Web and Cloud Computing - Cloud

UBCO Master of Data Science – DATA 534

This week slides

Spart

NoSQL

Lecture Learning Goals

- Describe the operation, implementation, benefits and limitations of distributed hash tables as a way to structure large-scale data and give examples of publicly available DHT services and also publicly available services that depend on DHTs.
- Identify the key differences between peer-to-peer and cloud-based DHTs and give examples of each.
- Identify specific features of the relational model that are difficult to provide at cloud scale and explain why, at an introductory level.
- Explain what a NoSQL database is, how it is similar to and different from both (a) relational databases and (b) distributed hash tables.
- Use a NoSQL database to store and access a (potentially) large, and interestingly complex data collection. Perform complex queries on this data.
- Identify queries that would be interesting to perform that could be performed on a relational database but that can not be performed on a NoSQL database and explain the benefit that NoSQL databases achieve from these limitations.

- Compare and contrast: sequential, parallel, and distributed computation.
- Describe the benefits of parallel computation and the limitations.
- Describe the tradeoffs between shared-memory and non-shared memory parallel programs.
- Explain the term “speedup” and the factors that can limit speedup.
- Explain the term “Big Data” and the ways in which it can benefit from scalable-data and parallel-computation services provided by the cloud.
- Write a simple, massively parallel application and deploy it in the cloud.

Programming languages

Procedural: functions or objects

- procedural: C, Pascal
- Object-oriented: Java, C++, C#, Objective-C,
- Scripting: JavaScript, Python, Perl, PHP

Functional

- Erlang, Haskell, Scala, Clojure

Bid Data Analytics with Spark, Mohammed Guller, Springer 2015

Big Data

Hadoop: Java

- supports Java, Python, C++, Scala, etc.

Spark: Scala

- supports Java, Python, Scala, R
- productivity jump
- write robust code
- Spark is written in Scala

Functional Programming

use functions as a building block

avoid mutable variables, loops, and other imperative control structures

treats computation as evaluation of mathematical functions

functions are first-class citizens

Functional Programming

mainstream languages have added support for functional programming

- C++, Java, Python

Pros

1. tremendous boost to programmer productivity
   - use fewer lines of code than imperative language
   - 100 LOC Java may need 10-20 LOC Scala
2. easier to write concurrent or multithreaded applications
3. helps you write robust code. Less LOC means less bugs
4. write elegant code that is easy to read

Functions

No side effects

- result of a function only depends on the input argument
- behavior does not change with time
- i.e. a function does not have a state
- does not depend or update any global variable
- benefits:
  - can be composed in any order
  - easy to reason about the code
  - easier to write multi-threaded applications

Immutable data structures

purely functional program does not use any mutable data structure or variable

i.e. data is never modified in place unlike C/C++, Java, Python

Benefits

- reduce bugs
- easy to reason about code
- functional languages provide constructs that allow a compiler to enforce immutability: may bugs caught at compile time
- easier to write multi-threaded code
  - avoid issues with race conditions and data corruption

Scala fundamentals

Scala is a hybrid programming language: supports

- object-oriented programming
  - class, object, trait
  - encapsulation, inheritance, polymorphism
- functional programming
  - immutable data structures
  - functions as first-class citizens

emphasizes functional programming

statically typed language

- compiled by Scala compiler
- type-safe language; compiler enforces type safety at compile time

Java virtual machine (JVM)-base language

- Scala compiler compiles Scala application into Java bytecode that runs on any JVM
- at bytecode level, Scala is indistinguishable from a Java application
- seamlessly interoperable with Java: Scala library can be used in a Java library; benefit from vast Java libraries

Apache Spark

A new general framework, which solves many of the short comings of MapReduce

Is capable of leveraging the Hadoop ecosystem, e.g. HDFS, YARN, HBase, S3, …

Has many other workflows, i.e. join, filter, flatMapdistinct, groupByKey, reduceByKey, sortByKey, collect, count, first…

- (around 30 efficient distributed operations)

In-memory caching of data (for iterative, graph, and machine learning algorithms, etc.)

