one output file, and it has 7 records.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

- To see what has been stored in the HDFS file system and the content of the file that was created, type the following commands:

```
hadoop fs -ls -R
hadoop fs -cat scooper/p*
```

```
[biadmin@ibmclass ~]$ hadoop fs -ls -R
drwx-----  - biadmin biadmin      0 2015-06-05 18:10 .staging
drwxr-xr-x  - biadmin biadmin      0 2015-06-05 18:10 scooper
-rw-r--r--   3 biadmin biadmin      0 2015-06-05 18:10 scooper/_SUCCESS
-rw-r--r--   3 biadmin biadmin    48 2015-06-05 18:10 scooper/part-m-00000
```

```
[biadmin@ibmclass ~]$ hadoop fs -cat scooper/p*
1,one
2,two
3,three
4,four
5,five
6,six
7,seven
```

Task 3. Import data from the database into HDFS using a sqoop script file.

Typing all of those statements into a command line, knowing that when you close the command line window, the command will be lost, seems like a waste. But what if your commands could be saved in a text file? That might be worth something. Also, what if you wanted to limit the rows imported? You will review each of these ideas.

Note that the UNIX rule for parameters applies. When the parameter name is a single letter, a single dash is used (`-m`); but when a parameter name has more than one letter, a double dash is used (`--connect`).

- From the command line, type `gedit &`.
- In the document window, type your import parameters as seen below:

Notice that you can add comments.

```
import
--connect
jdbc:mysql://localhost/test
--table
mytable
--target-dir
sqoopimport2
# Only select some rows
--where
ID > 4
# Remaining options should be specified on the command line
```

- From the **File** menu, click **Save As**, type **sqoop.params** as the filename, and then click **Save**.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

4. From the command line, type the following:

```
sqoop --options-file ~/sqoop.params -m 1
```

5. View your results and take note of how many records were imported.

Results:

You moved data into an HDFS cluster from a relational database. You connected into the MySQL database and created some data records in internal database storage format.

Sqoop was used to import that data into HDFS (file movements are in relation to Hadoop and HDFS).

Unit 13 Storing and Accessing Data

IBM Training



Storing and Accessing Data

IBM BigInsights v4.0

© Copyright IBM Corporation 2015
Course materials may not be reproduced in whole or in part without the written permission of IBM.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit objectives

- List the characteristics of representative data file formats, including flat/text files, CSV, XML, JSON, and YAML
- List the characteristics of the our types of NoSQL datastores
- Describe the storage used by HBase in some detail
- Describe and compare the open source programming languages, Pig and Hive
- List the characteristics of programming languages typically used by Data Scientists: R and Python

Unit objectives

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Introduction to data

- "Data are values of qualitative or quantitative variables, belonging to a set of items" - Wikipedia
- Common data representation formats used for big data include:
 - Row- or record-based encodings:
 - Flat files / text files
 - CSV and delimited files
 - Avro / SequenceFile
 - JSON
 - Other formats: XML, YAML
 - Column-based storage formats:
 - RC / ORC file
 - Parquet
 - NoSQL datastores
- Compression of data

Introduction to data

Storing petabytes of data in Hadoop is relatively easy, but it is more important to choose an efficient storage format for faster querying.

Row-based encodings (Text, Avro, JSON) with a general purpose compression library (GZip, LZO, CMX, Snappy) are common mainly for interoperability reasons, but column-based storage formats (Parquet, ORC) provide not only faster query execution by minimizing IO but also great compression.

Compression is important to big data file storage:

- Reduces file sizes and thus speeds up transfer to/from disk
- Generally faster to transfer a small file and then decompress than to transfer a larger file

Gathering and cleaning/munging/wrangling data

- "In our experience, the tasks of exploratory data mining and data cleaning constitute 80% of the effort that determines 80% of the value of the ultimate data."
 - Dasu, T., & Johnson, T. (2003) *Exploratory data mining and data cleaning*
- Sources of error in data: Data quality/veracity issues
 - Data entry errors (such as call center records manually entered)
 - Measurement errors (such as bad/inappropriate sampling)
 - Distillation errors (such as smoothing due to noise)
 - Data integration errors (such as multiple databases)
- Data manipulation
 - Filtering or subsetting
 - Transforming: add new variables or modify existing variables
 - Aggregating: collapse multiple values into a single value
 - Sorting: change the order of values

Gathering and cleaning/munging/wrangling data

Often the data that is gathered (raw data) needs to be seriously processed and even converted/transformed either before or in the process of loading into HDFS.

There is no settled terminology for the set of activities between acquiring and modeling data. You will see the phrase data preparation to describe these activities. Data preparation seeks to turn newly acquired raw data into clean data that can be analyzed and modeled in a meaningful way. This phase of the data science workflow, and subsets of it, have been variously labeled munging, wrangling, reduction, and cleansing. You can use the various terms indifferently, though some of them are often classified as jargon.

Data munging or data wrangling is loosely the process of manually converting or mapping data from one raw form into another format that allows for more convenient consumption of the data with the help of semi-automated tools. This may include further munging, data visualization, data aggregation, training a statistical model, as well as many other potential uses.

Data munging as a process typically follows a set of general steps which begin with extracting the data in a raw form from the data source, munging the raw data using algorithms (like sorting) or parsing the data into predefined data structures, and finally depositing the resulting content into a data sink for storage and future use.[1] Given the rapid growth of the internet such techniques will become increasingly important in the organization of the growing amounts of data available.

In the world of data warehousing, ETL (extract-transform-load) is referred to; here the process is often ELT - load first, transform later.

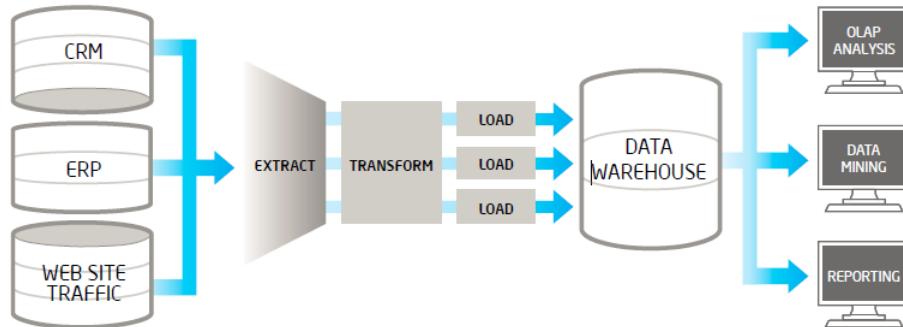
Dasu and Johnson's book (2003) is available online at <https://goo.gl/nloSvj> (this url shortener will open at the quotation).

Data wrangling also the subject of a New York Times article:

<http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html> (For Big-Data Scientists, "Janitor Work" Is Key Hurdle to Insights).

Flat files/text files

- The traditional ETL (extract-transform-load) process extracts data from multiple sources, then cleanses, formats, and loads it into a data warehouse for analysis



- Flat files or text files may need to be parsed into fields/attributes
 - Fields may be positional at fixed offset from the beginning of the record
 - Or, text analytics may be required to extract meaning

Flat files/text files

Data Mining Techniques: <http://www.ibm.com/developerworks/library/ba-data-mining-techniques>

CSV and various forms of delimited files

- Familiar to everyone as input/output to spreadsheet applications:
 - Rows correspond to individual records
 - Columns of plain text are separated by comma or some other delimiter (tab, |, etc.)
- May have a header record with names of columns
- Problems:
 - Quotes may be used in order to deal with strings of text
 - Escape characters may be present (typically backslash = \)
 - Windows and Linux/UNIX use different end-of-line characters
- Python has a standard library that includes a CSV package
- Though attractive, the capabilities of CSV-style formats are limited:
 - Assumes each record has a fix number of attributes
 - Not easy to represent sets, lists, or maps, or more complex data structures

CSV and various forms of delimited files

An example of CSV-formatted data with a header row:

```
id,title,description,price
1,shoes,red shoes,$70.00
2,hat,a black hat,$20.00
3,sweater,a wool sweater,$50.00
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Avro/SequenceFile



- Avro data files are a compact, efficient binary format that provides interoperability with applications written in other programming languages
 - Avro also supports versioning, so that when, for example, columns are added or removed from a table, previously imported data files can be processed along with new ones
- SequenceFiles are a binary format that store individual records in custom record-specific data types
 - This format supports exact storage of all data in binary representations, and is appropriate for storing binary data (for example, VARBINARY columns), or data that will be principally manipulated by custom MapReduce programs
 - Reading from SequenceFiles is higher-performance than reading from text files, as records do not need to be parsed).

Avro/SequenceFile

Doug Cutting (one of the original developers of Hadoop) answered the question, What are the advantages of Avro's object container file format over the SequenceFile container format? <http://www.quora.com/What-are-the-advantages-of-Avros-object-container-file-format-over-the-SequenceFile-container-format>

Two primary reasons:

- Language Independence. The SequenceFile container and each Writable implementation stored in it are only implemented in Java. There is no format specification independent of the Java implementation. Avro data files have a language-independent specification and are currently implemented in C, Java, Ruby, Python, and PHP. A Python-based application can directly read and write Avro data files.

Avro's language-independence is not yet a huge advantage in MapReduce programming, since MapReduce programs for complex data structures are generally written in Java. But, once you implement Avro tethers for other languages (<http://s.apache.org/ZOw>, Hadoop Pipes for Avro), then it will be possible to write efficient mappers and reducers in C, C++, Python, and Ruby that operate on complex data structures.

Language independence can be an advantage today however if you'd like to create or access data outside of MapReduce programs from non-Java applications. Moreover, as the IBM data platform expands, IBM would like to be able to include more non-Java applications and to easily interchange data with these applications, so establishing a standard, language-independent data format for this platform is a priority.

- Versioning. If a Writable class changes, if fields are added or removed, the type of a field is changed or the class is renamed, then data is usually unreadable. A Writable implementation can explicitly manage versioning, writing a version number with each instance and handling older versions at read-time. This is rare, but even then, it does not permit forward-compatibility (old code reading a newer version) nor branched versions. Avro automatically handles field addition and removal, forward and backward compatibility, branched versioning, and renaming, all largely without any awareness by an application.

The versioning advantages are available today for Avro MapReduce applications.

JSON format: JavaScript Object Notation

- JSON is a plain-text object serialization format that can represent quite complex data in a way that can be transferred between a user and a program or one program to another program
- Often called the language of Web 2.0
- Two basic structures:
 - Records consisting of maps (aka key/value pairs), in curly braces:
`{name: "John", age: 25}`
 - Lists (aka arrays), in square brackets: [. . .]
- Records and arrays can be nested in each other multiple times
- Support libraries are available in R, Python, and other languages
- Standard JSON format does not offer any formal schema mechanism although there are attempts at developing a formal schema
- APIs that return JSON data: Cnet, Flickr, Google Geocoder, Twitter, Yahoo Answers, Yelp, etc.

JSON format - Java Script Object Notation

JSON (JavaScript Object Notation) is an open standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML. Although originally derived from the JavaScript scripting language, JSON is a language-independent data format. Code for parsing and generating JSON data is readily available in many programming languages.

The primary reference site (www.json.org) describes JSON as built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures with great flexibility in practice. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable between programming languages is also based on these structures.

References:

- Get Started With JSON:
http://www.webmonkey.com/2010/02/get_started_with_json
- GNIP: <https://www.gnip.com>

The two basic data structures of JSON have also been described as dictionaries (maps) and lists (arrays). JSON treats an object as a dictionary where attribute names are used as keys into the map.

- Dictionaries are defined in a way that may be familiar to anyone who has initialized a python dict with some values (or has printed out the contents of a dict), pairs of keys and values, separated by a ":", with each key-value pair delimited by a ",", and each entire object/record surrounded by "{}".
- Lists are also represented using python-like syntax, a sequence of values separated by ",", surrounded by "[]". These two data structures can be arbitrarily nested, e.g., a dictionary that contains a list of dictionaries, etc. A

Additionally, individual attributes can be text strings, surrounded by double quotes (" "), numbers, true/false, or null. Note that there is no native support for a 'set' data structure. Typically a set is transformed into a list when an object is getting written to JSON, which would be input into a set when being consumed, For instance, in python: some_set = set [a list, here]. Quotes in text fields are escaped like \"". Note that when inserted into a file, by convention JSON objects are typically written one per line.

Two examples of JSON

#1:

```
{ "id":1, "name":"josh-shop", "listings":[1, 2, 3] }
{ "id":2, "name":"provost", "listings":[4, 5, 6] }
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

#2, from Wikipedia (at <http://en.wikipedia.org/wiki/JSON>):

```
{  
    "firstName": "John",  
    "lastName": "Smith",  
    "isAlive": true,  
    "age": 25,  
    "address": {  
        "streetAddress": "21 2nd Street",  
        "city": "New York",  
        "state": "NY",  
        "postalCode": "10021-3100"  
    },  
    "phoneNumbers": [  
        {  
            "type": "home",  
            "number": "212 555-1234"  
        },  
        {  
            "type": "office",  
            "number": "646 555-4567"  
        }  
    ],  
    "children": [],  
    "spouse": null  
}
```

