Aalto University
School of Science

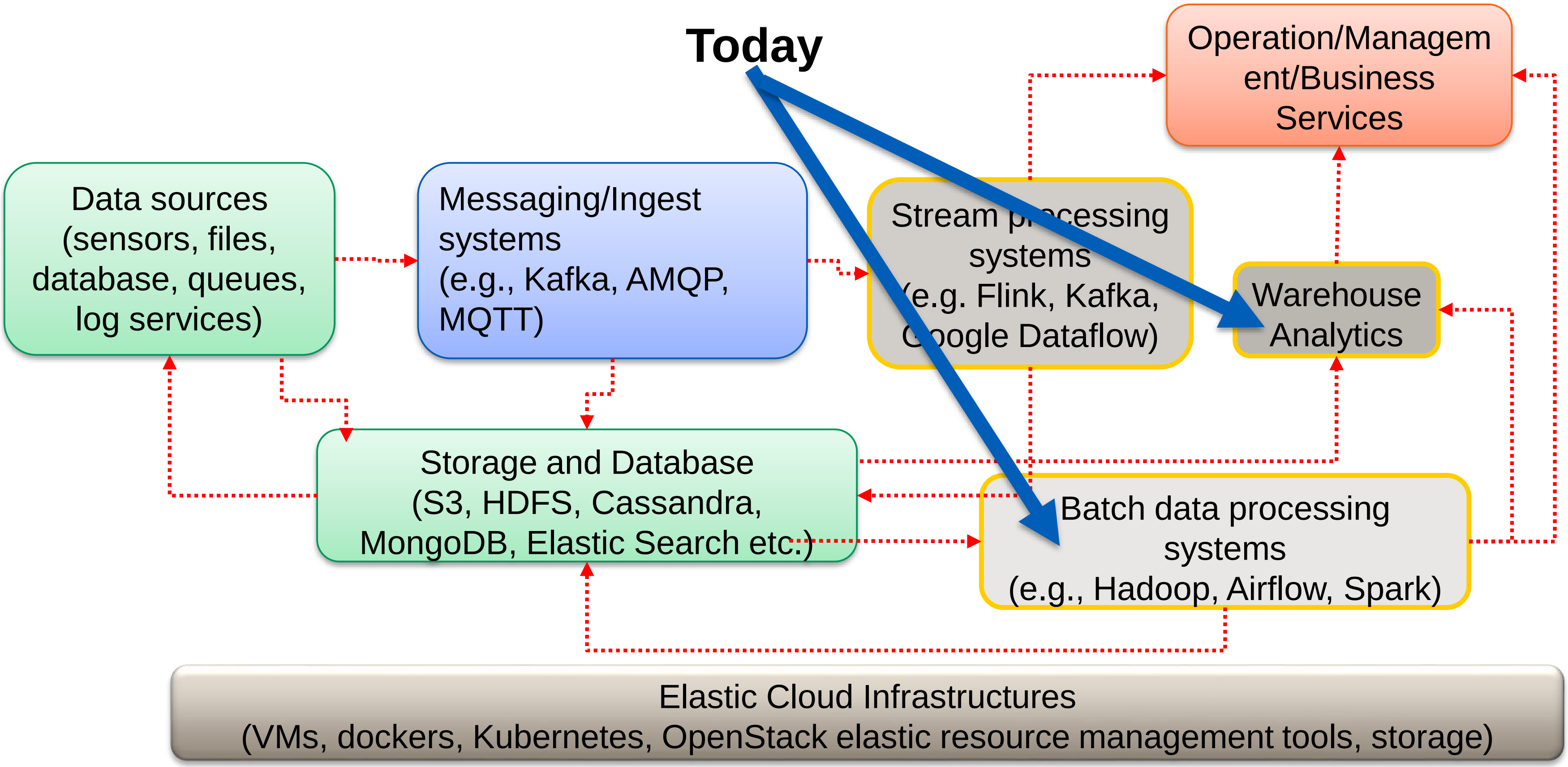
Big Data Processing with MapReduce/Spark Programming Models

Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi, *https://rdsea.github.io*

Schedule

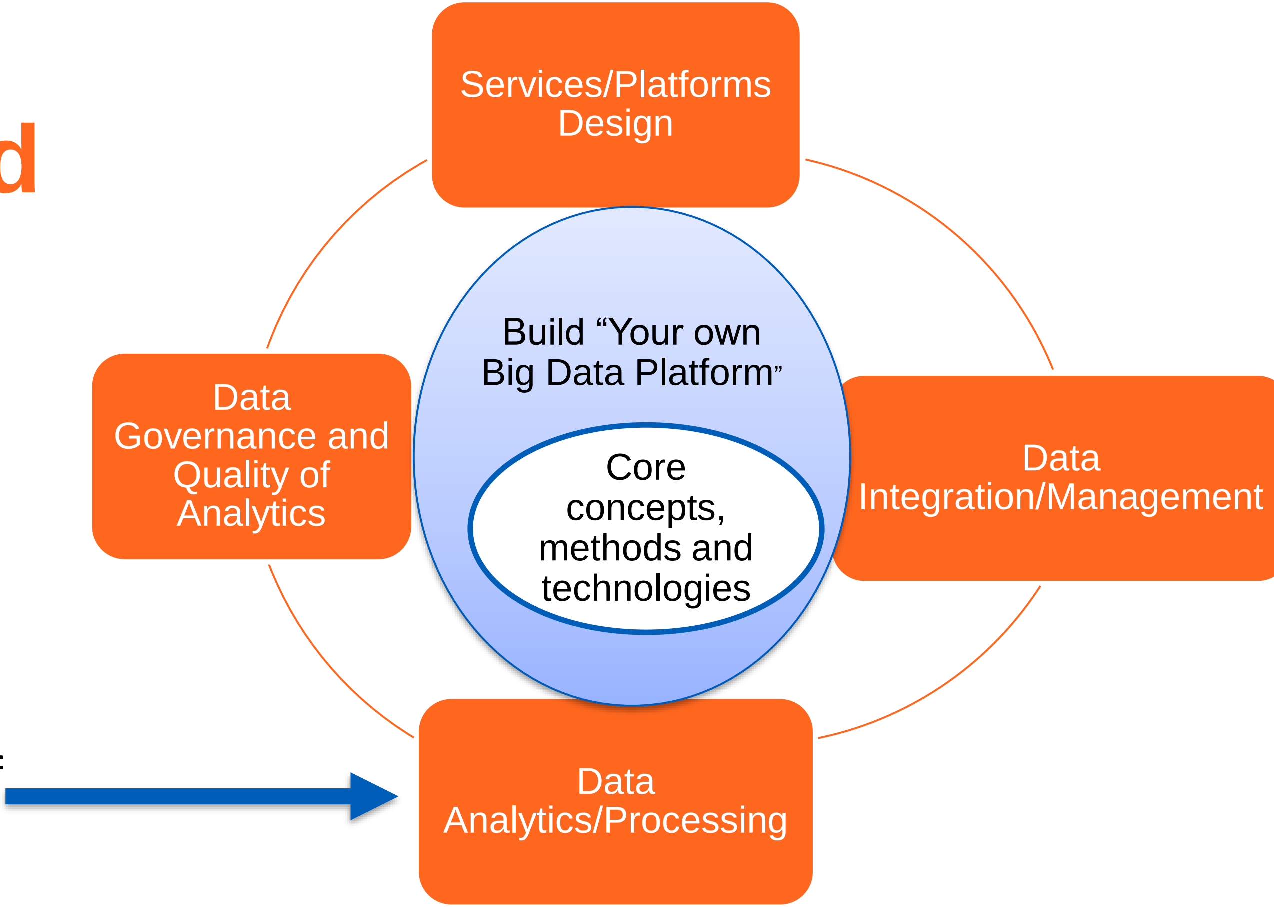
- Overview
- MapReduce programming model
- Apache Spark
- Summary

