Introduction to big data Analytics using Spark

Belaid Moa, Ph.D.
Compute Canada/WestGrid/University Systems
Email: bmoa@uvic.ca
Office: CLE D071
Agenda

- A slide on Compute Canada
- Review of Bigdata
- Review hands-on
- Mapreduce: why we need spark
- Spark Internals
- Spark Hands
- HBase Internals (not fully covered in the first workshop)
- HBase hands-on
Compute Canada: A national Platform for Advance Research Computing

- Support and experts in advanced research computing
- Many clusters of different nature: Compute Canada resources
  - All over Canada distributed over 4 regions
  - >120K cores.
  - HPC, GPU, PHI
  - Serial, parallel
  - Dynamic Hadoop on HPC clusters
  - Storage facilities
- Visualization and collaboration rooms
- Clouds
  - West and East clouds
Short Review

Mining the beauty of the big data realm...
Introduction to BigData- Characteristics

- Remember: Big data ~ Big confusion
- From “Unreasonable Effectiveness of Data” to “Unreasonable Value of Data”

“Big Data represents the Information assets characterized by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value“

What is big data? A consensual definition and a review of key research topics
BigData Ecosystem - Simplified

Generate Data

Aggregate Data

Analyze Data

Derive Value
BigData Ecosystem Essential Elements

Scalable Resilient FS

- Scale nicely as data grows
- Recover and heal itself after failure

Scalable Resilient PR

- Scale nicely as the data grows
- Recover and restart from failure
Hadoop – The beast, the Train

Hadoop – A Data Operating System

Economical, Scalable, Reliable, Flexible

HDFS

MapRed

Yarn
Infrastructure and platform for Bigdata workshops

- 6 dedicated nodes on hermes
  - 3 x 2TB disks per node
- On the west cloud we have as many VMs
  - Each VM has 8 cores, 15 Gigs and 83GB of ephemeral storage
Infrastructure and platform for Bigdata Workshops

- Software stack

- Dynamic hadoop services:
  - In-house scripting to allow hadoop and its services to run on top of the existing HPC clusters.
Review and Spark Hands-on Guidelines

- Log into your VM
  - `ssh -i .ssh/wg_workshop ubuntu@206.12.59.182`
  - Then ssh to your VM using the ip address you got: `ssh IPAddress`
- Start hadoop using the provided setup-hadoop.sh
- Start jupyter notebook
- Open a new terminal on your local host and do an ssh forwarding
  - `ssh -i .ssh/wg_workshop ubuntu@206.12.59.182 -NL 9995:IPAddress:9995`
- Launch the browser and point it to localhost:9995
Review and Spark Hands-on Guidelines

- Check the configuration files for Hadoop and Spark
- Explore your Hadoop cluster and ecosystem
  - `jps`
  - `hdfs dfsadmin -report`
  - Etc…
- Create a special directory in HDFS under ubuntu user
- Get your favorite text file and ingest it into HDFS.
- Display the first 10 lines
- Count the number of lines in it
Mapreduce

- A computational and programming paradigm designed to work with (key, value) data.
  - Known for many decades, especially in functional languages
  - Fault-tolerant and intuitive abstraction for parallel processing
- Map
  - Take a (key, value) and produce a set of (key, value)s
  - Keys and values can be your usual types: integers, strings, etc.
- Reduce
  - Takes a set of (key, value)s and reduces the values with the same key using a reduction operator
  - Reduction operators can be your favorite sum, multiplication, etc.
- Efficiency target
  - Whenever possible respect the data locality
MapReduce Toy Example

• Map a number to its square

\{(0,1),(0,2), (1,2), (1,3)\} \rightarrow \{(0,1), (0,4), (1,4),(1,9)\}

• Reduce the squares to their sum

\{(0,1), (0,4),(1,4), (1,9)\} \rightarrow \{(0,1+4),(1,4+9)\}
Mapreduce Simplified

https://www.cs.rutgers.edu/~pxk/417/notes/content/mapreduce.html
More on Mapreduce

The user needs just to write the map and the reduce, and the Hadoop processing framework takes care of the rest.