Python's standard library includes a JSON package. This is very useful for reading a raw JSON string into a dictionary; however, transforming that map into an actual object, and writing an arbitrary object out into JSON may require some additional programming.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

XML (eXtensible Markup Language)

- XML is an incredibly rich and flexible data representation format
 - Uses markup to provide context for fields in plain text
 - Provides an excellent mechanism for serializing objects and data
 - Widely used as an electronic data interchange (EDI) format within industry sectors
- XML has a formal schema language, written in XML, and data written within the constraints of a schema are guaranteed to be valid for later processing
- Webpages are written in HTML, a variant on XML
 - Web scraping/web harvesting/web data extraction can be used to extract information from websites
 - Web crawling and scraping can be done with languages such as Python, R, and others
- Many of the configuration files used in the Hadoop ecosystem are in XML format

XML(eXtensible Markup Language)

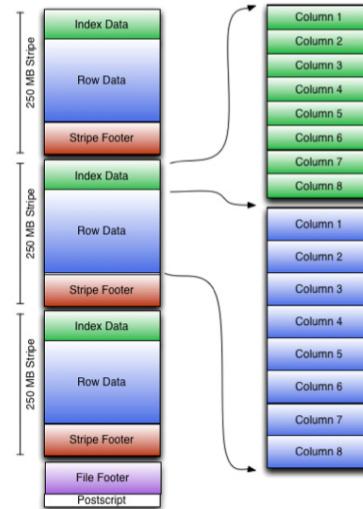
Example of XML formatted data:

```
<Items>
  <listing id=1 title="shoes" price="$70.00">
    <description>red shoes</description>
  </listing>
  <listing id=2 title="hat" price="$20.00">
    <description>black hat</description>
  </listing>
  <listing id=3 title="sweater" price="$50.00">
    <description>a wool sweater</description>
  </listing>
</Items>
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

RC/ORC file formats

- RC (Record Columnar File) file format was developed to support Hive
- ORC (Optimized Row Columnar) format now challenges the RC format by providing an optimized / more efficient approach
 - Specific encoders for different column data types
 - Light-weight indexing that enables skipping of complete blocks of rows
 - Provides basic statistics such as min, max, sum, and count, on columns
 - Larger default blocksize (256MB)



Storing and Accessing Data

© Copyright IBM Corporation 2015

RC/ORD file formats

ORC goes beyond RCFFile and uses specific encoders for different column data types to improve compression further, e.g. variable length compression on integers. ORC introduces a lightweight indexing that enables skipping of complete blocks of rows that do not match a query. It comes with basic statistics, min, max, sum, and count, on columns. Lastly, a larger block size of 256 MB by default optimizes for large sequential reads on HDFS for more throughput and fewer files to reduce load on the namenode.

Reference: <http://www.semantikoz.com/blog/orc-intelligent-big-data-file-format-hadoop-hive>

Parquet

- "Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language"
 - <http://parquet.apache.org>
- Compressed, efficient columnar storage, developed by Cloudera and Twitter
 - Efficiently encodes nested structures and sparsely populated data based on the Google BigTable / BigQuery / Dremel definition/repetition levels
 - Allows compression schemes to be specified on a per-column level
 - Future-proofed to allow adding more encodings to be added as they are invented and implemented
- Provides some of the best results in various benchmark, performance tests

Parquet

This is the official announcement from Cloudera and Twitter about Parquet, an efficient general-purpose columnar file format for Apache Hadoop (from <http://blog.cloudera.com/blog/2013/03/introducing-parquet-columnar-storage-for-apache-hadoop>):

Parquet is designed to bring efficient columnar storage to Hadoop. Compared to, and learning from, the initial work done toward this goal in Trevni, Parquet includes the following enhancements:

- Efficiently encode nested structures and sparsely populated data based on the Google Dremel definition/repetition levels
- Provide extensible support for per-column encodings (such as delta, run length, etc.)
- Provide extensibility of storing multiple types of data in column data (such as indexes, bloom filters, statistics)
- Offer better write performance by storing metadata at the end of the file

- Based on feedback from the Impala beta and after a joint evaluation with Twitter, IBM determined that these further improvements to the Trevni design were necessary to provide a more efficient format that IBM can evolve going forward for production usage. Furthermore, IBM found it appropriate to host and develop the columnar file format outside of the Avro project (unlike Trevni, which is part of Avro) because Avro is just one of many input data formats that can be used with Parquet.
- A new columnar storage format was introduced for Hadoop called Parquet, which started as a joint project between Twitter and Cloudera engineers.
- Parquet was created to make the advantages of compressed, efficient columnar data representation available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model, or programming language.
- Parquet is built from the ground up with complex nested data structures in mind. IBM adopted the repetition/definition level approach to encoding such data structures, as described in Google's Dremel paper; this seems to be a very efficient method of encoding data in non-trivial object schemas.
- Parquet is built to support very efficient compression and encoding schemes. Parquet allows compression schemes to be specified on a per-column level, and is future-proofed to allow adding more encodings as they are invented and implemented. The concepts of encoding and compression are separated, allowing Parquet consumers to implement operators that work directly on encoded data without paying decompression and decoding penalty when possible.
- Parquet is built to be used by anyone. The Hadoop ecosystem is rich with data processing frameworks. An efficient, well-implemented columnar storage substrate should be useful to all frameworks without the cost of extensive and difficult to set up dependencies.
- The initial code defines the file format, provides Java building blocks for processing columnar data, and implements Hadoop Input/Output Formats, Pig Storers/Loaders, and an example of a complex integration, Input/Output formats that can convert Parquet-stored data directly to and from Thrift objects.

Related references:

- Google Big Table: <https://cloud.google.com/files/BigQueryTechnicalWP.pdf>
- Dremel: <http://research.google.com/pubs/pub36632.html>

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

NoSQL

- NoSQL, aka "Not only SQL," aka "Non-relational" was specifically introduced to handle the rise in data types, data access, and data availability needs brought on the dot.com boom
- It is generally agreed that there are four types of NoSQL datastores:
 - Key-value stores
 - Graph stores
 - Column stores
 - Document stores
- Why consider NoSQL?
 - Flexibility
 - Scalability - they scale horizontally rather than vertically
 - Availability
 - Lower operational costs
 - Specialized capabilities

NoSQL

A good discussion of the items described above can be found at:

- https://cloudant.com/wp-content/uploads/Why_NoSQL_IBM_Cloudant.pdf

Examples of each of the four types of NoSQL datastores:

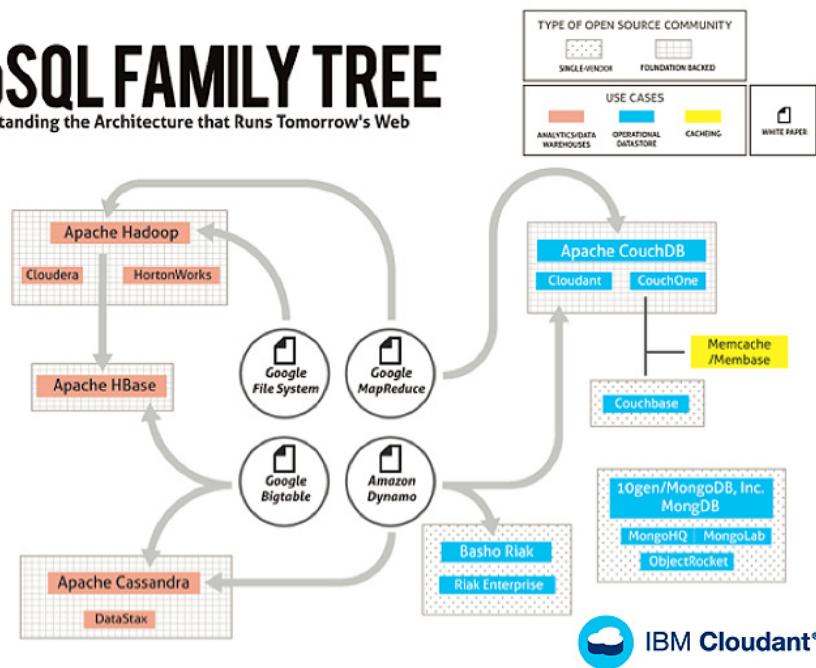
- Key-value stores: MemcacheD, REDIS, and Riak
- Graph stores: Neo4j and Sesame
- Column stores: HBase and Cassandra
- Document stores: MongoDB, CouchDB, Cloudant, and MarkLogic

NoSQL datastores will not be replacing the traditional RDMSs, neither transactional relational databases nor data warehouses. Nor will Hadoop be replacing them either.

Origins of the NoSQL products

NoSQL FAMILY TREE

Understanding the Architecture that Runs Tomorrow's Web



Storing and Accessing Data

© Copyright IBM Corporation 2015

Origins of the NoSQL products

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

HBase

- An implementation of Google's BigTable Design
 - "A Bigtable is a sparse, distributed, persistent multidimensional sorted map"
 - <http://research.google.com/archive/bigtable-osdi06.pdf>
- An open source Apache Top Level Project
 - Embraced and supported by IBM and all leading Hadoop distributions
- Powers some of the leading sites on the Web, such as Facebook, Yahoo, etc.
 - <http://wiki.apache.org/hadoop/Hbase/PoweredBy>
- Is a NoSQL datastore: column datastore

HBase

In 2004 Google began development of a distributed storage system for structured data called BigTable. Google engineers designed a system for storing big data that can scale to petabytes leveraging commodity servers. Projects at Google like Google Earth, web indexing and Google Finance required a new cost effective, robust and scalable system that the traditional RDBMS was incapable of supporting. In November of 2006 Google released a whitepaper describing BigTable (see the URL). Roughly one year later, Powerset created HBase which is a BigTable implementation compliant with the Google specification. In 2008 HBase was released as an open source top-level Apache project (<http://hbase.apache.org>) and HBase is now the Hadoop database.

HBase powers some of the leading sites on the World Wide Web refer to:
<http://wiki.apache.org/hadoop/Hbase/PoweredBy>

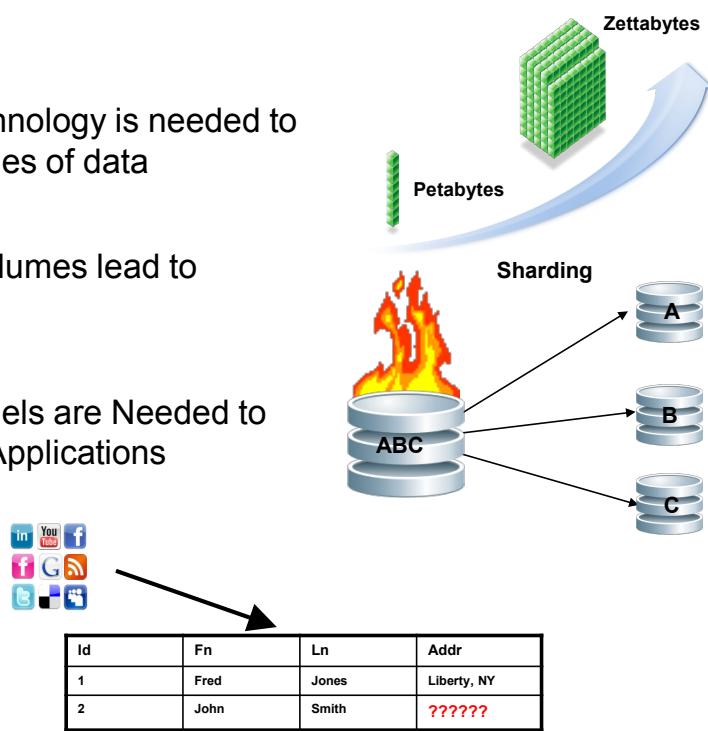
HBase is a NoSQL data store. The slide above explains the tagword "NoSQL" and describes IBM's position and strategy on this new emerging market.

What is HBase?

- An industry leading implementation of Google's BigTable Design
 - What's a BigTable?
 - "A Bigtable is a sparse, distributed, persistent multidimensional sorted map"
<http://research.google.com/archive/bigtable-osdi06.pdf>
- An open source Apache Top Level Project
 - Embraced and supported by IBM
- HBase powers some of the leading sites on the Web (such as Facebook, Yahoo, etc.)
 - <http://wiki.apache.org/hadoop/Hbase/PoweredBy>
- A NoSQL data store

Why NoSQL?

- Cost effective technology is needed to handle new volumes of data
- Increased data volumes lead to RDBMS Sharding
- Flexible Data Models are Needed to Support BigData Applications



Storing and Accessing Data

© Copyright IBM Corporation 2015

Why NoSQL?

So why consider NoSQL Technology? This slide presents three key reasons:

- Massive data sets exhaust the capacity and scale of existing RDBMSs. Buying additional licenses and adding more CPU, RAM and DISK is very expensive and not linear in cost. Many companies and organizations also want to leverage more cost effective commodity systems and open source technologies. NoSQL technology deployed on commodity high volume servers can solve these problems.

Distributing the RDBMS is operationally challenging and often technically impossible. The architecture breaks down when sharding is implemented on a large scale. Denormalization of the data model, joins, referential integrity and rebalancing are common issues.

Un-structured data (such as social media data: Twitter, Facebook, email, etc.) and semi-structured data (such as application logs, security logs) do not fit the traditional model of schema-on-ingest. Typically the schema is developed after ingest and analysis. Un-structured and semi-structured data generate a variable number of fields and variable data content, and are therefore problematic for the data architect when designing the database. There may be many NULL fields (sparse data), and or the number and type of fields may be variable.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Additional consideration points:

- In this new age of big data, most or all of these challenges are typical and as a result the NoSQL market is growing rapidly. Traditional RDBMS technologies and high end server platforms often exceed budgets. Organizations want to leverage commodity high volume servers. Elastic scale out is needed to handle new volumes of data (sensors, log files, social media data, etc.) and increased retention requirements.

Sharding is not cost effective in the age of big data. Sharding creates architectural issues (such as Joins/Denormalization, Referential Integrity and Challenges with Rebalancing).

New applications require a flexible schema. Records can be sparse (e.g. social media data is variable). Schema cannot always be designed up front.

And, briefly:

- Increased complexity of SQL
- Sharding introduces complexity
- Single point of failure
- Failover servers more complex
- Backups more complex
- Operational complexity added

Why HBase in particular?

- Highly Scalable
 - Automatic partitioning (sharding)
 - Scale linearly and automatically with new nodes
- Low Latency
 - Support random read/write, small range scan
- Highly Available
- Strong Consistency
- Very good for "sparse data" (no fixed columns)

Why HBase in particular?

So why are so many businesses and government organizations choosing HBase over other NoSQL technologies?

For starters, HBase is considered "...the Hadoop Database" (see hbase.apache.org) and is bundled with supported Apache Hadoop distributions like IBM's BigInsights product. If you need high performance random read/write access to your big data you are probably going to leverage HBase on your Hadoop cluster. HBase users can easily leverage the Map/Reduce model and other powerful features included with Apache Hadoop.

IBM's strategy for Apache Hadoop is to embrace and extend the technology with powerful advanced analytics, development tooling, performance and availability enhancements, security and manageability. As a key component of Hadoop, HBase is part of this strategy with strong support and a solid roadmap going forward.

When the requirements fit, HBase can replace certain costly RDBMSs.

HBase handles sharding seamlessly and automatically and benefits from the non-disruptive horizontal scaling feature of Hadoop. When more capacity and/or performance is needed, users add datanodes to the Hadoop cluster. This provides immediate growth to HBase data stores since HBase leverages the HDFS. Users can easily scale from TBs to PBs as their capacity needs grow.

HBase supports a flexible and dynamic data model. The Schema does not need to be defined up front which makes HBase a natural fit for many big data applications and some traditional applications as well.

The Apache Hadoop file system, HDFS, does not naturally support applications requiring random read/write capability. HDFS was designed for large sequential batch operations (e.g. write once with many large sequential reads during analysis). HBase does support high performance random r/w applications and for this reason alone it is often leveraged in Hadoop applications.

HBase and ACID Properties

- **Atomicity**
 - All reading and writing of data in one region is done by the assigned Region Server
 - All clients have to talk to the assigned Region Server to get to the data
 - Provides row level atomicity
- **Consistency and Isolation**
 - All rows returned via any access API will consist of a complete row that existed at some point in the table's history
 - A scan is not a consistent view of a table. Scans do not exhibit snapshot isolation. Any row returned by the scan will be a consistent view (for example, that version of the complete row existed at some point in time)
- **Durability**