Big data at large-scale

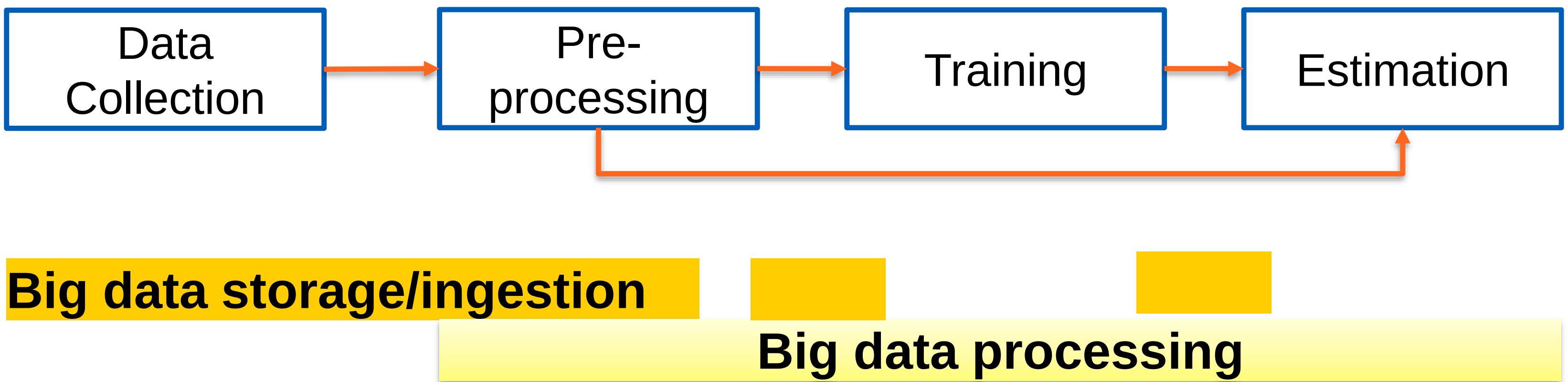


Recall: build your story

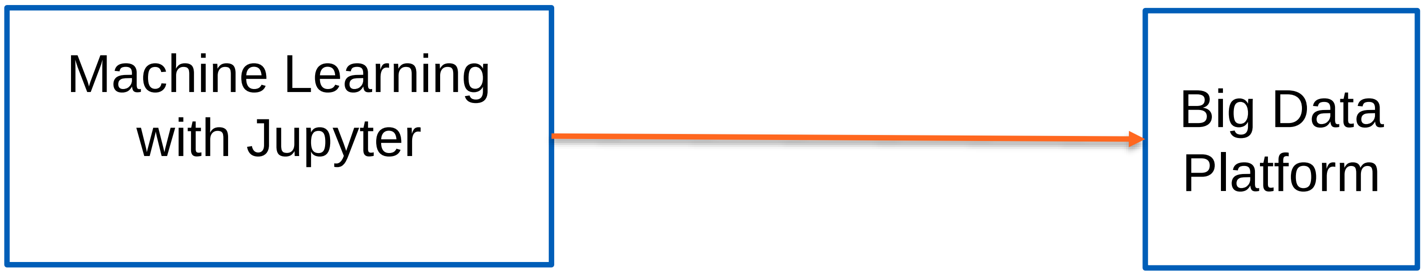
Play the roles of developer/users and of platform provider



Example of a typical big data machine learning pipeline (hot area!)



Big data processing in our story: we are not just “data scientist”



User (scientists, engineers)

Provider

Our learning goals: tasks in systems and application

- **Understand the user/developer needs**
- **Understand how to build platforms to support the users/developers**

Big data processing techniques in our focus

- **Programming models**
  - MapReduce
  - Streaming Processing
  - Workflows/Pipelines
- **Studied frameworks**
  - Apache Hadoop/Spark
  - Apache Flink
  - Apache Airflow

MapReduce programming model

- **MapReduce is a programming model orignal from from Google**
  - Various implementations/frameworks support MapReduce
  - Apache Hadoop (originally from Yahoo!) is the most famous one
- **Support batch data processing for very large datasets**
- **Suitable for batch jobs in big data, e.g.,**
  - Web search, document processing, ecommerce information
  - Extract, transform, data wrangling, and data cleansing tasks

MapReduce

https://hadoop.apache.org

Example of a real data

Look at the network monitoring data

PROVINCECODE,DEVICEID,IFINDEX,FRAME,SLOT,PORT,ONUINDEX,ONUID,TIME,SPEEDIN,SPEEDOUT
XXN,10XXXXXX023,26XXXXXX8,1,2,7,39,100XXXXXX2310207039,01/08/2019 00:04:07,148163,49018
XXN,10XXXXXX023,26XXXXXX8,1,2,7,38,100XXXXXX2310207038,01/08/2019 00:04:07,1658,1362
XXN,10XXXXXX023,26XXXXXX8,1,2,7,9,100XXXXXX2310207009,01/08/2019 00:04:07,6693,5185

Understand the MapReduce programming model

ONUID SPEEDIN

Key	Value
…	879656
…	65252
…	…
….	103831
…	1045755
…	

Map

Key	Value
7039	879656
7038	65252
…	

Key	Value
7039	103831
7038	1045755

Key	Value
7038	65252
7039	879656
…	…
7039	103831
7038	1045755
…	

Reduce

Key	Value
7039	148163
7038	1658
…	…
…	…
…	…
…	

Parallelizing data **Sort/Shuffle** **Group/Aggregation**

Map & Reduce

- **Map:**
  - Receives <key,value>
  - Outputs <key,value> - new set of <key,value>
- **Reducer:**
  - Receives <key, Iterable[value]>
  - Outputs <key,value>

Key ideas of MapReduce

- **Data can be divided by “Map” operators**
  - data processing tasks extract “intermediate results”
- **Intermediate results can be aggregated through “Reduce” operators**
  - data processing tasks produce a result from “intermediate results”
- **We can glue “Map” and “Reduce” operators into a multi-stage data flow model**
- **Other possible operators:**
  - Combiner: performs “Reduce” at local nodes
  - Partitioner: decides key/value for Reduce

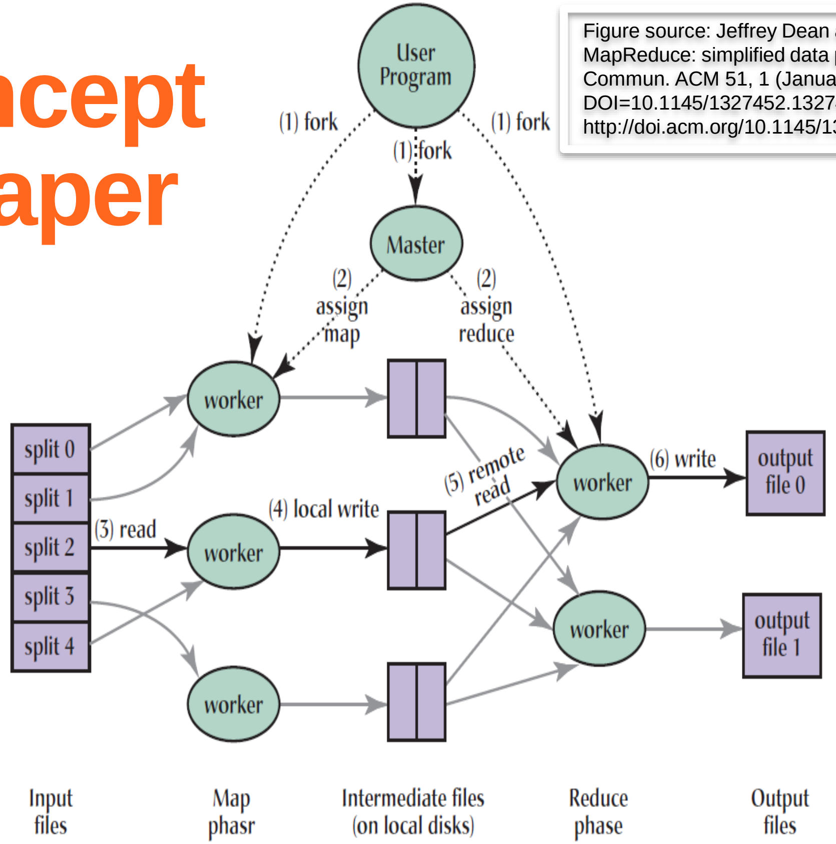
Key ideas of MapReduce

- **Key points for the developers:**
  - should write only the main “logic”: Map and Reduce operators
- **The runtime framework will**
  - handle data movement and input/output management for Map/Reduce tasks
  - parallelizing tasks in multiple nodes

MapReduce concept in the original paper

Figure source: Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113. DOI=10.1145/1327452.1327492 http://doi.acm.org/10.1145/1327452.1327492

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```



Key point: parallelize workers to process a lot of input files and produce a lot of output files