Native Scala, Java, Python, and R support

Supports interactive shells for exploratory data analysis

Spark API is extremely simple to use

Developed at AMPLab UC Berkeley, now by Databricks.com

Keywords

HDFS: Hadoop Distributed File System

YARN:resource management and job scheduling technology in the open source Hadoop distributed processing framework

HBase: open-source non-relational distributed database

S3: storage system provided by Amazon

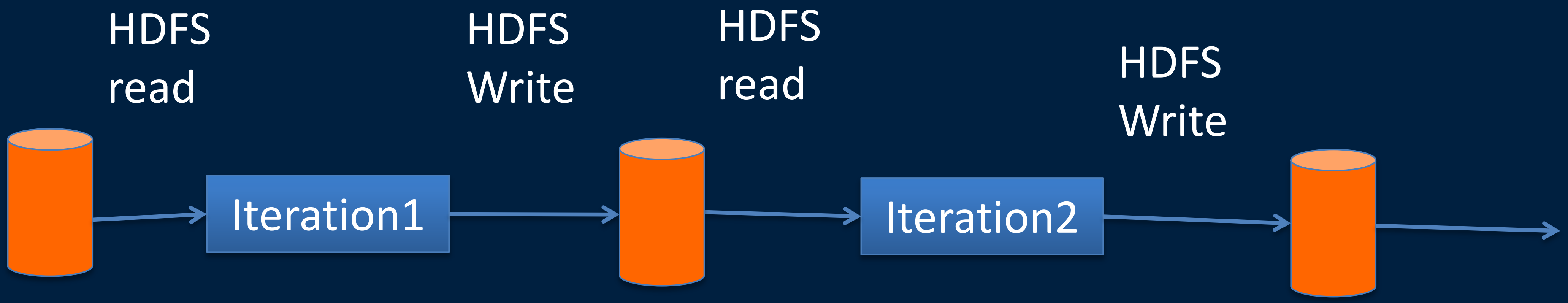
EMR: Amazon Elastic MapReduce

RDD: Resilient Distributed Datasets. Represents an immutable, partitioned collection of elements that can be operated on in parallel.

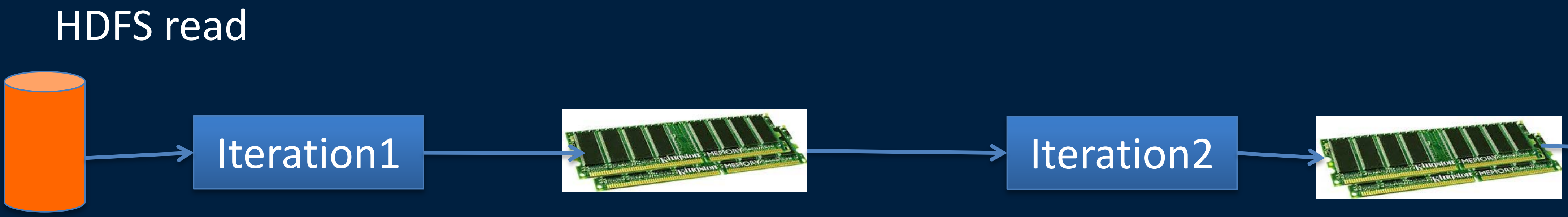
RDMS: Relational Database Management System

Spark Uses Memory instead of Disk

Hadoop: Use Disk for Data Sharing



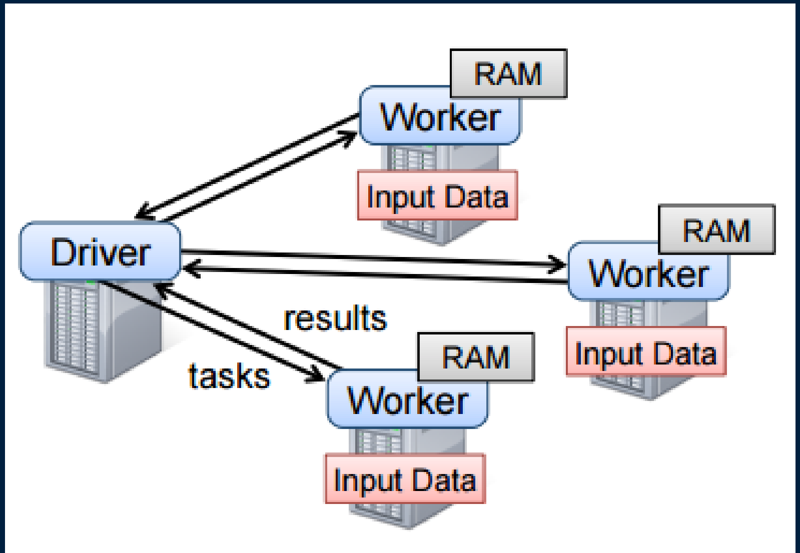
Spark: In-Memory Data Sharing



Spark on EMR Architecture

driver program

- program that performs operations on
- data stored on multiple machines in a cluster
- typically data organized in *memory* of cluster nodes



cluster

- nodes
- data partitions

data abstractions

- resilient distribute dataset (RDD)
- partitioning of data for parallel operation
- shared variables
- variables that can be shared across partitions during parallel executions tasks
- subdivision of execution running on individual cluster nodes