 - All visible data is also durable data. That is to say, a read will never return data that has not been made durable on disk

HBase and ACID properties

A frequently asked question whenever discussing HBase technology is "How does HBase adhere to the ACID Properties?" The HBase community provided a very good page discussing this, which is summarized on this slide: <http://hbase.apache.org/acid-semantics.html>

- When strict ACID properties are required:
 - HBase provides strict row level atomicity.
 - As far as Apache HBase is concerned - "There is no further guarantee or transactional feature that spans multiple rows or across tables."**
- See also Indexed-Transactional HBase project - <https://github.com/hbase-trx/hbase-transactional-tableindexed>

Note: HBase and other NoSQL distributed data stores are subject to the CAP Theorem which states that distributed NoSQL data stores can only achieve 2 out of the 3 properties: consistency, availability and partition tolerance.

For more information see: <http://hbase.apache.org/acid-semantics.html> and http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

HBase achieves consistency and partition tolerance. See chart "Understanding the CAP Theorem."

Primary reference:

- George, L. (2011). *HBase: The definitive guide*. Savatopol, CA: O'Reilly Media.

Note on Concurrency of writes and mutations

- HBase will automatically get a lock before a write and release that lock after the write
- Also the user can control the locking manually

HBase data model

- Data is stored in HBase table(s)
- Tables are made of rows and columns
- All columns in HBase belong to a particular column family
- Table schema only defines column families
 - Can have large, variable number of columns per row
 - (row key, column key, timestamp) → value
 - A {row, column, version} tuple exactly specifies a cell
- Each cell value has a version
 - Timestamp
- Row stored in order by row keys
 - Row keys are byte arrays; lexicographically sorted

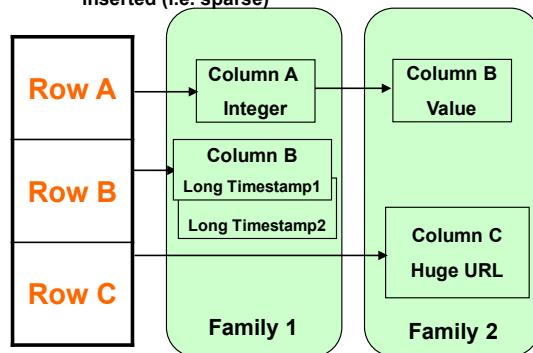
"... a sparse, distributed, persistent, multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterrupted array of bytes"
- Google BigTable paper

Think "Map": this is not a spreadsheet model

- Most literature describes HBase as a Column-Oriented data store which it is, but this can lead to confusion
 - "In computer science, an associative array, map, or dictionary is an abstract data type composed of a collection of (key,value) pairs, such that each possible key appears at most once in the collection."
- Technically HBase is a multidimensional sorted map

	Column A	Column B	Column C
Row A			
Row B			
Row C			

Note: Column Families contain Columns with time stamped versions. Columns only exist when inserted (i.e. sparse)



Think "Map": this is not a spreadsheet model

When researching HBase it's typical to find literature that describes HBase as a column oriented data store which it is.

However, this can lead to confusion and wrong impressions when you try to picture the spreadsheet or traditional RDBMS table model.

HBase is more accurately defined as a "multidimensional sorted map" as seen in the Big Table specification by Google.

Reference: http://en.wikipedia.org/wiki/Associative_array

HBase Data Model: logical view

Table	→ HBTABLE
Row key	Value
11111	<pre>cf_data: {'cq_name': 'name1', 'cq_val': 1111} cf_info: {'cq_desc': 'desc11111'}</pre>
22222	<pre>cf_data: {'cq_name': 'name2', 'cq_val': 2013 @ ts = 2013, 'cq_val': 2012 @ ts = 2012 }</pre>

HFile

11111 cf_data cq_name name1 @ ts1
11111 cf_data cq_val 1111 @ ts1
22222 cf_data cq_name name2 @ ts1
22222 cf_data cq_val 2013 @ ts1
22222 cf_data cq_val 2012 @ ts2

HFile

11111 cf_info cq_desc desc11111 @ ts1

Storing and Accessing Data

© Copyright IBM Corporation 2015

HBase Data Model: logical view

This slide covers the logical representation of an HBase table. A table is made of column families which are the logical and physical grouping of columns.

Each column value is identified by a key. The row key is the implicit primary key. It is used to sort rows.

The table shown on the slide (HBTABLE) has two column families: cf_data, cf_info. Cf_data has two columns with qualifiers cq_name and cq_val. A column in HBase is referred using family:qualifier. Cf_info column family has only one column: cq_desc.

The green boxes show how column families also provide physical separation. The columns in cf_data family are stored separately from columns in cf_info family. This is another important thing to keep in mind when designing the layout of HBase table. If you have data that is not often queried, it is better to assign that to a separate column family.

Column family

- Basic storage unit. Columns in the same family should have similar properties and similar size characteristics
- Configurable by column family
 - Multiple time-stamped versions
 - Such as 3rd dimension to Tables
 - Compression
 - (none, Gzip, LZO, SNAPPY)
 - Version retention policies
 - Time To Live (TTL)
- A column is named using the following syntax: family:qualifier

"Column keys are grouped into sets called column families, which form the basic unit of access control" - Google BigTable paper

HBase vs. traditional RDBMS

	HBase	RDBMS
Data Layout	A sparse, distributed, persistent multidimensional sorted map	Row or Column Oriented
Transactions	ACID Support on Single Row Only	Yes
Query Language	get/put/scan only unless combined with Hive or other technology	SQL
Security	Authentication / Authorization	Authentication / Authorization
Indexes	Row-Key only or special table	Yes
Throughput	Millions of Queries per Sec	Thousands of Queries per Sec
Maximum Database Size	PBs	TBs

Example of a classic RDBMS table

SSN - primary key	Last Name	First Name	Account Number	Type of Account	Timestamp
01234	Smith	John	abcd1234	Checking	20120618...
01235	Johnson	Michael	wxyz1234	Checking	20121118...
01235	Johnson	Michael	aabb1234	Checking	20151123...
01236	Mules	null	null	null	null

Example of a classic RDBMS table

Given the classic RDBMS table as shown in the slide, you will see what this would look like in HBase over the next few pages.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Example of HBase logical view ("records")

Row key	Value (CF, Column, Version, Cell)
01234	info: {'lastName': 'Smith', 'firstName': 'John'} acct: {'checking': 'abcd1234'}
01235	info: {'lastName': 'Johnson', 'firstName': 'Michael'} acct: {'checking': 'wxyz1234'@ts=2013, 'checking': 'aabb1234'@ts=2012}
01236	info: {'lastName': 'Mules'}

Good for sparse data since non-exist columns are just ignored
There are no nulls

Example of HBase logical view ("records")

The example table could be implemented logically as follows in HBase.

Note the time stamp data pointed to by the Row key 01235. This makes HBase multidimensional.

Example of the physical view ("cell")

info Column Family

Row Key	Column Key	Timestamp	Cell Value
01234	info:fname	1330843130	John
01234	info:lname	1330843130	Smith
01235	info:fname	1330843345	Michael
01235	info:lname	1330843345	Johnson
01236	info:lname		Mules

acct Column Family

Row Key	Column Key	Timestamp	Cell Value
01234	acct:checking	1330843130	abcd1234
01235	acct:checking	1330843345	wxyz1234
01235	acct:checking	1330843239	aabb1234

Storing and Accessing Data

© Copyright IBM Corporation 2015

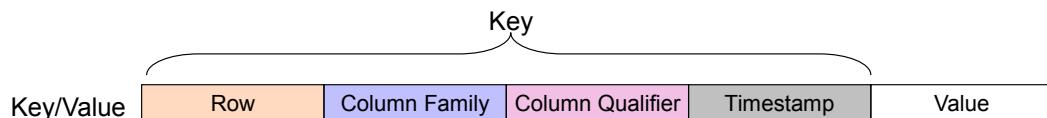
Example of the physical view ("cell")

While the physical cell layout in HBase would look something like this, there are more details to the physical layout of course which will be described further in this presentation (such as how data is actually stored in Apache Hadoop HDFS).

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

More on the HBase data model

- There is no Schema for an HBase Table in the RDBMS sense
 - Except that you have to declare the Column Families
 - Since it determines the physical on-disk organization
 - Thus every row can have a different set of Columns
- HBase is described as a key-value store



- Each key-value pair is versioned
 - Can be a timestamp or an integer
 - Update a column is just to add a new version
- All data are byte arrays, including table name, Column Family names, and Column names (also called Column Qualifiers)

More on the HBase data model

A detailed schema in the RDBMS sense does not need to be defined up front in HBase. You only need to define Column Families, since they impact physical on-disk storage

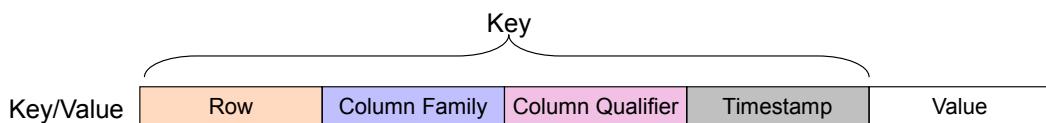
This slide also illustrates why HBase is called a Key/Value pair data store.

Varying the granularity of the key impacts retrieval performance and cardinality when querying HBase for a Value.

Data types are converted to and from the raw byte array format that HBase supports natively.

Indexes in HBase

- All table accesses are via table row key, effectively, its primary key.
- Rows are stored in byte-lexographical order
- Within a row, columns are stored in sorted order, with the result that it is fast, cheap, and easy to scan adjacent rows and columns
 - Partial key lookups also possible
- HBase does not support indexes natively
 - Instead, a Table can be created that serves the same purpose
- HBase supports "bloom filters"
 - Used to decide quickly if a particular row / column combination exists in the store file and reduce IO and access time
- Secondary Indexes are not supported



Indexes in HBase

This slide explains how data in HBase is sorted and can be searched and indexed.

The sorting within Tables makes adjacent queries and scans more efficient.

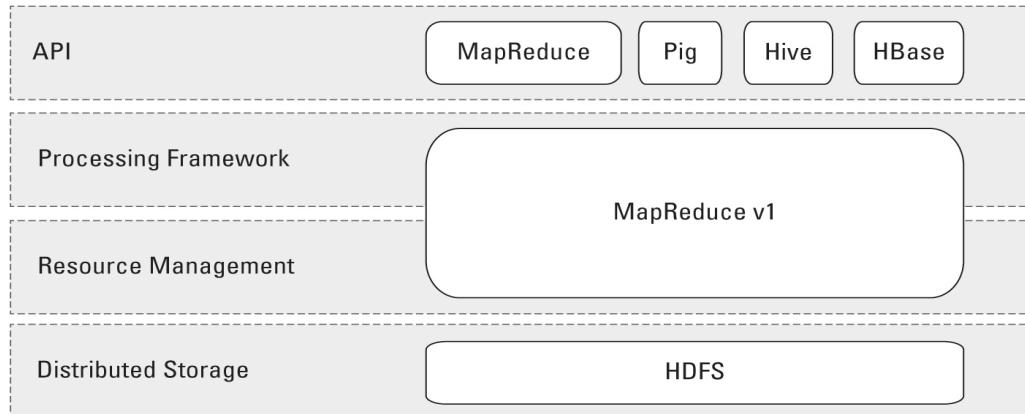
HBase does not support indexes natively but Tables can be created to serve the same purpose.

Bloom filters can be used to reduce IOs and look up time. More information on Bloom filters can be found at: http://en.wikipedia.org/wiki/Bloom_filter

More information on Bloom Filter usage in HBase can be found in: George, L. (2011). *HBase: The definitive guide*. Savastopol, CA: O'Reilly Media, p. 372.

Recap of the Hadoop v1 processing environment

- Multi-layered framework

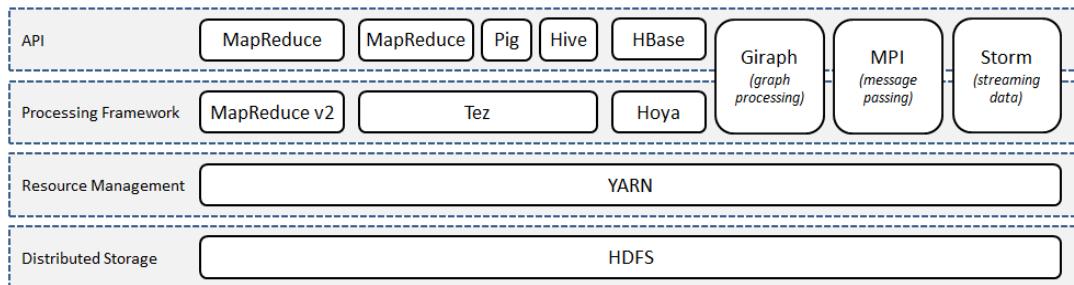


Recap of the Hadoop v1 processing environment

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Processing environment of Hadoop v2

- Four levels in an Hadoop cluster - from the bottom up
 - Distributed storage
 - Resource management
 - Processing framework
 - APIs (application programming interfaces)



Storing and Accessing Data

© Copyright IBM Corporation 2015

Processing environment of Hadoop v2

Diagram from deRoos, D., Zikopoulos, P. C., Melnyk, R. B., Brown, B., & Coss, R. (2014). *Hadoop for dummies*. Hoboken, NJ: Wiley. p. 109.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Open source programming languages: Pig and Hive

Pig: developed originally at Yahoo	Hive: developed originally at Facebook
High-level programming language, excellent for data transformation (ETL) - better than Hive for unstructured data	Provides and SQL-like interface, allowing abstraction of data on top of non-relational, semi-structured data
Language: Pig Latin	Language: HiveQL
Procedural / data-flow language	Declarative language (SQL dialect)
Not suitable for ad-hoc queries, but happy to do grunt work	Very good for ad-hoc analysis, but not necessarily for end users; leverages SQL expertise
Reads data in a large number of file formats, databases	Uses a SerDe (serialization/deserialization) interface to read data from a table and write it back out in any custom format. Many standard SerDes are available, and you can write your own for custom formats
Compiler translates Pig Latin into sequences of MapReduce programs	
Recommended for people are familiar with scripting languages like Python.	Recommended for people who are familiar to SQL

Open source programming languages: Pig and Hive

References:

- <http://pig.apache.org>
- <https://cwiki.apache.org/confluence/display/PIG>
- <http://hive.apache.org>
- <https://cwiki.apache.org/confluence/display/Hive>

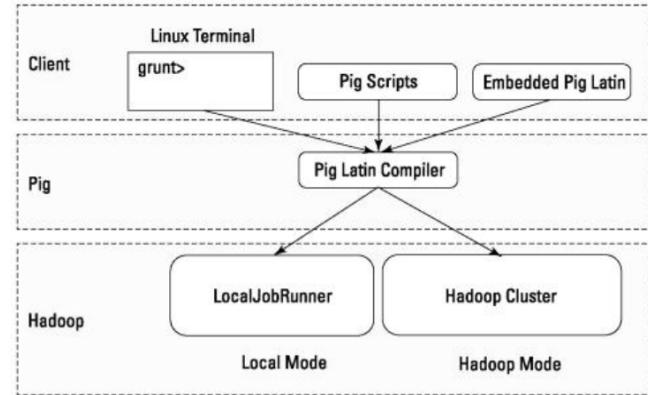
Books:

- Gates, A. (2001). *Programming Pig*. Sebastopol, CA: O'Reilly Media.
- Capriolo, E., Wampler, D., & Rutherford, J. (2012). *Programming Hive*. Sebastopol, CA: O'Reilly Media.
- Lam, C. P., & Davis, M. W. (2015, forthcoming). *Hadoop in action* (2nd ed.). Greenwich, CT: Manning Publications.

Holmes, A. (2015). *Hadoop in practice* (2nd ed.). Greenwich, CT: Manning Publications

Pig

- Pig runs in two modes:
 - Local mode: on a single machine without requirement for HDFS
 - MapReduce/Hadoop mode: execution on an HDFS cluster, with the Pig script converted to a MapReduce job
- When Pig run runs in an interactive shell, the prompt is `grunt>`
- Pig scripts have, by convention, a suffix of `.pig`
- Pig is written in the language *Pig Latin*



Storing and Accessing Data

© Copyright IBM Corporation 2015

Pig

Diagram from: deRoos, D., Zikopoulos, P. C., Melnyk, R. B., Brown, B., & Coss, R. (2014). *Hadoop for dummies*. Hoboken, NJ: Wiley. p. 125

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Pig vs. SQL

- In contrast to SQL, Pig:
 - uses lazy evaluation
 - uses ETL techniques
 - is able to store data at any point during a pipeline
 - declares execution plans
 - supports pipeline splits
- DBMSs are generally substantially faster than the MapReduce system once the data is loaded, but loading the data takes considerably longer in database systems
- RDBMSs offer out-of-the-box support for column-storage, working with compressed data, indexes for efficient random data access, and transaction-level fault tolerance
- Pig Latin is procedural language with a pipeline paradigm
- SQL is a declarative language

Pig vs. SQL

In SQL users can specify that data from two tables must be joined, but not what join implementation to use. But, with some RDBMS systems, extensions ("query hints") are available outside the official SQL query language to allow the implementation of queries and the type of joins to be performed on a single statement basis.

Pig Latin allows users to specify an implementation or aspects of an implementation to be used in executing a script in several ways.

Pig Latin programming is similar to specifying a query execution plan

Characteristics of the Pig language

- Most Pig scripts start with the LOAD statement to read data from HDFS (or from the local file system when running in local mode)
 - In the example on the next slide, you will load data from a .CSV file
 - The USING statement maps the file's data to the Pig data model
- Aggregations are commonly used to summarize dataset
 - Variations include: GROUP, ALL, GROUP ALL
- FOREACH ... GENERATE statements can be used to transform column data

Characteristics of the Pig language

At its core, Pig Latin is a data flow language, where you define a data stream and a series of transformations that are applied to the data as it flows through the application.

This is in contrast to a control flow language (such as C or Java), where you write a series of instructions. In control flow languages, you use constructs such as loops and conditional logic (if and case statements). There are no loops and no if-statements in Pig Latin.

Example of a Pig script