Hadoop MapReduce

- **Hadoop supports the MapReduce programming model**
  - Use nodes for computing tasks
  - Access data in HDFS and data partitioned in different nodes
  - Hadoop runtime automatically creates parallel tasks
  - YARN is used to run jobs
- **Data management (HDFS) and data processing (MapReduce) is aligned nicely**
  - Run in the same nodes → data locality optimization

Examples - Map

Input

Output

```
public static class SpeedInMapper
      extends Mapper<Object, Text, LongWritable , AverageWritable>{
  private LongWritable id =new LongWritable();
  private AverageWritable averagecount = new AverageWritable();
  public void map(Object key, Text value, Context output)
        throws IOException, InterruptedException {

        String valueString = value.toString();
        String[] record = valueString.split(",");
        id.set(Long.parseLong(record[7]));
        averagecount.setAverage(Float.parseFloat(record[9]));
        averagecount.setCount(1);
        output.write(id,averagecount);
  }
}
```

Parse the data to get ONUID and SPEEDIN

Map (ONUID,(SPEEDIN, count))

Example - Reduce

Input

Output

```
public static class SpeedInAverageReducer
     extends Reducer<LongWritable,AverageWritable,LongWritable,FloatWritable> {
  private FloatWritable new_result = new FloatWritable();


  public void reduce(LongWritable key, Iterable<AverageWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    float avg = 0;
    int count = 0;
    for (AverageWritable val : values) {
        float current_avg =val.getAverage();
        int current_count =val.getCount();
        avg = avg + (current_avg*current_count);
        count = count + current_count;
    }

   new_result.set(avg/count);
   context.write(key, new_result);
  }
}
```

Simple way to determine the average as "Reduce" operator

Reduce (ONUID,AVG)

Driver: connect Map and Reduce operators

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "simpleonuaverage");
  job.setJarByClass(SimpleAverage.class);
  job.setMapperClass(SpeedInMapper.class);
  job.setCombinerClass(SpeedInAverageCombiner.class);
  job.setReducerClass(SpeedInAverageReducer.class);
  job.setMapOutputKeyClass(LongWritable.class);
  job.setMapOutputValueClass(AverageWritable.class);
  job.setOutputKeyClass(LongWritable.class);
  job.setOutputValueClass(FloatWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Combiner

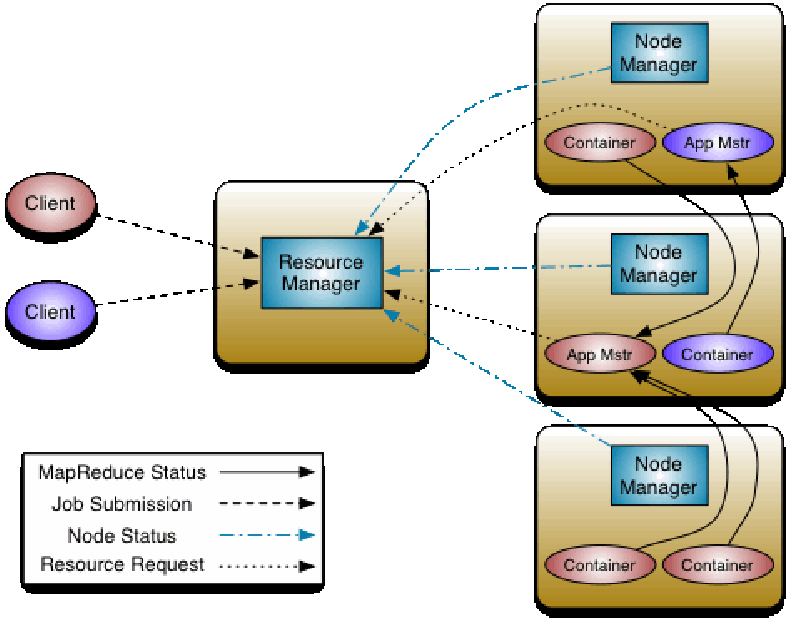
Example with Python using MRJob

```
class ONUSpeedinAverage(MRJob):
    def mapper(self, _, entry):
        provincecode,deviceid,ifindex,frame,slot,port,onuindex,onuid,timestamp,speedin,speedout= entry.split(",")
        #average speed its speedin with count = 1
        yield (onuid, (float(speedin),1))

   ## recalculate the new speedin average through an array of speedin average values
    def _recalculate_avg(self, onuid, speedin_avg_values):
        current_speedin_total = 0
        new_avg_count = 0
        for speedin_avg, avg_count  in speedin_avg_values:
            current_speedin_total = current_speedin_total +(speedin_avg*avg_count)
            new_avg_count = new_avg_count + avg_count
        new_speedin_avg = current_speedin_total/new_avg_count
        return (onuid, (new_speedin_avg, new_avg_count))

    def combiner(self, onuid, speedin_avg_values):
        yield self._recalculate_avg(onuid, speedin_avg_values)

    def reducer(self, onuid,speedin_avg_values):
        onuid, (speedin_avg, avg_count) = self._recalculate_avg(onuid,speedin_avg_values)
        yield (onuid, speedin_avg)

if __name__ == '__main__':
    ONUSpeedinAverage.run()
```

Resource management and execution for MapReduce in clusters

A cluster of computing nodes can be managed by YARN

or Mesos



Source: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Schedule and monitoring for MapReduce

- **A MapReduce program runs → MapReduce Job**
  - includes many tasks (Map and Reduce processes + others)
- **JobTracker**
  - monitors the whole job (all tasks of a MapReduce program)
- **TaskTracker**
  - performs a task of the MapReduce applications
  - informs JobTracker about the state of the tasks

Monitoring MapReduce Jobs

Logged in as: dr.who

Application application_1570429323498_0008

- Cluster
  - About
  - Nodes
  - Node Labels
  - Applications
    - NEW
    - NEW_SAVING
    - SUBMITTED
    - ACCEPTED
    - RUNNING
    - FINISHED
    - FAILED
    - KILLED
  - Scheduler
- Tools

Application Overview

User:	mybdp
Name:	cse4640-nytaxicount
Application Type:	SPARK
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	FINISHED
Queue:	default
FinalStatus Reported by AM:	SUCCEEDED
Started:	Fri Oct 25 19:22:08 +0000 2019
Elapsed:	3mins, 6sec
Tracking URL:	History
Log Aggregation Status:	DISABLED
Application Timeout (Remaining Time):	Unlimited
Diagnostics:	
Unmanaged Application:	false
Application Node Label expression:	<Not set>
AM container Node Label expression:	<DEFAULT_PARTITION>

Application Metrics

Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	5039065 MB-seconds, 973 vcore-seconds
Aggregate Preempted Resource Allocation:	0 MB-seconds, 0 vcore-seconds

Show 20 entries Search:

Attempt ID	Started	Node	Logs	Nodes blacklisted by the app	Nodes blacklisted by the system
appattempt_1570429323498_0008_000001	Fri Oct 25 22:22:08 +0300 2019	http://cluster-bdp-w-3.c.bigmultidatstore.internal:8042	Logs	0	0

Showing 1 to 1 of 1 entries First Previous 1 Next Last

Connecting MapReduce applications

Build complex MapReduce pipelines

HDFS, Hive, …

Using big data storage/database as data exchange

We can use workflows to coordinate different MapReduce apps

Problems with MapReduce

- **Strict Map & Reduce tasks connection → limitation**
- **Need more flexible in processing big data workloads**
  - Batch data flows and streaming data flows
- **Programming diversity support**
  - Software engineering productivity

Apache Spark

https://spark.apache.org/

Apache Spark

- **Cluster-based high-level computing framework**
- **"unified engine" for different types of big data processing**
  - SQL/structured data processing
  - Machine learning
  - Graph processing
  - (Microbatching)streaming processing
- **It is a powerful computing framework and system → an important service that a big data platform should support**

Apache Spark

Can be run a top

- **Hadoop (using HDFS and YARN)**
- **Mesos cluster**
  - http://mesos.apache.org/
- **Kubernetes**
- **Standalone machines**

