COMP5349 –Cloud Computing

Week 9: Programming with Apache Spark and Flink, Part II

A/Prof. Uwe Roehm
School of Information Technologies

Review Question 1

How does Spark represent key-value pairs?

- **A:** RDD
- **B:** PairRDD
- **C**: DataSet

http://tinyurl.com/kmowvyt

Review Question 2

What is a lambda expression?

- **A:** Anonymous Function
- **B:** Linear Algebra Concept
- **C**: MR Design Pattern
- **D**: Flink Aggregation Operator

http://tinyurl.com/k8qpn5k

Review Question 3

Which system supports user-defined data partitioning?

- **A:** Hadoop
- **B:** Spark
- **C**: Flink
- **D**: All of the above

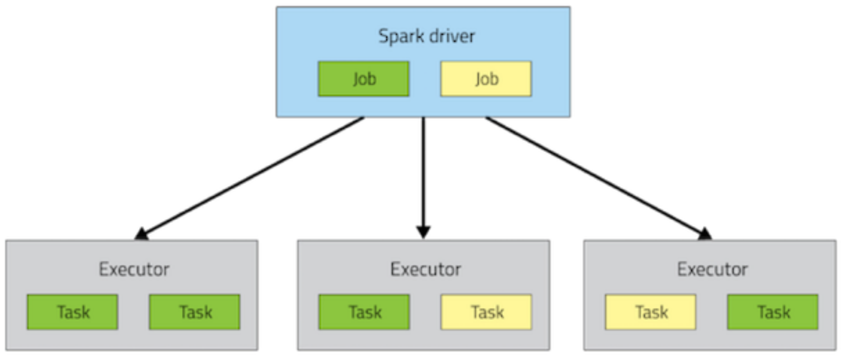
http://tinyurl.com/lyxj28c

Outline

- **Execution Engine**
  - **Job, Stage, Task**
  - **Shuffle**
- Additional Features
  - Function Closure
  - Sharing data among partitions
- Spark Machine Learning
  - Basic Data Types
  - Machine Learning Libraries
- k-Means Clustering Algorithm

[based on slides by Dr Ying Zhou]

How Spark Executes an Application

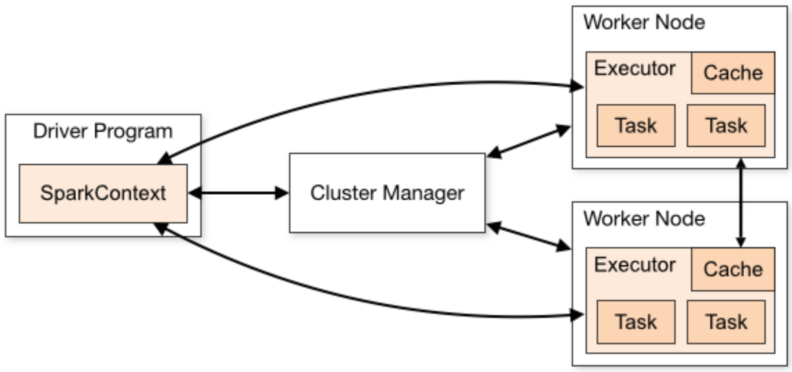


“The **driver** is the process that is in charge of the high-level control flow of work that needs to be done. The **executor** processes are responsible for executing this work, in the form of ***tasks***, as well as for storing any data that the user chooses to cache. Both the **driver** and the **executor**s typically stick around for the entire time the application is running”

Based on http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-1/
Section 5 of the original Spark paper published on NSDI’12 by Zaharia, Matei, et al