```

input_lines = LOAD '/tmp/my-data' AS (line: chararray);

-- Extract words from each line and put them into a Pig bag,
-- and then flatten the bag to get one word for each row
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(LINE)) AS word;

-- Filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\w+';

-- Create a group for each word
word_groups = GROUP filtered_words BY word;

-- Count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;

-- Order the records by count and write out the results
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/sorted-word-count';

```

Example of a Pig script

The actual program is 7 lines of code.

Lines that start with double-dash are comments, in this case explaining the line(s) that follow.

What is Hive?

- A system for managing and querying structured data built on top of Hadoop
 - MapReduce for execution
 - HDFS for storage
 - Metadata on raw files
- Key Building Principles:
 - SQL is a familiar data warehousing language
 - Extensibility - Types, Functions, Formats, Scripts
 - Scalability and Performance

What is Hive?

The Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage. Built on top of Apache Hadoop, it provides:

- tools to enable easy data extract/transform/load (ETL)
- a mechanism to impose structure on a variety of data formats
- access to files stored either directly in Apache HDFS or in other data storage systems such as Apache HBase
- query execution via MapReduce

Hive defines a simple SQL-like query language, called QL, which enables users familiar with SQL to query the data. At the same time, this language also allows programmers who are familiar with the MapReduce framework to be able to plug in their custom mappers and reducers to perform more sophisticated analysis that may not be supported by the built-in capabilities of the language. QL can also be extended with custom scalar functions (UDF's), aggregations (UDAF's), and table functions (UDTF's).

Hive does not mandate read or written data be in the "Hive format" - there is no such thing. Hive works equally well on Thrift, control delimited, or your specialized data formats. Please see File Formats and Hive SerDe in the Developer Guide for details.

Hive is not designed for OLTP workloads and does not offer real-time queries or row-level updates. It is best used for batch jobs over large sets of append-only data (like web logs). What Hive values most are scalability (scale out with more machines added dynamically to the Hadoop cluster), extensibility (with MapReduce framework and UDF/UDAF/UDTF), fault-tolerance, and loose-coupling with its input formats.

Components of Hive include HCatalog and WebHCat:

- **HCatalog** is a component of Hive. It is a table and storage management layer for Hadoop that enables users with different data processing tools - including Pig and MapReduce - to more easily read and write data on the grid.
- **WebHCat** provides a service that you can use to run Hadoop MapReduce (or YARN), Pig, Hive jobs or perform Hive metadata operations using an HTTP (REST style) interface.

SQL for Hadoop

- Data warehouse augmentation is a very common use case for Hadoop
- While highly scalable, MapReduce is notoriously difficult to use
 - Java API is tedious and requires programming expertise
 - Unfamiliar languages (such as Pig) also requiring expertise
 - Many different file formats, storage mechanisms, configuration options, etc.
- SQL support opens the data to a much wider audience
 - Familiar, widely known syntax
 - Common catalog for identifying data and structure
 - Clear separation of defining the what (you want) vs. the how (to get it)

Java vs. Hive: the wordcount algorithm

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

    public static class Map extends Mapper<LongWritable, Text, Text,
        IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }
}
```

```
public static class Reduce extends Reducer<Text, IntWritable, Text,
    IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
}
```

Storing and Accessing Data

© Copyright IBM Corporation 2015

Java vs. Hive: the wordcount algorithm

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Hive and wordcount

- Wordcount in Java is 70+ lines of Java code
- Below you have the equivalent program in HiveQL, just 8 lines of code and does not require compilation, nor the creation of a JAR file (Java ARchive) to run