- Implicit step
  - Shuffling and sorting the keys – all the values with the same will land on the same reducer.
  - Default input splitting
- Challenge:
  - Acyclic data flow from persistent storage to persistent storage
  - Reread from persistent storage for every query.
  - Not efficient for applications that reuse the data
    - Iterative and interactive applications
  - Restricted to (key, value) paradigm.
Spark

The powerful swiss knife of big data…
Spark – A Processing Swiss knife for Hadoop

- Takes the mapreduce to the next level
  - Made it efficient, extended it, beautified it and expressed it in different languages
  - Uses memory instead of disks as needed for speed
    - Allowing applications to save working data set into memory
  - Supports multiple languages: Scala, Python and R
- Advanced analytics

- Supports a wide range of applications through transformations and actions paradigm

- Reuses the attractive features of mapreduce including:
  - Scalability, Fault-tolerance, data locality
Spark – A Processing Swiss knife for Hadoop

- Spark Simplified
Spark’s Programming Model

- **RDDs**
  - partitioned collections of elements distributed across the nodes
  - *Cacheable* for efficient reuse
  - Immutable
  - Resilient
  - Manipulated via transformations and actions

- **Transformations**
  - Map-like functions such as map, filter, groupBy, and join
  - Take an RDD as input and produce an RDD as output
  - Lazy evaluation — only builds the DAG

- **Actions**
  - Reduce-like functions, e.g. reduce, count, collect, save.
  - DAGs are executed and results are returned to the driver program
Spark’s Program Workflow

- SparkContext
  - Should be created before start using spark RDDs and Transformations/actions
  - Usually denoted by sc and available in the shell or created by the user
  - We will use findSpark and pyspark context to get sc.

- Creating RDDs
  - Use spark context parallelize to turn a collection to RDD
  - Use textFile and others to turn a file(s) into an RDD

- Apply Transformations
  - See next…
Spark Transformations

- A transformation takes an RDD and returns a new one
- Transformations are lazy-evaluated
- Chaining them allows one to solve problems
- The chain is expressed as a DAG that Spark uses to schedule the tasks
- Spark supports many transformations including
  - Map
    - E.g. apply a function M to each element of an RDD
      \[ \text{newMRDD} = \text{oldRDD}.map(M) \]
  - Filter
  - FlatMap
  - Join
  - Sample
  - Distinct
- The user of anonymous functions is recommended
  \[ \text{newMRDD} = \text{oldRDD}.map(lambda a: a+1) \]
- Key-Value transformations are also available:
  - ReduceByKey
  - groupByKey
  - Etc.
Spark Actions

- An action takes an RDD and returns a value to the driver or write the results into a file
- Actions trigger the execution of the DAG - not lazy
- Spark supports many actions including
  - `reduce`
    - E.g. reduce the elements of an RDD using a function \( R \)
      
    
    \[
    \text{results} = \text{oldRDD}.\text{reduce}(M)
    \]
  - `collect`
  - `take`
  - `count`
  - `saveAsTextFile`
- The user of anonymous functions is recommended
  \[
  \text{results} = \text{oldRDD}.\text{reduce}(\lambda a,b: a+b)
  \]
Spark Task Scheduling

- DAG engine supports and optimizes cyclic data flows, and carry out in-memory cluster computing
- DAG are used to structure the execution of a job and keep the lineage of the RDDs involved
  - In case of failure only the non-available parent RDDs are recursively computed
- DAG structures the job into stages
  - Each stage is a set of independent tasks all computing the same function
  - The split is done at shuffling boundaries
- Spark Scheduler can run stages in the proper order
Spark stack

- Spark Core
  - Resilient Distributed Dataset (RDD)
  - A pipeline of transformations (e.g. map) and actions (e.g. reduce) on RDDs
  - A DAG is used to structure and order stages involved in a job