Cluster Deployment Modes

YARN/Mesos/standalone



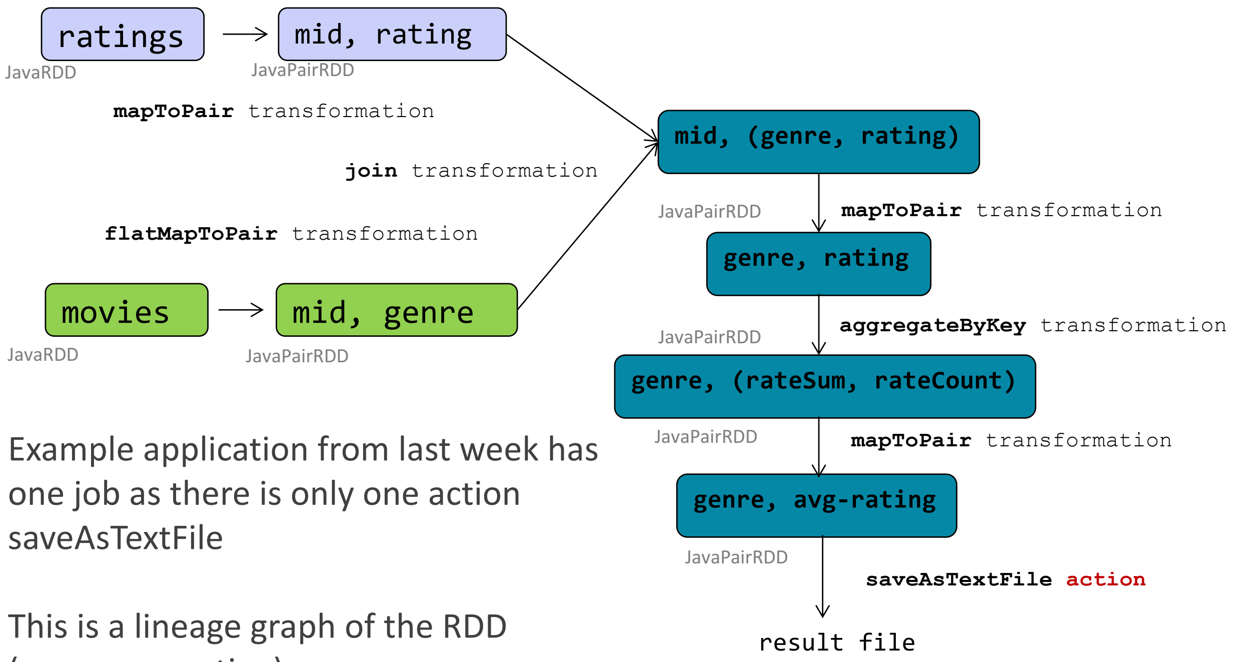
Depending on where the driver is running, spark applications can be submitted in either cluster or client mode

- "In client mode, the driver is launched directly within the spark-submit process which acts as a *client* to the cluster. Input and output of the application is attached to the console."
- "if an application is submitted from a machine far from the worker machine, it is common to use cluster mode."

[http://spark.apache.org/docs/latest/submitting-applications.html]

Job, Stage and Task

- Jobs are triggered by action such as **`count`**, **`save`**, etc

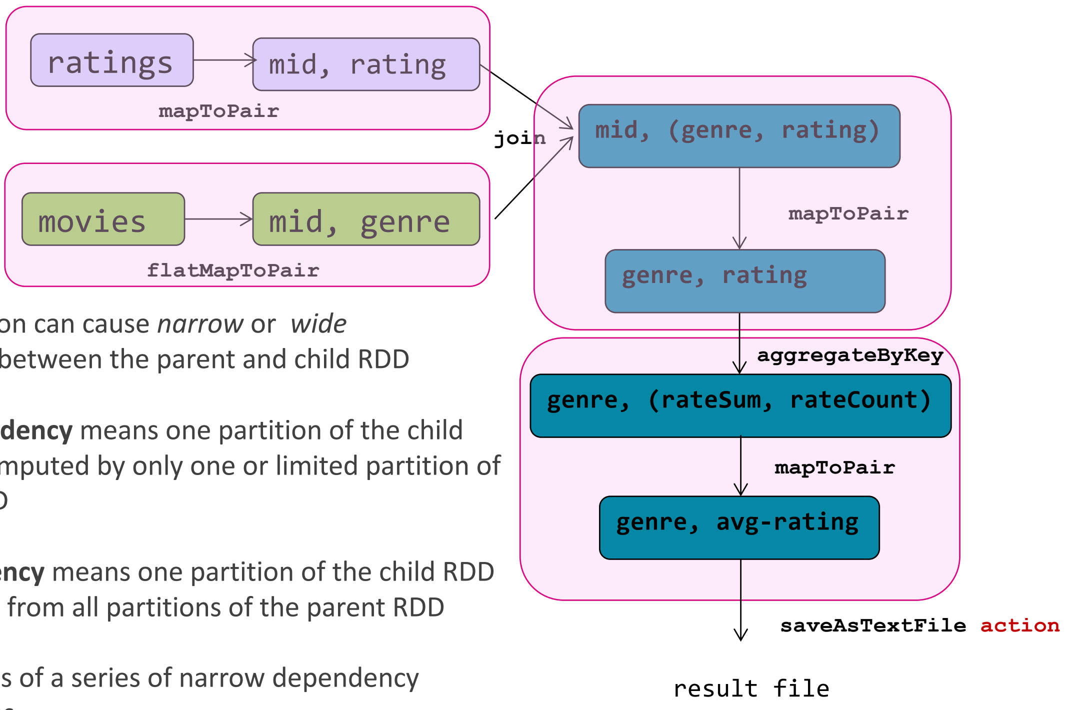


Example application from last week has one job as there is only one action saveAsTextFile