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

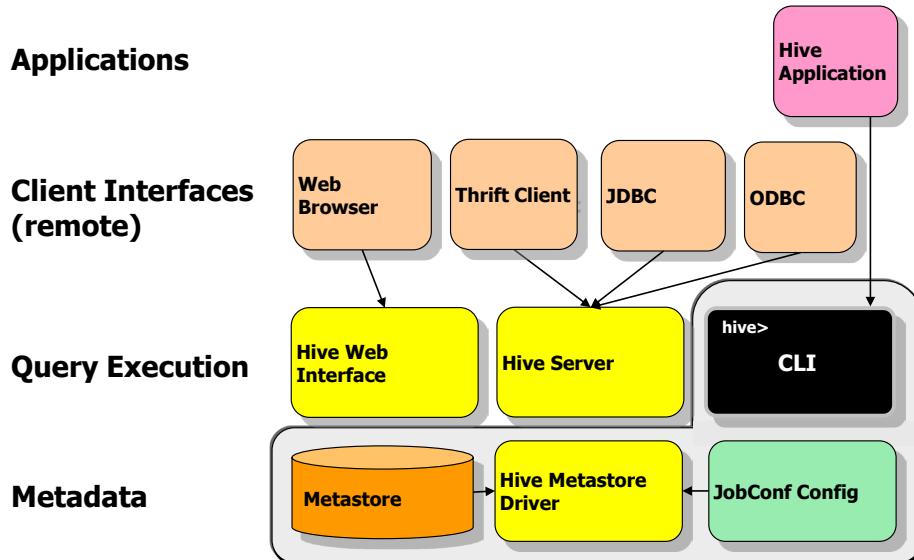
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
    (SELECT explode(split(line, '\s')) AS word
FROM docs)
GROUP BY word
ORDER BY word;
```

Capriolo, E., Wampler, D., & Rutherford, J. (2012). *Programming Hive*.

Hive and wordcount

Capriolo, E., Wampler, D., & Rutherford, J. (2012). *Programming Hive*. Savastopol, CA: O'Reilly Media. p. 12

Hive components



Storing and Accessing Data

© Copyright IBM Corporation 2015

Hive components

These are the major components that you might deal with when working with Hive.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Starting Hive: the Hive shell

- The Hive shell is located in
`$HIVE_HOME/bin/hive`
- From the shell you can:
 - perform queries, DML, and DDL
 - view and manipulate table metadata
 - retrieve query explain plans (execution strategy)

```
$ $HIVE_HOME/bin/hive
2013-01-14 23:36:52.153 GMT : Connection obtained for host: master-
Logging initialized using configuration in file:/opt/ibm/biginsight
Hive history file=/var/ibm/biginsights/hive/query/biadmin/hive_job

hive> show tables;
mytab1
mytab2
mytab3
OK
Time taken: 2.987 seconds
```

Starting Hive: the Hive shell

This Hive shell runs in a command line interface.

Data types and models

- Supports a number of scalar and structured data types
 - tinyint, smallint, int, bigint, float, double
 - boolean
 - string, binary
 - timestamp
 - Array: for example array<int>
 - struct: for example struct<f1:int,f2:array<string>>
 - map: for example map<int,string>
 - union: for example uniontype<int,string,double>
- Partitioning
 - can partition on one or more columns
 - value partitioning only, range partitioning is not yet supported
- Bucketing
 - sub-partitioning/grouping of data by hash within partitions
 - useful for sampling and improves some join operations

Data types and models

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Data model partitions

- Value partition based on partition columns
- Nested sub-directories in HDFS for each combination of partition column values
- Example
 - Partition columns : ds and ctry
 - HDFS subdirectory for ds = 20090801, ctry = US
 - .../hive/warehouse/pview/ds=20090801/ctry=US
 - HDFS subdirectory for ds = 20090801, ctry = CA
 - .../hive/warehouse/pview/ds=20090801/ctry=CA

Data model partitions

Here, partitioning is on columns:

- **ds** (datestamp)
- **ctry** (country)

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Data model external table

- Point to existing data directories in HDFS
- Can create tables and partitions; partition columns just become annotations to external directories
- Example : create external table with partition

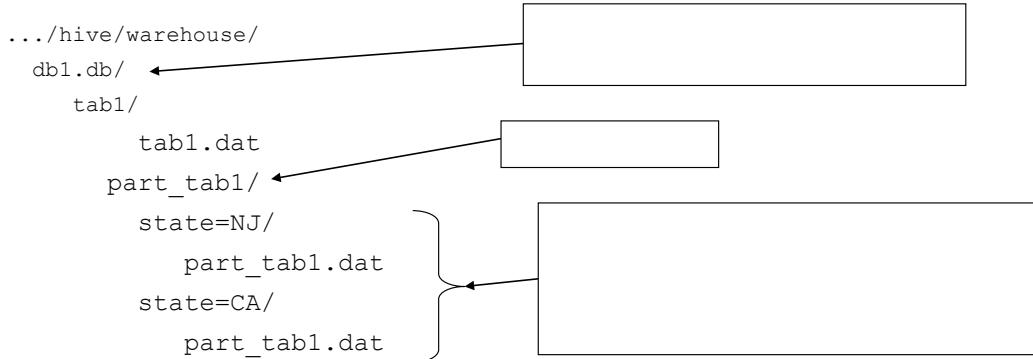
```
CREATE EXTERNAL TABLE pview (userid int, pageid int, ds string, ctry string)
PARTITIONED ON (ds string, ctry string)
STORED AS textfile
LOCATION '/path/to/existing/table'
```

- Example : add a partition to external table

```
ALTER TABLE pview
ADD PARTITION (ds='20090801', ctry='US')
LOCATION '/path/to/existing/partition'
```

Physical layout

- Hive warehouse directory structure



- Data files are just regular HDFS files
 - Internal format can vary table-to-table (delimited, sequence, etc.)
- Supports external tables

Creating a table

- Creating a delimited table

```
hive> create table users
(
    id      int,
    office_id int,
    name    string,
    children array<string>
)
row format delimited
  fields terminated by '|'
  collection items terminated by ':'
stored as textfile;
```

- Inspecting tables:

```
hive> show tables;
OK
users
Time taken: 2.542 seconds

hive> describe users;
OK
id      int
office_id  int
name    string
children array<string>
Time taken: 0.129 seconds
```

file: users.dat

1 1 Bob Smith Mary
2 1 Frank Barney James:Liz:Karen
3 2 Ellen Lacy Randy:Martin
4 3 Jake Gray
5 4 Sally Fields John:Fred:Sue:Hank:Robert

Storing and Accessing Data

© Copyright IBM Corporation 2015

Creating a table

Two levels of separator are used here:

- Column or attribute separator: |
- Collection item separator: :

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

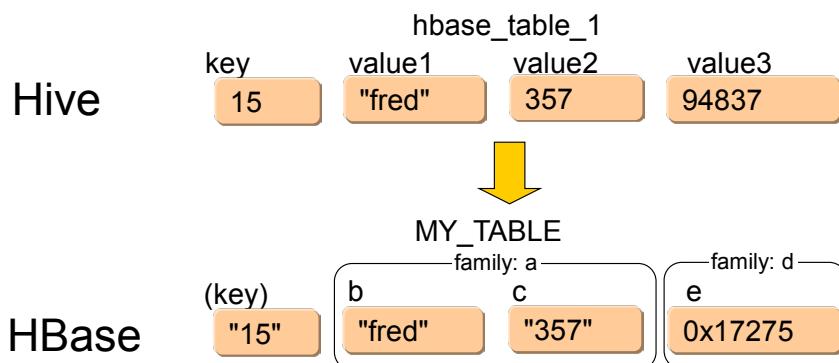
Hive and HBase

- Hive comes with an HBase *storage handler*
- Allows MapReduce queries and loading of HBase tables
- Uses predicate pushdown to optimize query
 - scans only necessary regions based upon table key
 - applies predicates as HBase row filters (if possible)
- Usually Hive must be provided additional jars and configuration in order to work with HBase

```
$ hive \  
  --auxpath \  
    $HIVE_SRC/build/dist/lib/hive-hbase-handler-0.9.0.jar, \  
    $HIVE_SRC/build/dist/lib/hbase-0.92.0.jar, \  
    $HIVE_SRC/build/dist/lib/zookeeper-3.3.4.jar, \  
    $HIVE_SRC/build/dist/lib/guava-r09.jar \  
-hiveconf hbase.master=hbase.yoyodyne.com:60000
```

HBase table mapping

```
CREATE TABLE hbase_table_1 (
    key      int,
    value1   string,
    value2   int,
    value3   int)
WITH SERDEPROPERTIES ("hbase.columns.mapping"= ":key,a:b,a:c,d:e")
TBLPROPERTIES("hbase.table.name" = "MY_TABLE");
```



HBase table mapping

For more information:

- <http://hive.apache.org>
- <https://cwiki.apache.org/confluence/display/Hive/GettingStarted>

Languages used by data scientists: R and Python

- Comparison of programming languages typically used by Data Scientists: R and Python

R	Python
"R is an interactive environment for doing statistics" - "has a programming language, rather than being a programming language"	Real programming language
Rich set of libraries, graphic and otherwise, that are very suitable for Data Science	Lacks some of R's richness for data analytics, but said to be closing the gap
Better if the need is to perform data analysis	Better for more generic programming tasks (e.g. workflow control of a computer model)
Focuses on better, user friendly data analysis, statistics, and data models	Python emphasizes productivity and code readability
More adoption from researchers, data scientists, statisticians, and quants	More adoption from developers and programmers
Active user communities	Active user communities
Standard R library + many, many additional libraries where statistical algorithms often appear first	Python + numpy + scipy + scikit + Django + Pandas

Languages used by data scientists: R and Python

The conclusion is that both are excellent, but they have their individual strengths. Over time, you probably need both.

One approach, if you know Python already, is to use that as your first tool. When you find Python lacking, learn enough R to do what you want, and then either:

- write scripts in R and run them from Python using the subprocess module, or
- install the RPy module

Use Python for what Python is good at and fill in the gaps with one of the above. Use R for plotting things, and Python for the heavy lifting.

Sometimes though, the tool you already know or that is easy to learn, is far more likely to win, than the powerful-but-complex tool that is currently out of your reach.

In the 2013 KDnuggets poll of top languages for data analytics, data mining, and data science, R was the most-used software for the third year running (60.9%), with Python in second place (38.8%). More tellingly, R's usage grew almost four times faster than Python's in 2013 versus 2012 (8.4 percentage points for R, compared to 2.7 percentage points for Python).

Quick overview of R

- R is a free interpreted language
- Best suited for statistical analysis and modeling
 - Data exploration and manipulation
 - Descriptive statistics
 - Predictive analytics and machine learning
 - Visualization
 - +++
- Can produce "publication quality graphics"
- Emerging as a competitor to proprietary platforms
 - Widely used in Universities and companies
 - Not as easy to use or performant as SAS or SPSS
- Algorithms tend to first be available in R
 - Made available by companies and universities as packages
 - rpart(classification), tree(random forest trees), etc.

Quick overview of R

More details on R can be found widely in the web. It is similar to Matlab and SAS/SPSS.

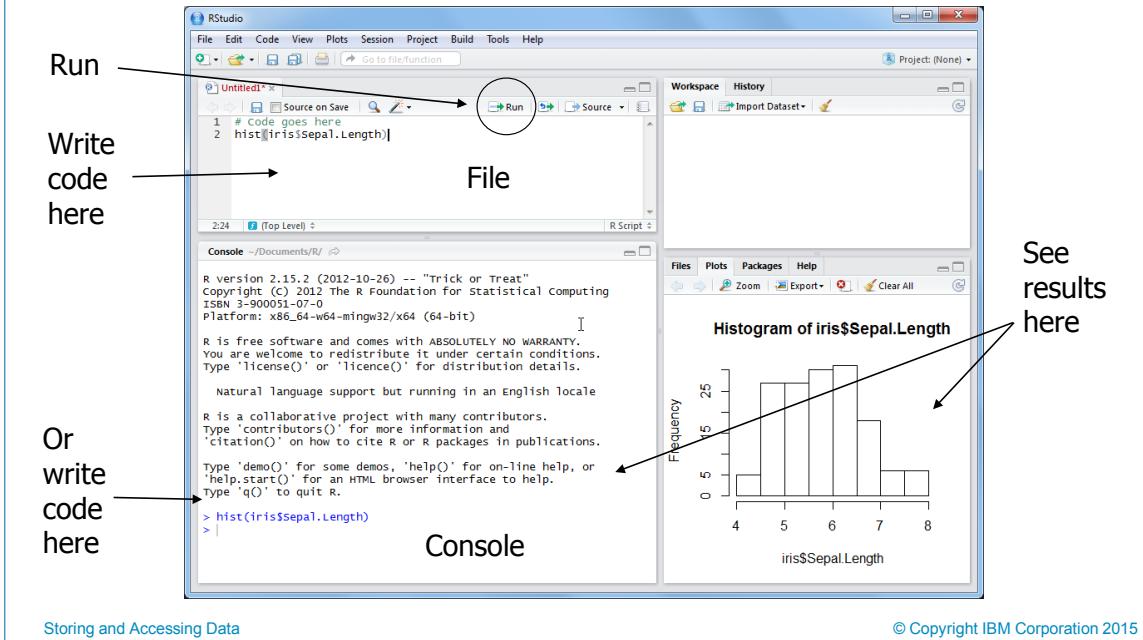
Pluses and minuses:

- + Very sophisticated algorithms,
- + Very good graphics
- Not very good data transformation and file type support
- Not very fast
- Mostly in memory so limited data sizes

More information on IBM's SPSS is beyond the scope of this course.

R clients

RStudio (shown) is a simple, popular IDE, but there are others too



R clients

There are different R clients out there. This is **R Studio** (a free download, available for Windows, Linux, and Mac); this the most common one.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Simple example

- R supports atomic data types, lists, vectors, matrices and data frames
- Data Frames are analogous to database tables
 - Can be created or read from CSV files
- Large set of statistical functions
- Additional functions can be loaded as packages