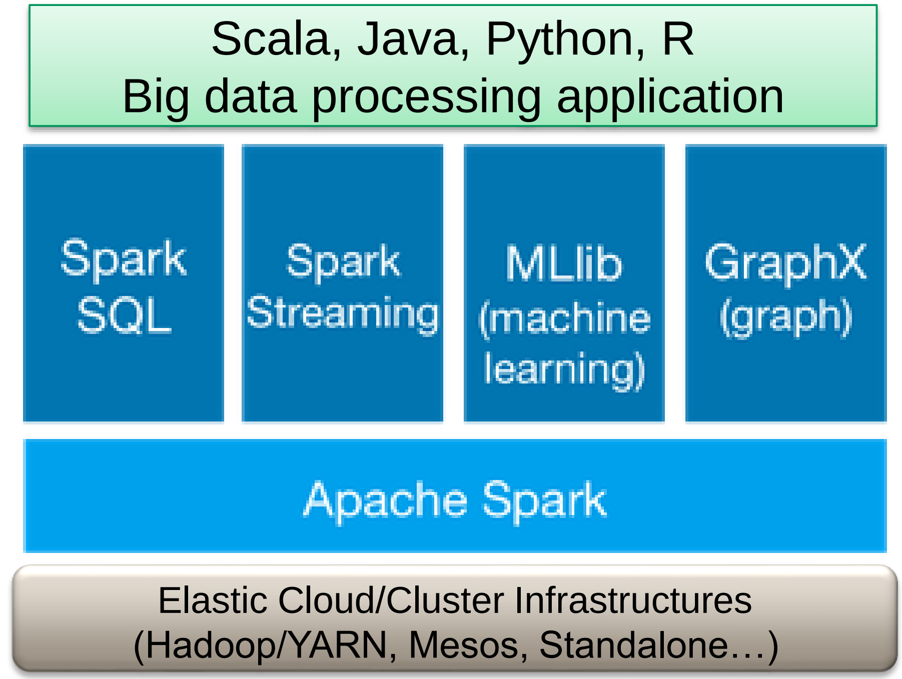


Figure source: http://spark.apache.org/

Key features

- **Data is distributed in different nodes for processing**
  - Like data distributed in different nodes in big storage/database
- **Leverage parallel computing concepts to run multiple tasks**
  - Parallel tasks, task pipeline, DAG of processing stages
- **Program driver steers the execution of parallel tasks**
  - Tasks are paralleled automatically and are scheduled with different underlying schedulers
- **Key data operators**
  - Transformations and actions on data

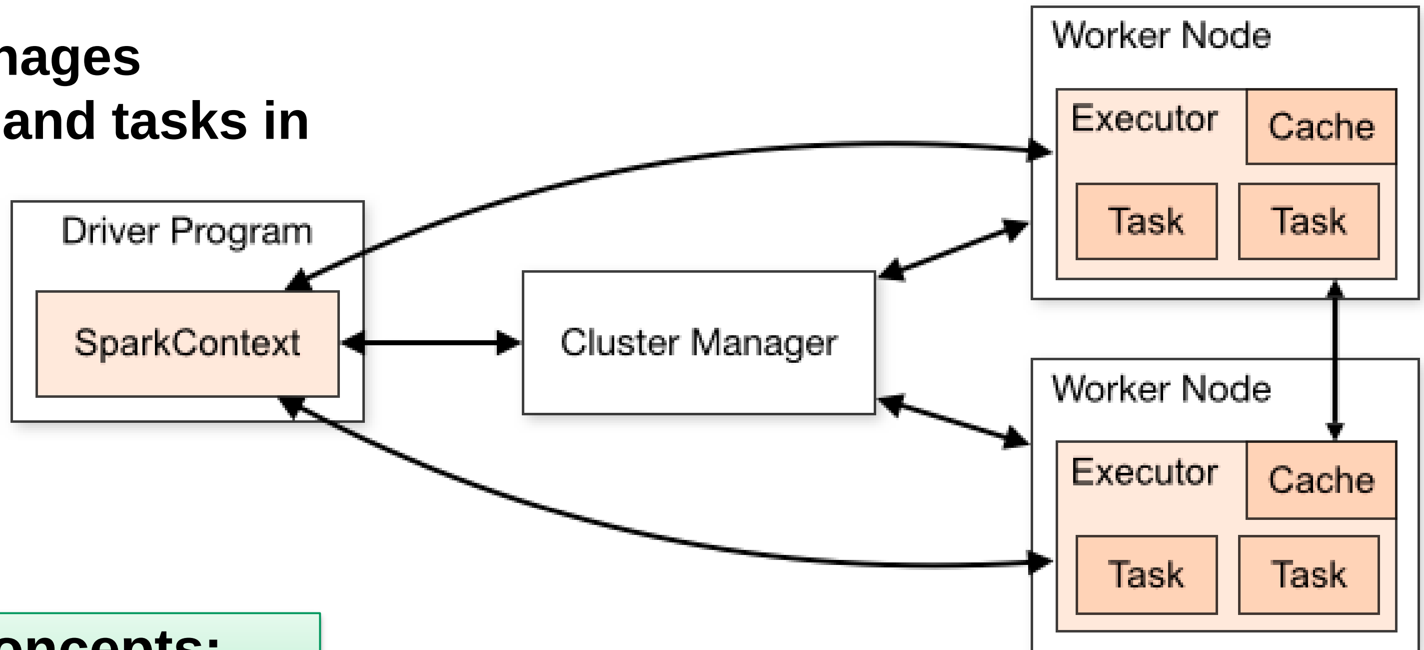
Spark Program: programming elements

- **SparkSession**
  - Act as a program driver to manage the execution of tasks
  - SparkContext: manages connection to cluster, manage internal services
- **Data APIs**
  - Resilient Distributed Dataset (RDD)/DataFrames/DataSets
  - Load and hold distributed data
  - Transformation and action functions
- **ML, Graph and Streaming functions and pipelines**

Execution Model

Map into a resource in a cluster node

Driver manages operators and tasks in nodes



Common concepts: Driver, Nodes, Tasks

Figure source: http://spark.apache.org/docs/latest/cluster-overview.html

Spark application management: high-level view

- **Submission/Request**
  - Submit the Spark application for running
  - Resource is provided for running the Driver
- **Launch**
  - The Driver requests resources for executors (through SparkContext)
  - Establish executors across worker nodes
- **Execution**
  - The driver starts to execute code and move data
- **Finish/Completion:**
  - Finish, release executors

Spark program logic: typical steps

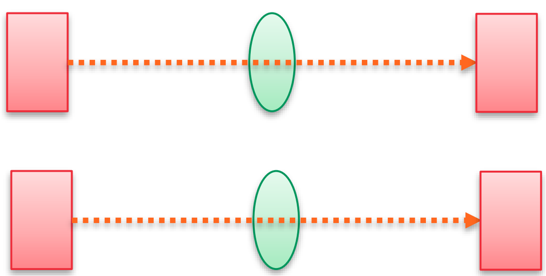
- **Load data and distribute data**
  - Data is immutable after created
  - Data partition in Spark: a partition is allocated in a node
- **Perform transformation and action operators**
  - Transformations: build plans for transforming data models
  - Actions: perform computation on data
- **The developer mostly focuses on loading data and performing operators**

Resilient distributed dataset (RDD)

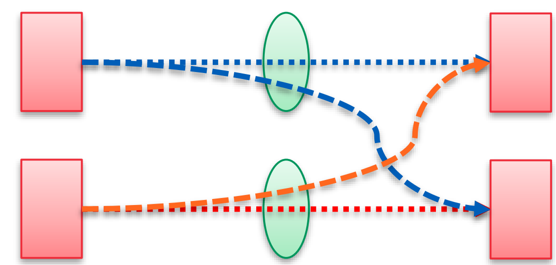
- **Low-level data structure**
  - Collection of data elements partitioned across nodes in the cluster
- **Create RDD**
  - Created by loading data from files (text, sequence file) including your local file system, HDFS, Cassandra, HBase, Amazon S3, etc.
- **Persist RDD**
  - In memory or to files

Transformation operators

- **Transformations:**
  - Instructions about how to modify a RDD to have another RDD
  - Only tell what to do: to build a DAG (direct acyclic graph) of RDDs
  - lazy approach → doing at action operators