Spark Programming Languages

R

- domain-specific language for statistical computing
- based on S with lexical scoping similar to Scheme

Python

- dynamically typed

Java

- statically typed

Scala

- functional language (like Scheme, ML, and Haskell)
- static types with type inference
- object-oriented
- compiles to java byte code language (executed by java vm)

Resilient Distributed Datasets (RDD)

Parallelized Collections

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data, P)
```

Transformations

- create new dataset by transforming existing dataset (lazily)
- e.g., map

Actions

- perform computation on dataset and return resulting value
- e.g., like reduce

Transformations

Spark partitions RDD data across multiple cluster nodes

- key/value pairs

Spark sends transformations to each cluster node

Types of Transformations

- apply function to each element of dataset (or each partition)
  - map, filter, …
- combine 2 datasets by key
  - union, intersection, reduceByKey, join
- repartition
  - shuffle data among partitions to change # of partitions, rebalance partitions, etc

Actions

reduce

- aggregate elements using f; f must be associative and commutative
- runs in parallel on each partition

enumerate

- collect, count, first, takeSample, forEach

Spark Example

```
import sys
from random import random
from operator import add

from pyspark import SparkContext


if __name__ == "__main__":
    """
        Usage: pi [partitions]
    """
    sc = SparkContext(appName="PythonPi")
    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
    n = 100000 * partitions

    def f(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 < 1 else 0

    count = sc.parallelize(xrange(1, n + 1), partitions).map(f).reduce(add)
    print "Pi is roughly %f" % (4.0 * count / n)

    sc.stop()
```

Variables

You write one program that runs on many nodes

- what's the problem when it comes to variables

Consider this Spark / Python code

- why is this broken?

```python
counter = 0
rdd = sc.parallelize(data)

def increment_counter(x):
    global counter
    counter += x
rdd.foreach(increment_counter)

print("Counter value: ", counter)
```

Shared Variables

Broadcast variables

- read-only variable whose value is broadcast to every node

Accumulators

- add-to-only variable (associative and commutative)
  - numeric types are built-in, but other types can be added by program
- any task can update variable
- only driver task and read the variable
- used for reduce, for example

Corrected Example

Wrong

```
counter = 0
rdd = sc.parallelize(data)

def increment_counter(x):
    global counter
    counter += x
rdd.foreach(increment_counter)

print("Counter value: ", counter)
```

Correct

```
counter = sc.accumulator(0)
rdd = sc.parallelize(data)

def increment_counter(x):
    counter.add(x)
rdd.foreach(increment_counter)