```
# Vectors
> kidsNames <- c("Henry", "Peter", "Allison")
> kidsAges <- c(7, 11, 17)

# data.frame
> kids <- data.frame(ages = kidsAges, names = kidsNames)

> print(kids)
  ages names
1    7 Henry
2   11 Peter
3   17 Allison

> mean(kids$ages)
[1] 11.66667
```

Simple example

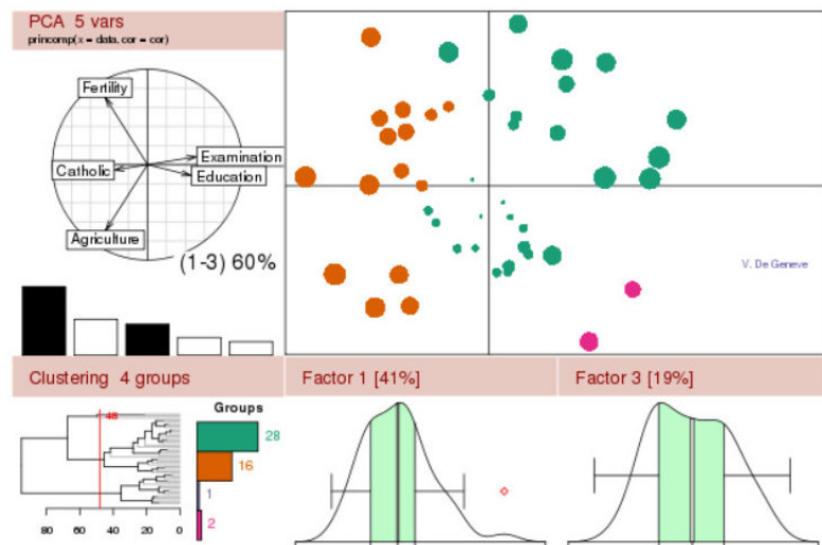
This is a very simple interactive R program that works on basic data. The program creates two vectors then merges them together into one Data frame as two columns. Then shows how to display the table and how to compute a basic average.

This type of interactive use of R can be done in the free,-downloadable RStudio, which runs on Linux, Windows, and Mac.

R is noted for its ability to produce graphical output



The R Project for Statistical Computing



R is noted for its ability to produce graphical output

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Quick overview of Python

- Python is dynamic
 - No need to declare variables: just assign variables (with "=") and use them
 - No explicit Boolean type
 - Classes, with most of the tools you expect in an object-oriented language
- Python does not use braces: begin-end, {}
 - Indentation is used to denote blocks: everything at the same level of indentation is considered in the same block
- Data structures
 - Python has a variety : strings, lists, tuples, dictionaries (hash tables)
 - Python data structures are either mutable (changeable) or not, but strings are not mutable
- And, of course, ...
 - Control constructs: if, for (more like foreach), and while control statements
 - Module, namespaces, ...

Quick overview of Python

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Python wordcount program

- Open the file passed as the parameter to the program (argv[0]) and read as encoded in UTF-8
- Split the text of the line using space, comma, semi-colon, single quote
 - Split() can be called with no argument - in this case, split() uses spaces as the delimiter, with multiple spaces treated as a single space
- Accumulate the count for each word (+=)

```
import sys
file=open(sys.argv[0],"r+", encoding="utf-8-sig")
wordcount={}
for word in file.read().split(" ,;"):
    if word not in wordcount:
        wordcount[word] = 1
    else:
        wordcount[word] += 1
for key in wordcount.keys():
    print ("%s %s " %(key , wordcount[key]))
file.close()
```

Unit summary

- List the characteristics of representative data file formats, including flat/text files, CSV, XML, JSON, and YAML
- List the characteristics of the our types of NoSQL datastores
- Describe the storage used by HBase in some detail
- Describe and compare the open source programming languages, Pig and Hive
- List the characteristics of programming languages typically used by Data Scientists: R and Python

Unit summary

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1

Using Hive to access Hadoop/HBase data

Exercise 1: Using Hive to access Hadoop/HBase data

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Exercise 1: Using Hive to access Hadoop/HBase data

Purpose:

This exercise is intended to provide you with experience in accessing Hadoop/HBase data using a command line interface (CLI).

Task 1. Storing and accessing HBase data.

The major references for the HBase can be found on the Apache.org and Apache Wiki websites:

- <http://hbase.apache.org>
- <http://wiki.apache.org/hadoop/Hbase>

Big Data University has a free courses on HBase at:

- <http://bigdatauniversity.com/bdu-wp/bdu-course/using-hbase-for-real-time-access-to-your-big-data-version-2>

1. Connect to and login to your lab environment with user **biadmin** and password **biadmin** credentials.
2. Launch **Firefox**, and then if necessary, navigate to the **Ambari** login page, <http://localhost:8080>, logging in as **admin/admin**.
3. Verify that Hive is running by clicking on **Hive** in the left panel:

The screenshot shows the Ambari Web Console interface. The left sidebar lists various services: HDFS, MapReduce2, YARN, Nagios, Ganglia, Hive (which is selected), HBase, Pig, Sqoop, and Oozie. The main content area displays the 'Summary' tab for the Hive service. It shows that Hive Metastore, HiveServer2, MySQL Server, and WebHCat Server are all started. Below this, it indicates 1 HCat Client Installed and 1 Hive Client Installed. To the right, there is a 'Service Actions' button and a 'Alerts and Health Checks' section. This section lists three items: 'HiveServer2 process' (TCP OK - 3.616 second response time on port 10000), 'Hive Metastore process' (TCP OK - 0.000 second response time on port 9083), and 'WebHCat Server status' (OK: WebHCat Server status [{"status": "ok", "version": "v1"}] <status_code:200>). All items show a green checkmark and the status 'OK for 2 days'.

If Hive is not running, you will have to start it using the central panel.

When running, minimize the Ambari Web Console browser.

4. In a new terminal window, type `cd` to change to your home directory.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

5. To start the HBase CLI shell, type the following command:

```
/usr/bin/hbase shell
```

```
[biadmin@ibmclass ~]$ /usr/bin/hbase shell
2015-06-07 11:57:36,247 INFO [main] Configuration.deprecation: hadoop.native.lib is
deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.8_IBM_4-hadoop2, rUnknown, Fri Mar 27 21:53:57 PDT 2015

hbase(main):001:0>
```

The last line here is the prompt for the Hbase CLI Client. Note that the interactions are numbered (001) for each CLI session.

6. To view the online CLI help manual, type `help` at the prompt and then press Enter:

```
hbase(main):001:0> help
HBase Shell, version 0.98.8_IBM_4-hadoop2, rUnknown, Fri Mar 27 21:53:57 PDT 2015
Type 'help "COMMAND"', (e.g. 'help "get"' -- the quotes are necessary) for help on a
specific command.
Commands are grouped. Type 'help "COMMAND_GROUP"', (e.g. 'help "general"') for help on a
command group.

COMMAND GROUPS:
Group name: general
Commands: status, table_help, version, whoami

Group name: ddl
Commands: alter, alter_async, alter_status, create, describe, disable, disable_all,
drop, drop_all, enable, enable_all, exists, get_table, is_disabled, is_enabled, list,
show_filters

Group name: namespace
Commands: alter_namespace, create_namespace, describe_namespace, drop_namespace,
list_namespace, list_namespace_tables

Group name: dml
Commands: append, count, delete, deleteall, get, get_counter, incr, put, scan, truncate,
truncate_preserve

Group name: tools
Commands: assign, balance_switch, balancer, catalogjanitor_enabled, catalogjanitor_run,
catalogjanitor_switch, close_region, compact, compact_rs, flush, hlog_roll, major_compact,
merge_region, move, split, trace, unassign, zk_dump

Group name: replication
Commands: add_peer, disable_peer, enable_peer, list_peers, list_replicated_tables,
remove_peer, set_peer_tableCFs, show_peer_tableCFs

Group name: snapshots
Commands: clone_snapshot, delete_snapshot, list_snapshots, rename_snapshot,
restore_snapshot, snapshot

Group name: security
Commands: grant, revoke, user_permission

Group name: visibility labels
Commands: add_labels, clear_auths, get_auths, set_auths, set_visibility

SHELL USAGE:
Quote all names in HBase Shell such as table and column names. Commas delimit
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

command parameters. Type <RETURN> after entering a command to run it. Dictionaries of configuration used in the creation and alteration of tables are Ruby Hashes. They look like this:

```
{'key1' => 'value1', 'key2' => 'value2', ...}
```

and are opened and closed with curly-braces. Key/values are delimited by the '>' character combination. Usually keys are predefined constants such as NAME, VERSIONS, COMPRESSION, etc. Constants do not need to be quoted. Type 'Object.constants' to see a (messy) list of all constants in the environment.

If you are using binary keys or values and need to enter them in the shell, use double-quote'd hexadecimal representation. For example:

```
hbase> get 't1', "key\x03\x3f\xcd"
hbase> get 't1', "key\003\023\011"
hbase> put 't1', "test\xef\xff", 'f1:', "\x01\x33\x40"
```

The HBase shell is the (J)Ruby IRB with the above HBase-specific commands added. For more on the HBase Shell, see <http://hbase.apache.org/book.html>

Take a minute to look through the Help to see what is available to you. Note that in the commands, the names of the tables, rows, column families are all in quotes. You will need to make sure that when you are referring to specific tables, rows, column families, that they are enclosed in quotes.

Practical notes:

- Command (such as create) must be lowercase
- Table name (such as t1) must be quoted, ...
- In interactive mode, you do not need a semicolon as statement terminator or separator (unlike standard SQL)

7. Create an Hbase table using the **create** command:

```
create 't1', 'cf1', 'cf2', 'cf3'
```

to create a table t1 with three column families (cf1, cf2, cf3).

```
hbase(main):002:0> create 't1', 'cf1', 'cf2', 'cf3'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/hadoop/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/zookeeper/lib/slf4j-log4j12-
1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.
0 row(s) in 5.9840 seconds

=> Hbase::Table - t1
hbase(main):003:0>
```

The table *t1* has been created.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Note that these single quotes are inch-type quotes ('') and not Microsoft smart-quotes (""). Take care anytime, here and elsewhere, if you cut-and-paste code that Microsoft or other products have not changed the original code available to use by converting to "smart-quotes".

Other notes:

- The create command only requires the name of the table and one or more column families. Columns can be added dynamically to the table. Also, each row can have a different set of columns (within each column family). However, the table may not be mappable to SQL in such cases.
- Our column family names have been deliberately kept short. This is a best practice: keep your column family names short. For example, instead of 'col_fam1' use 'cf1'. HBase stores the entire names across all of their nodes where the data resides. If you use a long name, it will get repeated across all of the nodes increase the total usage. You want to avoid this by using as short of a name as possible.
- The table name does not need to be short. The name t1 here is short, and cryptic, merely to save your typing convenience. In reality you should use more expressive names as that is better documentation of your data model.

8. Type `describe 't1'` to verify your table creation.

```

hbase(main):001:0> describe 't1'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/hadoop/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/zookeeper/lib/slf4j-log4j12-
1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
Table t1 is ENABLED
t1
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf1', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE =>
'0', VERSIONS => '1', COMPRESSION => 'NONE', M
IN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536',
IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'cf2', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE =>
'0', VERSIONS => '1', COMPRESSION => 'NONE', M
IN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536',
IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'cf3', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE =>
'0', VERSIONS => '1', COMPRESSION => 'NONE', M
IN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536',
IN_MEMORY => 'false', BLOCKCACHE => 'true'}
3 row(s) in 1.4450 seconds

hbase(main):002:0>
```

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Note the following metadata features in your describe response:

- COMPRESSION
- IN_MEMORY
- VERSIONS

You will be changing some of these values shortly with an alter statement:

The next question is: just where is the table t1 stored? It is stored in HDFS, but where in particular?