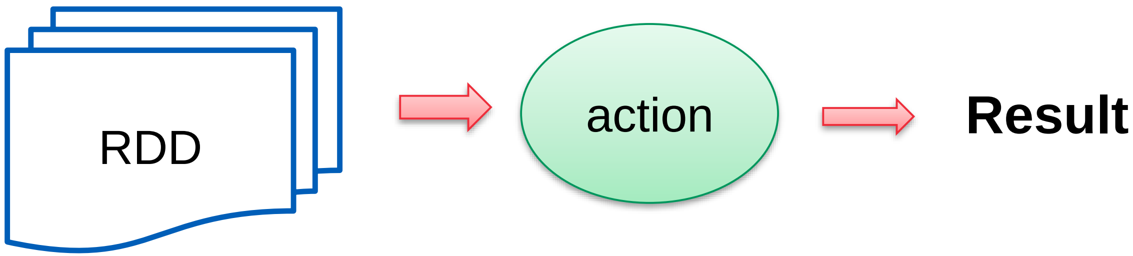
Narrow transformation, no data shuffle



Wide transformation, cross data partitions, require shuffles

Action operators

- **Compute the results for a set of transformations**
  - Examples: count or average
- **Actions: view, collect, write, calculation**



RDD transformations and actions

Transformations

- map
- filter
- sample
- intersection
- groupByKey

Actions

- reduce()
- collect()
- count()
- saveAs…File()

Example with RDD

VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fwd_flag,PULocationID,DOLocationID,payment_type,fare_amount,extra,mta_tax,tip_amount,tolls_amount,improvement_surcharge,total_amount
2,11/04/2084 12:32:24 PM,11/04/2084 12:47:41 PM,1,1.34,1,N,238,236,2,10,0,0.5,0,0,0.3,10.8
2,11/04/2084 12:32:24 PM,11/04/2084 12:47:41 PM,1,1.34,1,N,238,236,2,10,0,0.5,0,0,0.3,10.8
2,11/04/2084 12:25:53 PM,11/04/2084 12:29:00 PM,1,0.32,1,N,238,238,2,4,0,0.5,0,0,0.3,4.8

as a text file

```
conf = SparkConf().setAppName("cse4640-rddshow").setMaster(args.master)
sc = SparkContext(conf=conf)
##modify the input data
rdd=sc.textFile(args.input_file)
## if there is a header we can filter it otherwise comment two lines
csvheader = rdd.first()
rdd =  rdd.filter(lambda csventry: csventry != csvheader)
## using map to parse csv text entry
rdd=rdd.map(lambda csventry: csventry.split(","))
rdd.repartition(1)
rdd.saveAsTextFile(args.output_dir)
```

Example with RDD

Output: RDD with id index for columns

```
['2', '03/21/2019 04:24:52 PM', '03/21/2019 04:47:01 PM', '5', '4.49', '1', 'N', '236', '113', '1', '18', '0.5', '0.5', '3.86', '0', '0.3', '23.16']
['2', '03/21/2019 04:10:59 PM', '03/21/2019 04:19:22 PM', '5', '1.95', '1', 'N', '236', '75', '2', '8.5', '0.5', '0.5', '0', '0', '0.3', '9.8']
['2', '03/17/2019 01:38:09 AM', '03/17/2019 02:00:03 AM', '1', '4.24', '1', 'N', '170', '238', '1', '17.5', '0.5', '0.5', '4.7', '0', '0.3', '23.5']
['2', '03/17/2019 01:26:07 AM', '03/17/2019 01:30:28 AM', '1', '0.7', '1', 'N', '113', '234', '2', '5', '0.5', '0.5', '0', '0', '0.3', '6.3']
['2', '03/17/2019 01:13:50 AM', '03/17/2019 01:19:25 AM', '1', '1.22', '1', 'N', '113', '249', '1', '6', '0.5', '0.5', '1.46', '0', '0.3', '8.76']
['2', '03/17/2019 01:04:30 AM', '03/17/2019 01:10:43 AM', '1', '1.23', '1', 'N', '231', '113', '1', '6.5', '1', '0.5', '1.24', '0', '0.3', '9.54']
['2', '03/13/2019 06:50:11 AM', '03/13/2019 06:55:53 AM', '1', '1.02', '1', 'N', '43', '238', '1', '6', '0', '0.5', '1.36', '0', '0.3', '8.16']
['2', '03/12/2019 11:36:56 PM', '03/13/2019 07:16:34 AM', '1', '5.27', '1', 'N', '100', '7', '2', '26.5', '0', '0.5', '0', '0', '0.3', '27.3']
['2', '03/12/2019 11:28:13 PM', '03/12/2019 11:34:55 PM', '1', '0.86', '1', 'N', '234', '164', '1', '6', '0', '0.5', '1.7', '0', '0.3', '8.5']
['2', '03/12/2019 11:12:48 PM', '03/12/2019 11:23:38 PM', '1', '1.1', '1', 'N', '137', '113', '1', '8.5', '0', '0.5', '1.86', '0', '0.3', '11.16']
['2', '03/07/2019 06:25:03 AM', '03/07/2019 03:25:31 PM', '1', '8.66', '1', 'N', '163', '54', '1', '38', '0', '0.5', '7.76', '0', '0.3', '46.56']
['2', '03/04/2019 12:35:59 PM', '03/04/2019 01:07:03 PM', '2', '5.52', '1', 'N', '50', '179', '1', '23', '0', '0.5', '0', '0', '0.3', '23.8']
['2', '03/04/2019 12:13:56 AM', '03/04/2019 12:32:39 AM', '5', '11.94', '1', 'N', '70', '265', '2', '32.5', '0', '0.5', '0', '0', '0.3', '33.3']
['2', '03/03/2019 03:27:20 AM', '03/03/2019 12:01:14 PM', '1', '6.37', '1', 'N', '170', '13', '1', '27', '0', '0.5', '4.5', '0', '0.3', '32.3']
['2', '03/03/2019 03:08:07 AM', '03/03/2019 03:20:33 AM', '1', '0.92', '1', 'N', '161', '170', '1', '8.5', '0', '0.5', '1.86', '0', '0.3', '11.16']
['2', '03/02/2019 12:21:41 AM', '03/02/2019 09:00:42 AM', '2', '9.75', '1', 'N', '161', '138', '1', '34', '1', '0.5', '8', '5.76', '0.3', '49.56']
['2', '03/01/2019 01:07:25 AM', '03/01/2019 09:45:50 AM', '2', '14.54', '1', 'N', '100', '51', '1', '43.5', '0.5', '0.5', '0', '0', '0.3', '44.8']
['2', '02/28/2019 07:41:58 AM', '02/28/2019 07:55:33 AM', '5', '4.33', '1', 'N', '186', '238', '2', '15', '1', '0.5', '0', '0', '0.3', '16.8']
['2', '02/28/2019 07:03:51 AM', '02/28/2019 07:13:14 AM', '5', '1.75', '1', 'N', '230', '68', '2', '8', '1', '0.5', '0', '0', '0.3', '9.8']
['2', '02/25/2019 07:15:01 AM', '02/25/2019 07:22:26 AM', '2', '1.48', '1', 'N', '42', '116', '2', '7.5', '0.5', '0.5', '0', '0', '0.3', '8.8']
['2', '02/24/2019 05:57:25 AM', '02/24/2019 02:11:37 PM', '1', '2.21', '1', 'N', '137', '163', '1', '10.5', '0.5', '0.5', '2.36', '0', '0.3', '14.16']
['2', '02/24/2019 05:27:39 AM', '02/24/2019 05:31:59 AM', '1', '0.36', '1', 'N', '234', '137', '1', '4.5', '0.5', '0.5', '1.16', '0', '0.3', '6.96']
['2', '02/23/2019 04:48:38 PM', '02/23/2019 05:01:14 PM', '1', '2.04', '1', 'N', '107', '231', '1', '10', '1', '0.5', '2.95', '0', '0.3', '14.75']
['2', '02/23/2019 04:38:23 PM', '02/23/2019 04:46:14 PM', '1', '0.91', '1', 'N', '100', '234', '2', '6.5', '1', '0.5', '0', '0', '0.3', '8.3']
```

Shared variables

- **Implement common patterns in parallel computing:**
  - broadcast and global counter
- **Variables used in parallel operations**
  - variables are copied among parallel tasks
  - shared among tasks or between tasks and the driver
- **Types of variables**
  - broadcast variables: cache a value in all nodes
  - accumulators: a global counter shared across processes

Examples

```
conf = SparkConf().setAppName("cse4640-broadcast").setMaster(args.master)
sc = SparkContext(conf=conf)
bVar = sc.broadcast([5,10])
print("The value of the broadcast",bVar.value,sep=" ")
counter = sc.accumulator(0)
sc.parallelize([1, 2, 3, 4]).foreach(lambda x: counter.add(bVar.value[0]))
print("The value of the counter is ",counter.value,sep=" ")
```