print("Counter value: ", counter.value)
```

Digging Deeper into Spark

RDD Programming Guide

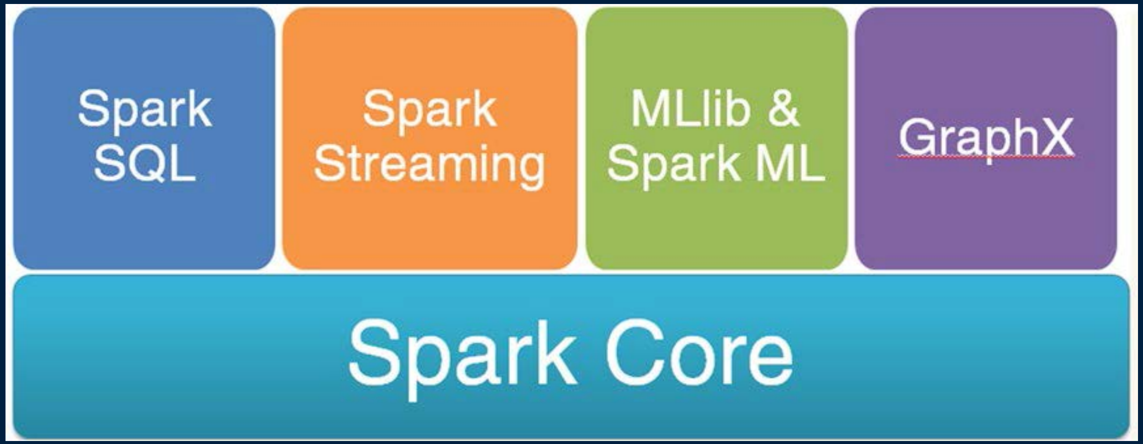
- https://spark.apache.org/docs/latest/rdd-programming-guide.html

Examples

- https://github.com/apache/spark/tree/master/examples/src/main/python

Parameters	Spark	Hadoop
Data Storage	Spark stores data in-memory.	Hadoop stores data on disk.
Fault tolerance	Spark's data storage model, resilient distributed datasets (RDD) guarantees fault tolerance.	It uses replication to achieve fault tolerance.
Line of code	Apache Spark is project of 20,000 Line of code.	Hadoop 2.0 has 1,20,000 Line of code
Speed	It is Faster due to In-memory computation.	It is relatively slower than Spark.
OS Support	• Linux • Windows • Mac OS	• Linux
High level language	• Scala • Python • Java • R	• Java
Streaming data	Spark can be used to process as well as modify real-time data with Spark streaming.	With Hadoop Map-Reduce one can process batch of stored data.
Machine Learning	Spark has its own set of Machine learning libraries (MLib).	Hadoop requires interface with other Machine learning library. Eg: Apache Mahout.

Menu



Bid Data Analytics with Spark, Mohammed Guller, Springer 2015
http://link.springer.com/book/10.1007%2F978-1-4842-0964-6

MLlib and Spark ML

Machine Learning

MLlib

regression and classification

- linear regression
- logistic regression
- SVM
- naïve Bayes
- decision tree
- random forest
- gradient-boosted trees
- isotonic regression

Clustering

- K-means
- streaming k-mean
- Gaussian misture
- Power iteration clustering
- Latent Dirichlet allocation

MLlib

Dimensionality reduction

- PCA
- SVD

Feature extraction and transformation

- TF-IDF
- Word2Vec
- Standard Scaler
- Normalizer
- Chi-Squared feature selection
- elementwise product

frequent pattern mining

- FP-growth
- association rules
- prefixSpan

Recommendation

- Collaborative filtering with Alternating Least Squares

Spark ML

newer library

higher-level abstraction

share many classes and method names

easier to assemble ML steps

uses DataFrame as primary data abstraction

key abstractions

- transformer, estimator, pipeline, parameter grid, crossValidator, evaluator

NoSQL Databases (vs RDMS)

General term that includes Key-Value, Document, and Graph (Object) Databases

Some of the Main Players

- MongoDB, Casandra, DynamoDB

Collection, Document, Field vs Table, Row, Column

- collection is roughly a table … document a row
- DynamoDB uses terms: table and item for collection and document

Dynamic Scheme vs Fixed

- document format typically in JSON or BSON (binary JSON)

CRUD vs Query Language (e.g., SQL)

- create, read, update and delete are operations collections or documents

No Join

- join doesn’t scale
- can be avoided through de-normalization (and sometimes by linking)

Maybe no other things

- transactions are one example

MySQL vs MongoDB

<table>
<tr><th>MySQL</th><th>MongoDB</th></tr>
<tr><td><pre>INSERT INTO users (user_id, age, status)
VALUES ('bcd001', 45, 'A')</pre></td><td><pre>db.users.insert({
  user_id: 'bcd001',
  age: 45,
  status: 'A'
})</pre></td></tr>
<tr><td><pre>SELECT * FROM users</pre></td><td><pre>db.users.find()</pre></td></tr>
<tr><td><pre>UPDATE users SET status = 'C'
WHERE age > 25</pre></td><td><pre>db.users.update(
  { age: { $gt: 25 } },
  { $set: { status: 'C' } },
  { multi: true }
)</pre></td></tr>
</table>

DynamoDB

Massively Scalable

- deployed in Amazon’s cloud (AWS) framework

Table

- has a name, keys and a set of items
- schema specified when table is created names its key(s)
- hash key and range key
- hash key is globally unique and unordered; range key is unique wrt hash key and index is sorted by range key

Item

- a collection of property, value pairs, including values for keys
- items can not be nested as in JSON (no de-normalization)

Access

- read, update and delete items by their key value (hash) range (range)

Boto3 (Python API)

- http://boto3.readthedocs.io/en/latest/guide/dynamodb.html

DynamoDB Examples

```python
table.put_item(
   Item={
        'username': 'janedoe',
        'first_name': 'Jane',
        'last_name': 'Doe',
        'age': 25,
        'account_type': 'standard_user',
    }
)
```

```python
table.update_item(
    Key={
        'username': 'janedoe',
        'last_name': 'Doe'
    },
    UpdateExpression='SET age = :val1',
    ExpressionAttributeValues={
        ':val1': 26
    }
)
```

Amazon training services

https://www.youtube.com/watch?v=2mVR_Qgx_RU


THE UNIVERSITY OF BRITISH COLUMBIA