9. Open another terminal window (so that you can continue later in the first terminal window) and execute the following command:

```
hadoop fs -ls -R / 2>/dev/null | grep t1
```

where this command lists all files (recursive, -R) and passes the results to the Linux command grep. There would be errors, but these are discarded (2>/dev/null).

```
[biadmin@ibmclass Desktop]$ hadoop fs -ls -R / 2>/dev/null | grep t1
drwxr-xr-x  - hbase      hdfs          0 2015-06-07 12:17
/apps/hbase/data/data/default/t1
drwxr-xr-x  - hbase      hdfs          0 2015-06-07 12:17
/apps/hbase/data/data/default/t1/.tabledesc
-rw-r--r--  3 hbase      hdfs         769 2015-06-07 12:17
/apps/hbase/data/data/default/t1/.tabledesc/.tableinfo.0000000001
drwxr-xr-x  - hbase      hdfs          0 2015-06-07 12:17
/apps/hbase/data/data/default/t1/.tmp
drwxr-xr-x  - hbase      hdfs          0 2015-06-07 12:17
/apps/hbase/data/data/default/t1/8a45456f26ee4569360c6af03e893ed6
-rw-r--r--  3 hbase      hdfs         35 2015-06-07 12:17
/apps/hbase/data/data/default/t1/8a45456f26ee4569360c6af03e893ed6/.regioninfo
drwxr-xr-x  - hbase      hdfs          0 2015-06-07 12:17
/apps/hbase/data/data/default/t1/8a45456f26ee4569360c6af03e893ed6/cf1
drwxr-xr-x  - hbase      hdfs          0 2015-06-07 12:17
/apps/hbase/data/data/default/t1/8a45456f26ee4569360c6af03e893ed6/cf2
drwxr-xr-x  - hbase      hdfs          0 2015-06-07 12:17
/apps/hbase/data/data/default/t1/8a45456f26ee4569360c6af03e893ed6/cf3
[biadmin@ibmclass Desktop]$
```

Note that directories are created. Files will be put into those directories as records are stored into the table t1.

For HBase packaged with BigInsights, only gzip compression can be used out of the box. For this, you will use the alter command.

10. In the first terminal window, specify compression for a column family in the table using the following statement:

```
alter 't1', {NAME => 'cf1', COMPRESSION => 'GZ'}
```

```
hbase (main):002:0> alter 't1', {NAME => 'cf1', COMPRESSION => 'GZ'}
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.4600 seconds

hbase (main):003:0>
```

The other compression algorithms that are supported but would need extra configuration are SNAPPY and LZO. Note that GZIP is slow but also has the most efficient compression option.

Sometimes you may find that you need to disable the table prior to executing the alter statement:

```
disable 't1'
alter 't1', {NAME
```

You will now make additional changes to the metadata for the table.

11. Specify the IN_MEMORY option for a column family that will be queried frequently. This does not ensure the data will be in memory always. It only gives priority for the corresponding data to stay in the cache longer.

```
alter 't1', {NAME => 'cf1', IN_MEMORY => 'true'}
```

12. Specify the required number of versions for a column. By default, HBase stores 1 version of the value, but you can set to have more than 1 versions stored.

Enter the following in one continuous line:

```
alter 't1', {NAME => 'cf1', VERSIONS => 3},
{NAME => 'col_fam2', VERSIONS => 2}
```

13. Run the describe statement again, and verify that these changes were made to the table and the column families.

```
describe 't1'
```

14. Insert dates into the table using the put command. Each row could have different set of columns. The below set of put commands inserts two rows with a different set of column names. Go ahead and enter each of these commands (singly or as a group) into the HBase CLI shell, or insert something similar of your choice.

```
put 't1', 'row1', 'cf1:c11', 'r1v11'
put 't1', 'row1', 'cf1:c12', 'r1v12'
put 't1', 'row1', 'cf2:c21', 'r1v21'
put 't1', 'row1', 'cf3:c31', 'r1v31'
put 't1', 'row2', 'cf1:d11', 'r2v11'
put 't1', 'row2', 'cf1:d12', 'r2v12'
put 't1', 'row2', 'cf2:d21', 'r2v21'
```

```
hbase(main):010:0> put 't1', 'row1', 'cf1:c11', 'r1v11'
put 't1', 'row1', 'cf1:c12', 'r1v12'
put 't1', 'row1', 'cf2:c21', 'r1v21'
put 't1', 'row1', 'cf3:c31', 'r1v31'
put 't1', 'row2', 'cf1:d11', 'r2v11'
put 't1', 'row2', 'cf1:d12', 'r2v12'
put 't1', 'row2', 'cf2:d21', 'r2v21'
0 row(s) in 0.3680 seconds

0 row(s) in 0.0410 seconds
0 row(s) in 0.0590 seconds
0 row(s) in 0.0890 seconds
0 row(s) in 0.0520 seconds
0 row(s) in 0.0420 seconds
0 row(s) in 0.0550 seconds

hbase(main):016:0>
```

15. To view the data, you may use the **get** command to retrieve an individual row, or the **scan** command to retrieve multiple rows:

```
get 't1', 'row1'
hbase(main):021:0> get 't1', 'row1'
COLUMN                           CELL
cf1:c11                         timestamp=1433702916069, value=r1v11
cf1:c12                         timestamp=1433702916309, value=r1v12
cf2:c21                         timestamp=1433702916379, value=r1v21
cf3:c31                         timestamp=1433702916476, value=r1v31
4 row(s) in 0.0570 seconds
```

Curiosity point: All data is versioned either using an integer timestamp (seconds since the epoch, 1 Jan 1970 UCT/GMT), or another integer of your choice.

16. If you run the **scan** command, you will see a per-row listing of values:

```
scan 't1'
```

```
hbase (main):022:0> scan 't1'
ROW                                COLUMN+CELL
  row1                             column=cf1:c11, timestamp=1433702916069, value=r1v11
  row1                             column=cf1:c12, timestamp=1433702916309, value=r1v12
  row1                             column=cf2:c21, timestamp=1433702916379, value=r1v21
  row1                             column=cf3:c31, timestamp=1433702916476, value=r1v31
  row2                             column=cf1:d11, timestamp=1433702916539, value=r2v11
  row2                             column=cf1:d12, timestamp=1433702916595, value=r2v12
  row2                             column=cf2:d21, timestamp=1433702916661, value=r2v21
2 row(s) in 0.1230 seconds

hbase (main):023:0>
```

Notes:

- The above scan results show that HBase tables do not require a set schema. This is good for some applications that need to store arbitrary data. To put this in other words, HBase does not store null values. If a value for a column is null (e.g. values for d11, d12, d21 are null for row1), it is not stored. This is one aspect that makes HBase work well with sparse data.
- In addition to the actual column value (*r1v11*), each result row has the row key value (*row1*), column family name (*col_fam1*), column qualifier/column (*c11*) and timestamp. These pieces of information are also stored physically for each value. Having a large number of columns with values for all rows (in other words, dense data) would mean this information gets repeated. Also, larger row key values, longer column family and column names would increase the storage space used by a table. For example use *r1* instead of *row1*.

Good business practices:

- Try to use smaller row key values, column family and qualifier names.
- Try to use fewer columns if you have dense data

Task 2. Storing and accessing HBase data.

The major references for the Hive can be found on the Apache.org and Apache Wiki websites:

- <https://hive.apache.org>
- <https://cwiki.apache.org/confluence/display/Hive/Tutorial>

Big Data University (BDU) has a free courses on Hive:

- <http://bigdatauniversity.com/bdu-wp/bdu-course/accessing-hadoop-data-using-hive-version-2>

You will not have time in this unit to do a full exercise on Hive, but you will start the Hive CLI client to learn where to find it.

1. In a new terminal window, type `cd` to change to the home directory.
2. To start the Hive client, type `hive`.

```
[biadmin@ibmclass ~]$ hive
15/06/07 14:57:18 WARN conf.HiveConf: HiveConf of name hive.optimize.mapjoin.mapreduce
does not exist
15/06/07 14:57:18 WARN conf.HiveConf: HiveConf of name hive.heapsize does not exist
15/06/07 14:57:18 WARN conf.HiveConf: HiveConf of name
hive.auto.convert.sortmerge.join.noconditionaltask does not exist

Logging initialized using configuration in file:/etc/hive/conf/hive-log4j.properties
hive>
```

Some configuration is needed for Hive. With the appropriate configuration and setup, the HBase table that you created can be accessed with Hive and HiveQL.

It is recommended that you take the BDU course and/or continue your learning with one of the many tutorials available on the internet.

3. Close all open windows.

Results:

You accessed Hadoop/HBase data using a command line interface (CLI).

Unit 14 Advanced topics

IBM Training



Advanced topics

IBM BigInsights v4.0

© Copyright IBM Corporation 2015
Course materials may not be reproduced in whole or in part without the written permission of IBM.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit objectives

- Explain the use for Oozie workflows
- Describe a workflow when applied to a big data processing environment
- List some of the workflow elements
- Understand the importance and use of text search engines in exploring big data
- Understand the Solr search procedure
- Identify Solr components

Advanced topics

© Copyright IBM Corporation 2015

Unit objectives

In this unit you will study two topics:

- Ozzie: for developing workflows
- Solr: for developing search on big data

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Topic: Oozie

Advanced topics

© Copyright IBM Corporation 2015

Topic: Oozie

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

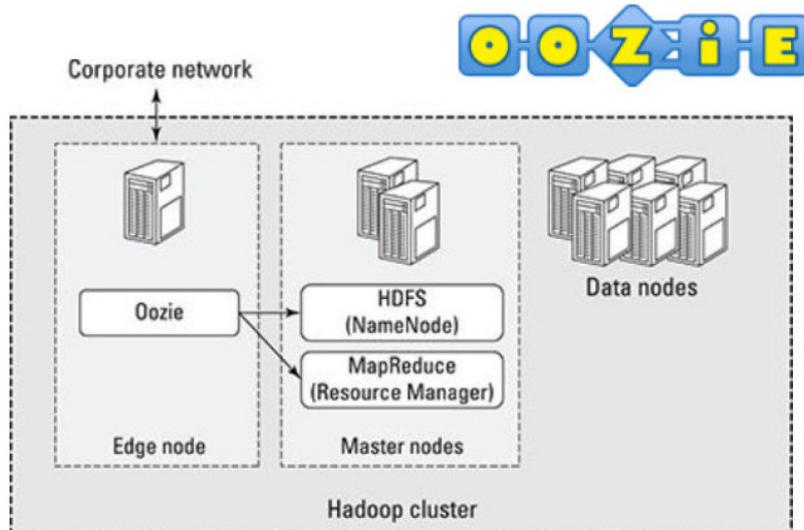
Introduction to Oozie



- Oozie is a workflow scheduler system to manage Apache Hadoop jobs
- Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions
- Oozie Coordinator jobs are recurrent Oozie Workflow jobs that can be triggered by
 - time (frequency), or
 - data availability
- Oozie is integrated with the rest of the Hadoop stack supporting various types of Hadoop jobs out-of-the-box (such as Pig, Hive, Sqoop, Java MapReduce, Streaming MapReduce, and Distcp) as well as system specific jobs (such as Java programs and shell scripts)
- Oozie is a scalable, reliable, and extensible system
- <http://oozie.apache.org>

Oozie Workflow Scheduler for Hadoop

- Oozie supports a wide range of job types, including Pig, Hive, and MapReduce, as well as jobs coming from Java programs and Shell scripts



Advanced topics

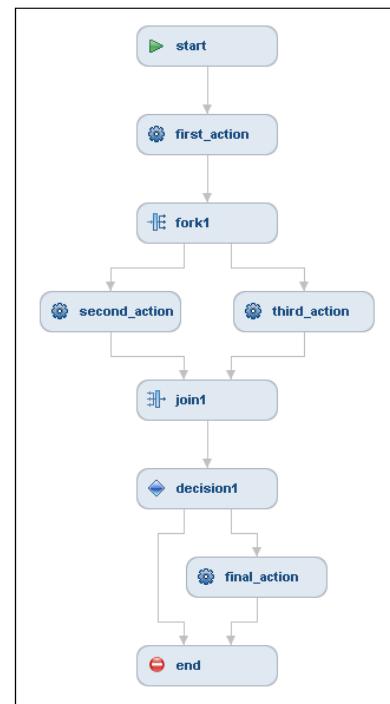
© Copyright IBM Corporation 2015

Oozie Workflow Scheduler for Hadoop

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Elements in a Workflow XML

- Action - invokes a program or script
 - Java
 - MapReduce
 - Pig
 - Hive
 - Sub-workflow (app chaining)
 - File system action (fs)
- Control Nodes - controls the flow
 - Fork, Join, and Decision
- Start → End
 - One directional flow (DAG)
 - No loops
- Kill
 - Used when an error is encountered and the workflow should terminate



Advanced topics

© Copyright IBM Corporation 2015

Elements in a Workflow XML

The worflow.xml file has elements that are: Action, Fork, Join, Decision, Start, End, Kill.

A graphical workflow editor is available in Hue. Hue's target is the user experience and lets end-users focus on quick data processing, directly from their browser (no command line is needed). Hue integrates into a single UI the Hadoop components and their main satellite projects (HDFS, YARN, Hive, Pig, Impala, Spark, Oozie, HBase, Solr, Sqoop2, ZooKeeper, and more).