Answer: https://tinyurl.com/yxaty238

Use cases:

- **Broadcast variables: lookup tables**
- **Accumulators: monitoring/checkpoint counters**

Spark SQL: DataFrames

- **SparkSQL: enable dealing with structured data**
- **DataFrame**
  - A distributed collection of tabular data
  - Organized into rows and named columns, similar to a table in relational database
  - Pandas and Spark DataFrames have similar concepts
    - *But single machine versus multiple machines*
  - RDD can be converted to DataFrame

DataFrame

```
inputFile =args.input_file
df =spark.read.csv(inputFile,header=True,inferSchema=True)
print("Number of partition",df.rdd.getNumPartitions())
df.show()
```

PROVINCECODE	DEVICEID	IFINDEX	FRAME	SLOT	PORT	ONUINDEX	ONUID	TIME	SPEEDIN	SPEEDOUT
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	39	10[redacted]07039	01/08/2019 00:04:07	148163	49018
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	38	10[redacted]07038	01/08/2019 00:04:07	1658	1362
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	9	10[redacted]07009	01/08/2019 00:04:07	6693	5185
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	8	10[redacted]07008	01/08/2019 00:04:07	640	544
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	11	10[redacted]07011	01/08/2019 00:04:07	118	114
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	10	10[redacted]07010	01/08/2019 00:04:07	28514	12495
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	13	10[redacted]07013	01/08/2019 00:04:07	868699	23400
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	15	10[redacted]07015	01/08/2019 00:04:07	1822	1120
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	17	10[redacted]07017	01/08/2019 00:04:07	998069	117345
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	16	10[redacted]07016	01/08/2019 00:04:07	22402	1804
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	19	10[redacted]07019	01/08/2019 00:04:07	640	791
[redacted]YN	[redacted]3023	[redacted]760	1	1	10	49	10[redacted]10049	01/08/2019 00:04:07	662	494
[redacted]YN	[redacted]3023	[redacted]760	1	1	10	48	10[redacted]10048	01/08/2019 00:04:07	2158	759
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	21	10[redacted]07021	01/08/2019 00:04:07	0	0
[redacted]YN	[redacted]3023	[redacted]760	1	1	10	51	10[redacted]10051	01/08/2019 00:04:07	2600890	54153
[redacted]YN	[redacted]3023	[redacted]528	1	2	7	20	10[redacted]07020	01/08/2019 00:04:07	330	184

Create DataFrame

DataFrames can be created from a Hive table, from Spark data sources, or another DataFrame

Load and save

- From Hive, JSON, CSV
- HDFS, local file, etc

Formats and Sources supported by DataFrames

Source: https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html

DataFrame Transformations

- **Several transformations can be done**
  - *Think transformation for relational database or matrix*
- **Select**
  - *df.select(“PROVINCECODE").show()*
- **Filter**
  - *df.filter(df[‘DEVICEID]).show()*
- **Groupby**
  - *df.groupBy(“ONUID").count().show()*
- **Handle missing data**
  - *Drop duplicate rows, drop rows with NA/null data*
  - *Fill NA/null data*

Actions with DataFrame

Actions

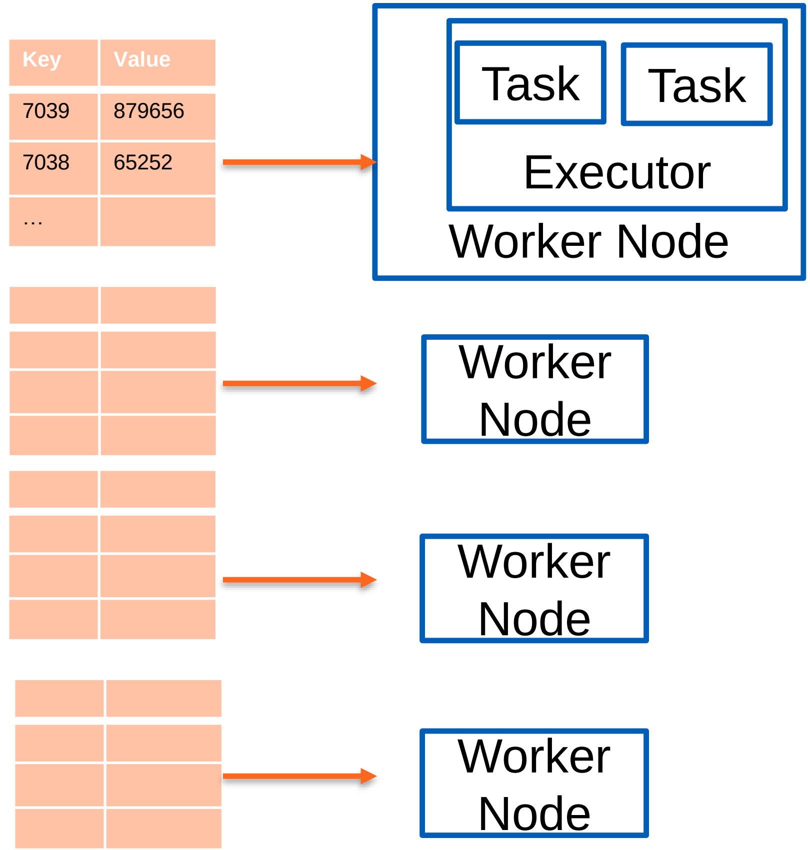
- Return values calculated from DataFrame

Examples

- reduce, max, min, sum, variance and stdev

→Distributed and parallel processing but it is done by the framework

Data Distribution



One task works on a partition at a time

→ Parallelism and performance are strongly dependent on number of partitions, tasks, CPU cores

Data distribution imbalance (skewness)

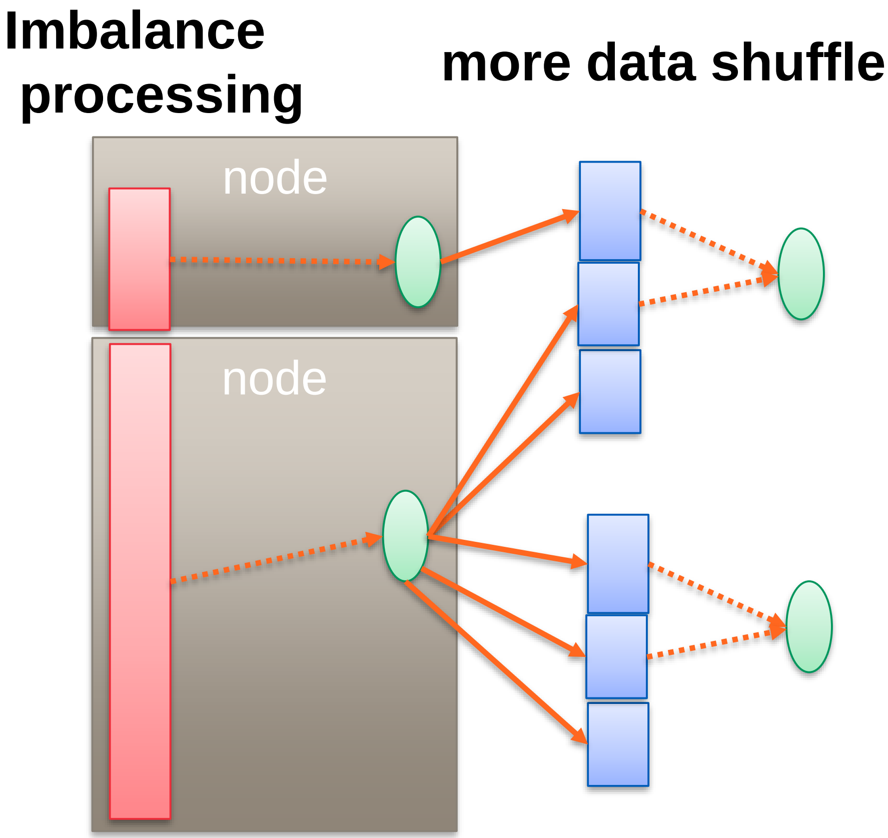
e.g.,

PYN,100XXXXX023,268XXXXX8,1,2,7,17,1005XXXXX2310207017,01/08/2019 00:04:07,998069,117345

PYN,100XXXXX023,268XXXXX8,1,2,7,16,1005XXXXX2310207016,01/08/2019 00:04:07,22402,1804

Assume that we have many data come from a province!