- Spark Modules
  - Spark SQL
    - SQL-like interface for structure data
  - MLLIB
    - Machine learning library
  - GraphX
    - Library for processing graphs
  - Streaming
    - Library for handling and processing streamed data
Spark’s Deployment Modes

• **Standalone mode**
  - Launch spark’s slaves and the masters on the cluster
  - No resource manager required

• **Cluster mode**
  - Mesos
  - Yarn
    - Cluster: driver running inside a Yarn master application
    - Client: driver running on the client side
Spark Components, yarn-client Mode

Client
Driver Program
Spark Context

Resource Manager

Worker Node
Executor
tasks
Executor
tasks
More on Spark

- **Data Frames**
  - Similar to RDD but for named columns data
  - Very powerful and efficient especially for relational-like operations
  - Very effective when used with Pandas

- **Broadcast Variables**
  - Allow for an efficient sharing of read-only data
  - Broadcasted variables are cached on each node and tasks have access to them
  - Shouldn’t be changed by the tasks – read-only

- **Accumulators**
  - As global variables to use for accumulating values of an associative operation such as +
  - Shouldn’t be read by the tasks
Spark Hands-on Guidelines

- Warm up
  - Get the spark context based on findSpark
  - Sum the numbers from 1,1000.
  - Compute the sum of their squares
  - Get 200 samples and compute their mean and std
  - Sum the odd numbers of the sampled RDD
  - Sum numbers: 0; 0,1; 0,1,2; 0,1,2,3; …

- Estimate the value of pi using a Monte Carlo approach

- Document processing use case
  - Get your favorite text file
  - Ingest it into HDFS
  - Count the number of lines
  - Count the number of words
  - Print the word counts; sort them and display the first 10
  - Print only the lines containing your favorite word
  - Plot the word counts for the first 10 words
  - Use your favorite stop word list to only count important words
  - Plot the word counts of the first 10 words
HBase

Transforming the beast into the beast rabbit ...
HBase – the Beast Rabbit

- A Hadoop database
  - Distributed and Scalable big data store
- NoSQL, No schema, (Key, value), and Column-oriented DB
- A multidimensional map implementing Bigtable.
- Random, low-latency access to big data (billion of rows and million of columns).
- Composed of:
  - HMaster
  - HRegionServers
- Relies on:
  - HDFS – make the beast faster 😊
  - Zookeeper
- Phoenix can be added on top to make it look like SQL DB
HBase components

- HMaster (active and standby)
  - Manages the Hbase cluster by relying mostly on the zookeeper
  - Dictate region assignment and load balancing
  - Carry out administrative tasks, create, delete, etc.
- HRegionServers
  - Handles and serves the region data it is assigned to.
  - The HRegion data is stored in HDFS
  - HRegionServers are co-located with data nodes, which offers data locality opportunities
Tables in HBase

- HTable
  - partitioned into a set of regions
- Each region
  - a contiguous range of sorted row keys with their column families
- Region
  - Store for each column family
    - Memstore – memory store sorts and flushes to Storefiles.
    - StoreFile – the physical representation of Hbase data on disk.
Hbase – Basics

- Interacting with HBase via hbase-shell or sqlline if Phoenix is used
  - Hbase shell can be used to manipulate tables and their content
  - sqlline can be used to run SQL commands
- HBase workflow
  - Manipulate tables
    - Create a table, Drop table, Etc.
  - Manipulate the content of the tables
    - Put, get, scan, delete, etc.
Hbase – Hands-on

- Check the configuration files
- Check the processes running
- `./bin/hbase shell`
  - version
  - Status
  - create ‘books’, ‘Author’, ‘Book’
  - describe ‘book’
  - scan ‘books’
- Case study:
  - Get your favorite CSV file
  - Ingest it to HDFS
  - Do a Hbase/Phoenix bulk load on it