Additional references:

- <http://gethue.com/tutorial-a-new-ui-for-oozie>
- <http://demo.gethue.com>
- http://demo.gethue.com/oozie/list_workflows
- <http://gethue.com/category/presentation>

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Example of Oozie workflow XML

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="streaming-wf">
  <start to="streaming-node"/>
  <action name="streaming-node">
    <map-reduce>
      <job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>
      <prepare>
        <delete path=" ${nameNode}/user/${wf:user()}/${examplesRoot}/output-data/streaming"/>
      </prepare>
      <streaming>
        <mapper>/bin/cat</mapper>
        <reducer>/usr/bin/wc</reducer>
      </streaming>
      <configuration>
        <property>
          <name>mapred.job.queue.name</name>
          <value>${queueName}</value>
        </property>
        <property>
          <name>mapred.input.dir</name>
          <value>/user/${wf:user()}/${examplesRoot}/input-data/text</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>/user/${wf:user()}/${examplesRoot}/output-data/streaming</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to="end"/>
    <error to="fail"/>
  </action>
  <kill name="fail">
    <message>Streaming Map/Reduce failed, error message[$wf:errorMessage(wf:lastErrorNode())]</message>
  </kill>
  <end name="end"/>
</workflow-app>
```

[Advanced topics](#)

© Copyright IBM Corporation 2015

Example of Oozie workflow XML

The example shown above can be found at:

- <https://github.com/yahoo/oozie/blob/master/examples/src/main/apps/streaming/workflow.xml>

Full examples of Oozie workflow with explanation can be found at:

- <http://www.thecloudavenue.com/2013/10/executing-oozie-workflow-with-pig-hive.html>

Note that **\${nameNode}**, and similar, are parameters passed to the script at run-time.

The example shown above is related to Hadoop 1 and MapReduce 1. More complex scripts are needed for Hadoop 2/YARN environments, but, because of the detailed nature of the XML, it is generally best to use a graphical editor such as Hue to develop and validate the script.

The XML shown here is intended to illustrate some of the XML constructs in a workflow file:

- Outer level is: `<workflow-app ...> ... </workflow-app>`
- Starting point is: `<start>` tag
- Branching labels are: `ok`, `error`, `end`
- Etc. (as noted on the previous slide)

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

How to run an example Oozie application

- Running the Oozie application

```
$ oozie job -oozie http://localhost:8080/oozie -config examples/apps/map-reduce/job.properties -run
job: 14-20090525161321-oozie-tucu
```

- Checking the workflow job status

```
$ oozie job -oozie http://localhost:8080/oozie -info 14-20090525161321-oozie-tucu
```

```
-----
Workflow Name : map-reduce-wf
App Path      : hdfs://localhost:9000/user/tucu/examples/apps/map-reduce
Status        : SUCCEEDED
Run           : 0
User          : tucu
Group         : users
Created       : 2009-05-26 05:01 +0000
Started       : 2009-05-26 05:01 +0000
Ended         : 2009-05-26 05:01 +0000
Actions

-----
Action Name      Type     Status   Transition External Id      External Status Error Code
Start Time      End Time
-----
mr-node          map-reduce OK       end       job_200904281535_0254 SUCCEEDED -
2009-05-26 05:01 +0000 2009-05-26 05:01 +0000
-----
```

[Advanced topics](#)

© Copyright IBM Corporation 2015

How to run an example Oozie application

The free **BigDataUniversity.com** (BDU) course on Oozie (**Controlling Hadoop Jobs using Oozie**) provide a deeper understanding of Oozie:

- <https://bigdatauniversity.com/bdu-wp/bdu-course/controlling-hadoop-jobs-with-oozie>

Note, however, that the BDU course is based on the previous release of IBM BigInsights (namely v3) and uses a workflow editor that is no longer provided with BigInsights (v4). But that course is still very valuable for understanding Oozie and the workflow.xml that supports Oozie.

Future releases of IBM BigInsights may support Hue, another editor that provides Oozie edit capabilities. Some courses based on Hue are available (<http://gethue.com/tutorials>) and others that are not product / vendor specific will undoubtedly become available in the future.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Topic: Apache Solr

Advanced topics

© Copyright IBM Corporation 2015

Topic: Apache Solr

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Rationale for searching

- Crucial to many applications, and often performance on big data can limit the application
- Supports the need to quickly find specific information in large data sets
 - Databases
 - Mail servers
 - Documents
 - Distributed File Systems (like Hadoop)
- Requires more than just text matching
 - Matches are rarely exact
 - Requires spell checking and correction as there can be errors in query and/or document
 - Need to provide ranking: what makes one match better than another?

Rationale for searching

Retrieving information is a crucial function or feature of many applications, and with large datasets it can easily be the limiting factor of performance. You need to be able to quickly search the data of your application, whether it is a database, mail server, documents, or a distributed file system like Hadoop.

It is not enough to use a brute-force approach and simply look for an exact string match. Performance issues aside, what if your query, or the text itself, contains spelling errors? Most likely you do not even know the exact text you are looking for, and want to search by keywords, synonyms, etc. You also need to consider what makes one match better than another and rank the results accordingly.

Solr and other available search technologies

- Whoosh
 - Python indexing and search library
- Xapian
 - Open-source, written in C++, with bindings for Perl, Python, PHP, Java, Tcl, C#, Ruby, Lua, Erlang, Node.js
- ElasticSearch
 - Popular search platform
 - Standalone product (elastic.co) and via Amazon Elastic Search
- Apache Solr
 - Popular, full featured, open-source search platform with many client APIs
 - Integration with other Apache projects - built on top of Lucene

Solr and other available search technologies

The library you choose depends on your needs. If you just really like python, you might choose Whoosh. If you have a simple desktop-only application you may just need Lucene on its own. More complex web applications benefit from the features provided by Solr or ElasticSearch

"Solr is the popular, blazing-fast, open source enterprise search platform built on Apache Lucene" - <http://lucene.apache.org/solr> - current release is v5.1:

- "Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more. Solr powers the search and navigation features of many of the world's largest internet sites."
- "Solr is a standalone enterprise search server with a REST-like API. You put documents in it (called "indexing") via JSON, XML, CSV or binary over HTTP. You query it via HTTP GET and receive JSON, XML, CSV or binary results."

Apache Solr

- Written in Java, based on Lucene
- Solr is a web application
 - Runs on its own server
 - Interact via HTTP
- Text indexing across databases and documents
 - Basic text formats (XML, CSV)
 - Databases, mail servers
 - Special formats (PDF, Word document) with Apache Tika
- High performance
- Highly scalable with SolrCloud
- Native clients for many languages
 - Ruby (+Rails), Python, PHP, Java, JavaScript, C#, .NET

[Advanced topics](#)

© Copyright IBM Corporation 2015

Apache Solr

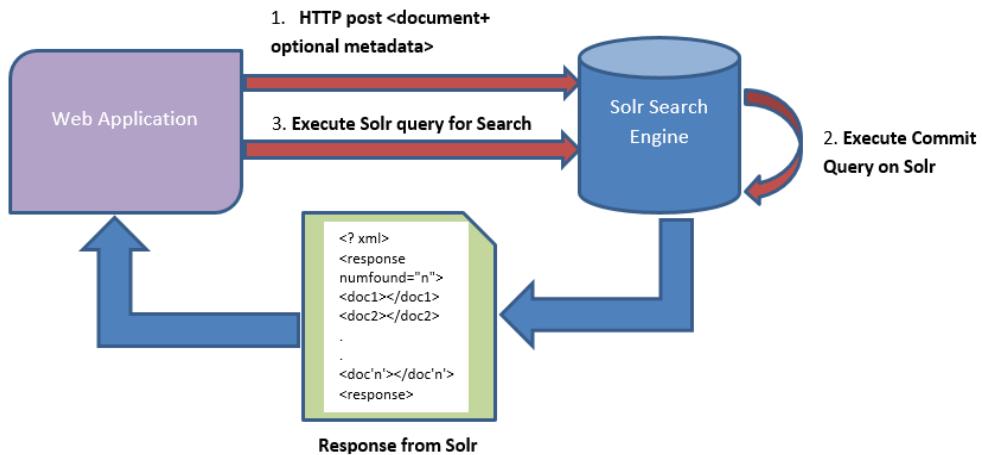
Solr is a popular, powerful, and versatile open-source text search platform developed by the Apache Software Foundation. At its core it is based on Lucene, a search engine written in Java. It works with virtually any file type or source.

There are native clients for many popular languages as noted on the slide.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Solr usage

- Solr uses Lucene Java search library, with XML/HTTP and JSON APIs
- Store documents with optional metadata to Solr for indexing
- Use an HTTP Get to produce an XML, JSON, CSV or binary result



Advanced topics

© Copyright IBM Corporation 2015

Solr usage

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Search procedure

- Solr uses an inverted index that maps tokens to locations
 - Like the index you might find in the back of a textbook
- Documents are analyzed and transformed into a series of tokens
 - The tokens, not the documents themselves, are searched
- Search queries are also analyzed and tokenized
 - Typos are eliminated, word stems are generated, etc
 - Example: runnign → running → run
 - Use the SynonymFilterFactory to provide thesaurus capability
- Results are:
 - Collected
 - Ranked
 - Formatted
 - Grouped / aggregated

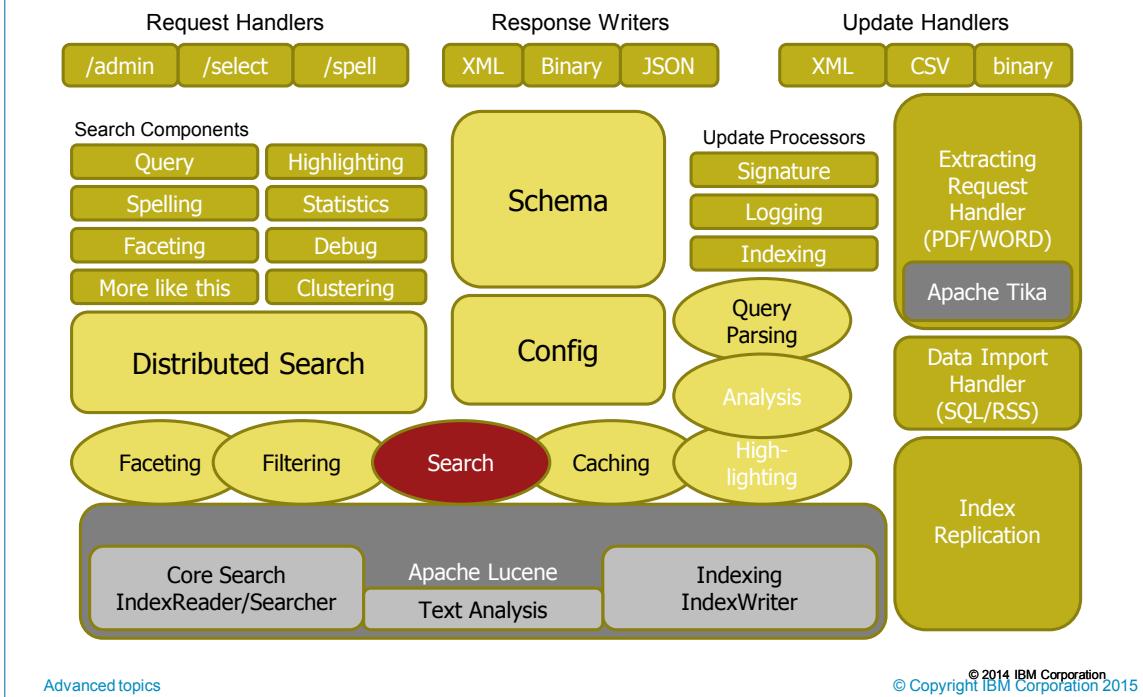
Advanced topics

© 2014 IBM Corporation
© Copyright IBM Corporation 2015

Search procedure

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Overall Solr/Lucene architecture



Overall Solr/Lucene architecture

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Advanced topics © Copyright IBM Corporation 2015

Solr web console

Solr ships with an example server that includes a web console. Through the console you can administer your server, perform queries, view the index, and more.

Copying the example server to your working directory is a very common starting point for your own Solr projects.

The free BigDataUniversity.com course on Solr explores using Solr at greater depth and it uses this Solr web console for some of the exercises:

- <https://bigdatauniversity.com/bdu-wp/bdu-course/introduction-to-solr>

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

Unit summary

- Explain the use for Oozie workflows
- Describe a workflow when applied to a big data processing environment
- List some of the workflow elements
- Understand the importance and use of text search engines in exploring big data
- Understand the Solr search procedure
- Identify Solr components

Advanced topics

© Copyright IBM Corporation 2015

Unit summary

There are no exercises for this unit.

Exercises are available with the Big Data University course mentioned in the notes for this unit.

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE



IBM Training

This material is meant for IBM Academic Initiative use only. NOT FOR RESALE



© Copyright IBM Corporation 2015. All Rights Reserved.