Data Distribution: Load balance



- **It is important to have well-balanced data distribution across nodes**
- **Detection:**
  - look at runtime execution time to see problems or check your data
- **Examples of solution:**
  - Repartition
  - Broadcast
  - Change group keys

Example of a Spark program

```
#!/usr/bin/env python2
#encoding: UTF-8
# CS-E4640
import csv
import sys
from datetime import datetime
from pyspark.sql import SparkSession
import numpy as np
from pyspark.sql import functions as F
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--input_file', help='input data file')
parser.add_argument('--output_dir',help='output dir')
args = parser.parse_args()

##define a context
spark = SparkSession.builder.appName("cse4640-onu").getOrCreate()
#NOTE: using hdfs:///..... for HDFS file or file:///
inputFile =args.input_file
df =spark.read.csv(inputFile,header=True,inferSchema=True)
#df.show()
print("Number of records", df.count())
exprs = {"SPEEDIN":"avg"}
df2 = df.groupBy('ONUID').agg(exprs)
df2.repartition(1).write.csv(args.output_file,header=True)
```

Session/Driver

Read data

Apply operators

Results

Average speedin

df2 = df.groupBy('ONUID').agg(exprs)

```
ONUID,avg(SPEEDIN)
10056[redacted]0059,2089433.0
10056[redacted]0045,1.0372027E7
10056[redacted]7061,1322842.0
10056[redacted]5059,1.710654E7
10056[redacted]3009,183.0
10056[redacted]3035,990.0
10056[redacted]9022,1576631.0
10056[redacted]9056,246569.0
10056[redacted]6006,32.0
10056[redacted]5045,508326.0
10056[redacted]0004,186.0
10056[redacted]2047,3672.0
10056[redacted]2028,54056.0
10056[redacted]2005,537.0
10056[redacted]9009,319535.5
10056[redacted]3039,19238.5
10056[redacted]6051,3123.5
10056[redacted]5030,94192.0
10056[redacted]4053,1099273.0
10056[redacted]3026,14099.0
10056[redacted]4027,0.0
```

Spark application runtime view

- **Tasks:**
  - A unit of work executed in an executor, e.g., set of transformations for a data partition
- **Stage**
  - A set of tasks executed in many nodes for computing the same operation
  - Move to a new stage: through shuffle operations
- **Job**
  - Runtime view of an action (produce a result), includes many stages

Pipelining, Shuffle and DAG

- **Operations work in a pipeline without moving data across nodes**
  - E.g., map->filter, select->filter
- **Shuffle Persistent**
  - Shuffle needs move data across nodes
  - Source tasks save shuffle files into local disks for data shuffle, then the target tasks will read data from source nodes
    - *Save time, recovery, fault tolerance*

Monitoring Spark: check Spark jobs in YARN

hadoop

Logged in as: anonymous

All Applications

- Application History
  - About
  - Applications
    - FINISHED
    - FAILED
    - KILLED
- Tools

Show 20 entries Search:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1570429323498_0009	mybdp	cse4640-nytaxicount	SPARK	default	0	Fri Oct 25 22:32:45 +0300 2019	N/A	RUNNING	UNDEFINED		Unassigned
application_1570429323498_0008	mybdp	cse4640-nytaxicount	SPARK	default	0	Fri Oct 25 22:22:08 +0300 2019	Fri Oct 25 22:25:15 +0300 2019	FINISHED	SUCCEEDED		History
application_1570429323498_0007	truong	cse4640-nytaxicount	SPARK	default	0	Fri Oct 25 22:04:20 +0300 2019	N/A	RUNNING	UNDEFINED		Unassigned
application_1570429323498_0006	truong	cse4640-nytaxicount	SPARK	default	0	Fri Oct 25 22:00:33 +0300 2019	Fri Oct 25 22:01:54 +0300 2019	FINISHED	SUCCEEDED		History
application_1570429323498_0005	truong	PySparkShell	SPARK	default	0	Fri Oct 25 21:50:36 +0300 2019	Fri Oct 25 21:51:29 +0300 2019	FINISHED	SUCCEEDED		History

Showing 1 to 5 of 5 entries First Previous 1 Next Last

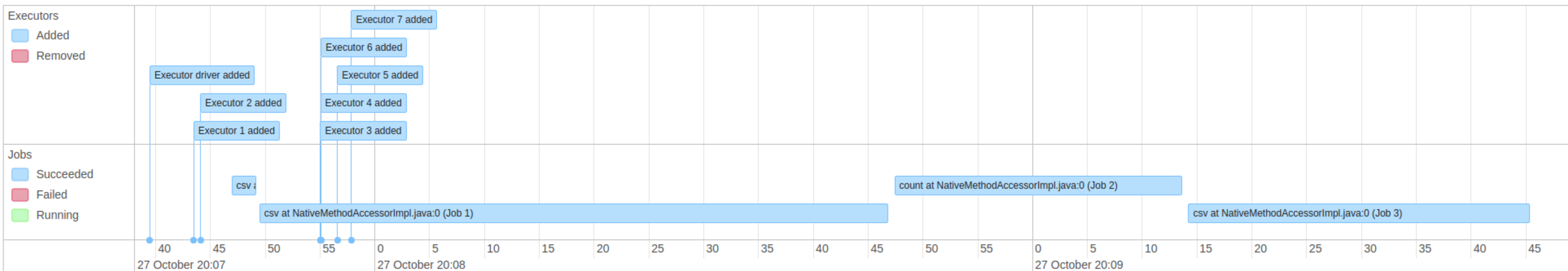
Monitoring Spark: Executors and tasks

Spark 2.3.3 | Jobs | Stages | Storage | Environment | Executors | SQL | **cse4640-nytaxicount** application UI

Spark Jobs (?)

User: truong
Total Uptime: 2.1 min
Scheduling Mode: FAIR
Completed Jobs: 4

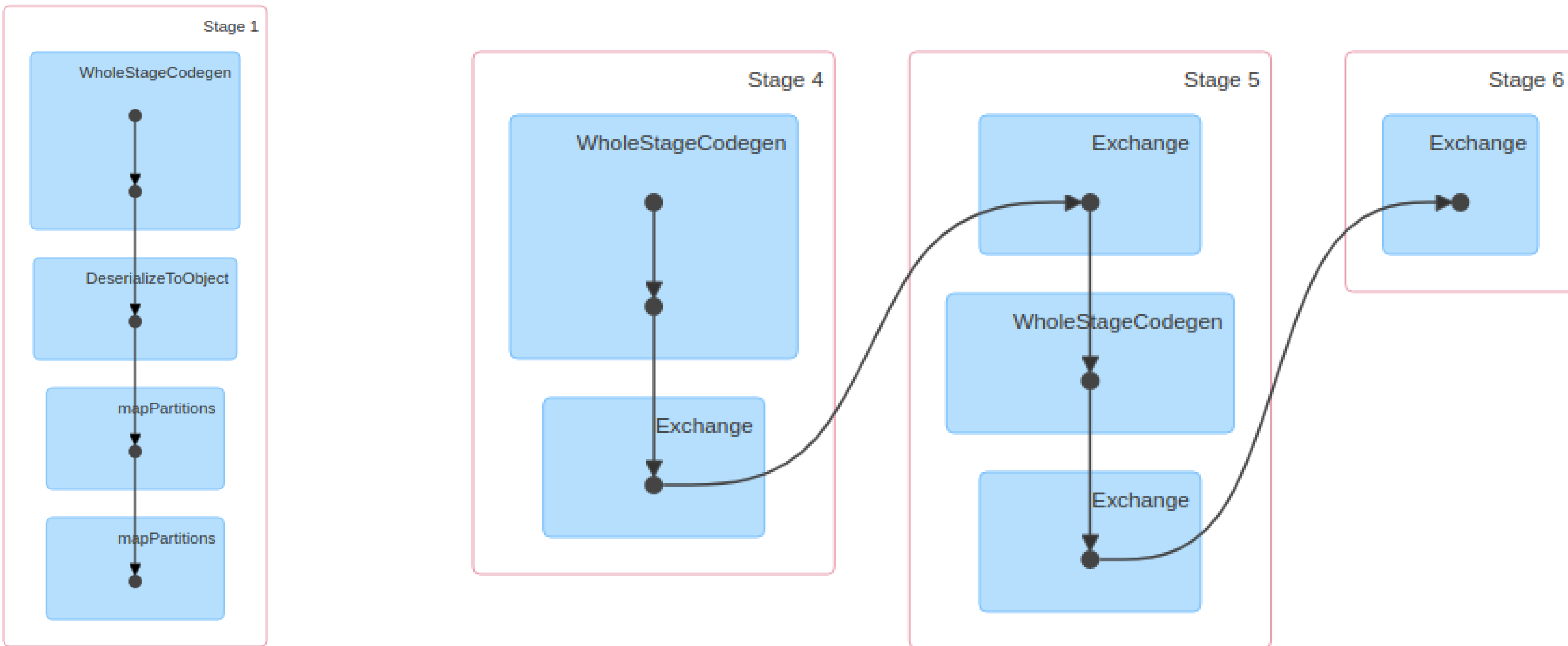
▾ Event Timeline
☐ Enable zooming



Completed Jobs (4)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
3	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2019/10/27 20:09:14	31 s	3/3	279/279
2	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2019/10/27 20:08:47	26 s	2/2	79/79
1	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2019/10/27 20:07:49	57 s	1/1	78/78
0	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2019/10/27 20:07:46	2 s	1/1	1/1

Example of Stages



Executors and tasks

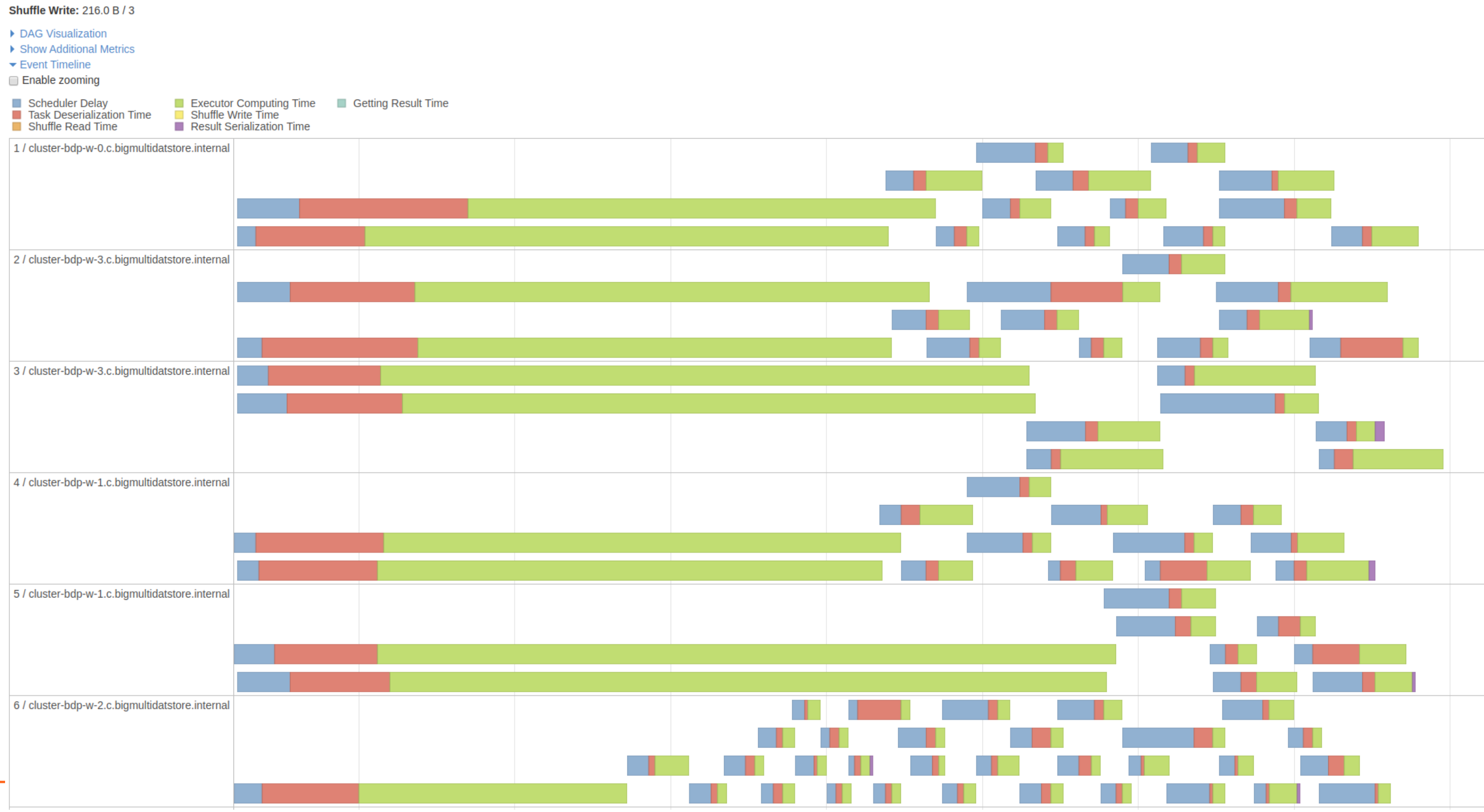
› Aggregated Metrics by Executor

Tasks (200)

Page: 1 2 > 2 Pages. Jump to 1 . Show 100 items in a page. Go

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	238	0	SUCCESS	PROCESS_LOCAL	5	cluster-bdp-w-1.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.2 s		0.0 B / 0		0.0 B / 0	
1	239	0	SUCCESS	PROCESS_LOCAL	4	cluster-bdp-w-1.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.2 s		0.0 B / 0		0.0 B / 0	
2	240	0	SUCCESS	PROCESS_LOCAL	1	cluster-bdp-w-0.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.2 s		0.0 B / 0		0.0 B / 0	
3	241	0	SUCCESS	PROCESS_LOCAL	3	cluster-bdp-w-3.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.2 s		0.0 B / 0		0.0 B / 0	
4	242	0	SUCCESS	PROCESS_LOCAL	7	cluster-bdp-w-0.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.1 s	3 ms	0.0 B / 0		0.0 B / 0	
5	243	0	SUCCESS	PROCESS_LOCAL	2	cluster-bdp-w-3.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.2 s		0.0 B / 0		0.0 B / 0	
6	244	0	SUCCESS	PROCESS_LOCAL	5	cluster-bdp-w-1.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.2 s		0.0 B / 0		0.0 B / 0	
7	245	0	SUCCESS	PROCESS_LOCAL	4	cluster-bdp-w-1.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.2 s		0.0 B / 0		0.0 B / 0	
8	246	0	SUCCESS	PROCESS_LOCAL	1	cluster-bdp-w-0.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.2 s		0.0 B / 0		0.0 B / 0	
9	247	0	SUCCESS	PROCESS_LOCAL	3	cluster-bdp-w-3.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.2 s		0.0 B / 0		0.0 B / 0	
10	248	0	SUCCESS	PROCESS_LOCAL	7	cluster-bdp-w-0.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.1 s	3 ms	0.0 B / 0		0.0 B / 0	
11	249	0	SUCCESS	PROCESS_LOCAL	2	cluster-bdp-w-3.c.bigmultidatstore.internal stdout stderr	2019/10/27 20:09:44	0.2 s		0.0 B / 0		0.0 B / 0	
12	251	0	SUCCESS	PROCESS_LOCAL	6	cluster-bdp-w-2.c.bigmultidatstore.internal stdout	2019/10/27 20:09:44	11 ms		0.0 B / 0		0.0 B / 0	

Executors and tasks



Other important support of Spark

- **Mlib - Machine learning**
  - Distributed and parallel machine learning algorithms with big data and clusters
    - *You might run many ML algorithms, but many of them are sequential*
- **Streaming**
  - Microbatching data processing in near-realtime
- **Graph Processing**
  - Parallel computation for graphs

Summary

- **Facts:**
  - MapReduce and Spark are important frameworks
  - A user/developer needs to learn to develop MapReduce/Spark applications
  - A platform operator/provider offer services for managing resources, processing and monitoring
- **Thoughts:**
  - Think about the success of Apache Spark: rich ecosystems!
  - Think if you combine data, different distributed programming supports for your big data platform

Summary

- **Focus:**
  - Learning objectives: do not just focus on "programming" – at least in this course!
  - Practical programming with MapReduce/Spark
  - What about data ingestion with Spark: using Spark to process raw data in Hadoop and ingest high-level data into big databases?
- **Action:**
  - Link to other courses: interested in ML, try to use Spark for ML → move to ML with parallel processing
  - Links to the previous lectures: check again Hadoop lectures

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io