This is a lineage graph of the RDD (genre, ave-rating)

Job, Stage and Task

- A job can have many stages as a DAG based on the RDD's lineage



A transformation can cause *narrow* or *wide* dependencies between the parent and child RDD

Narrow dependency means one partition of the child RDD can be computed by only one or limited partition of the parent RDD

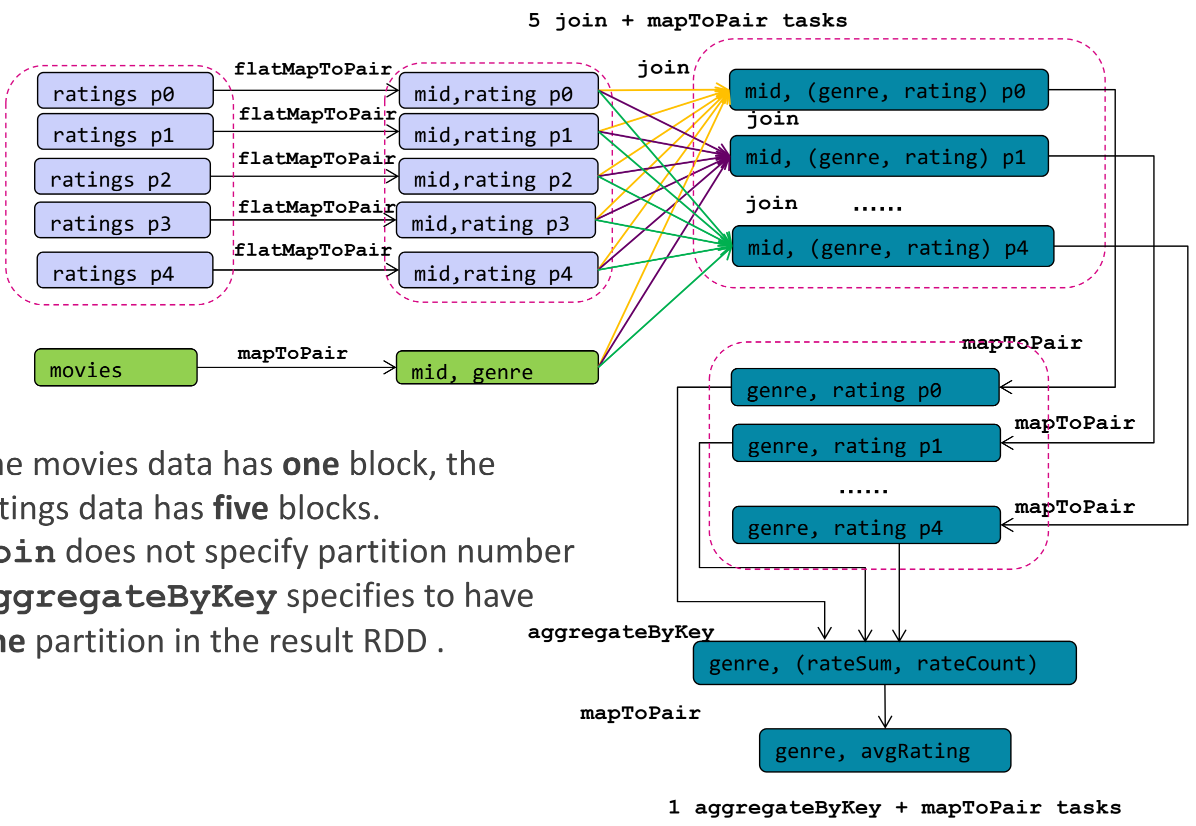
Wide dependency means one partition of the child RDD may need data from all partitions of the parent RDD

A **stage** consists of a series of narrow dependency transformations

Job, Stage and Task

- Transformations inside a stage can execute in a pipeline style to improve efficiency
  - There is no global data shuffle inside a stage
- There will be data shuffling across stages
  - Similar to shuffle in MapReduce framework
  - child-stages need to wait for shuffle phase to finish before starting
- The pipelined transformations inside a stage represent a task
  - Task is the actual execution unit of an application
  - The actual number of tasks of an application depends on the number of partitions the parent RDD has
  - Wide dependency transformation can take a number of partition parameter

Job, Stage and Task

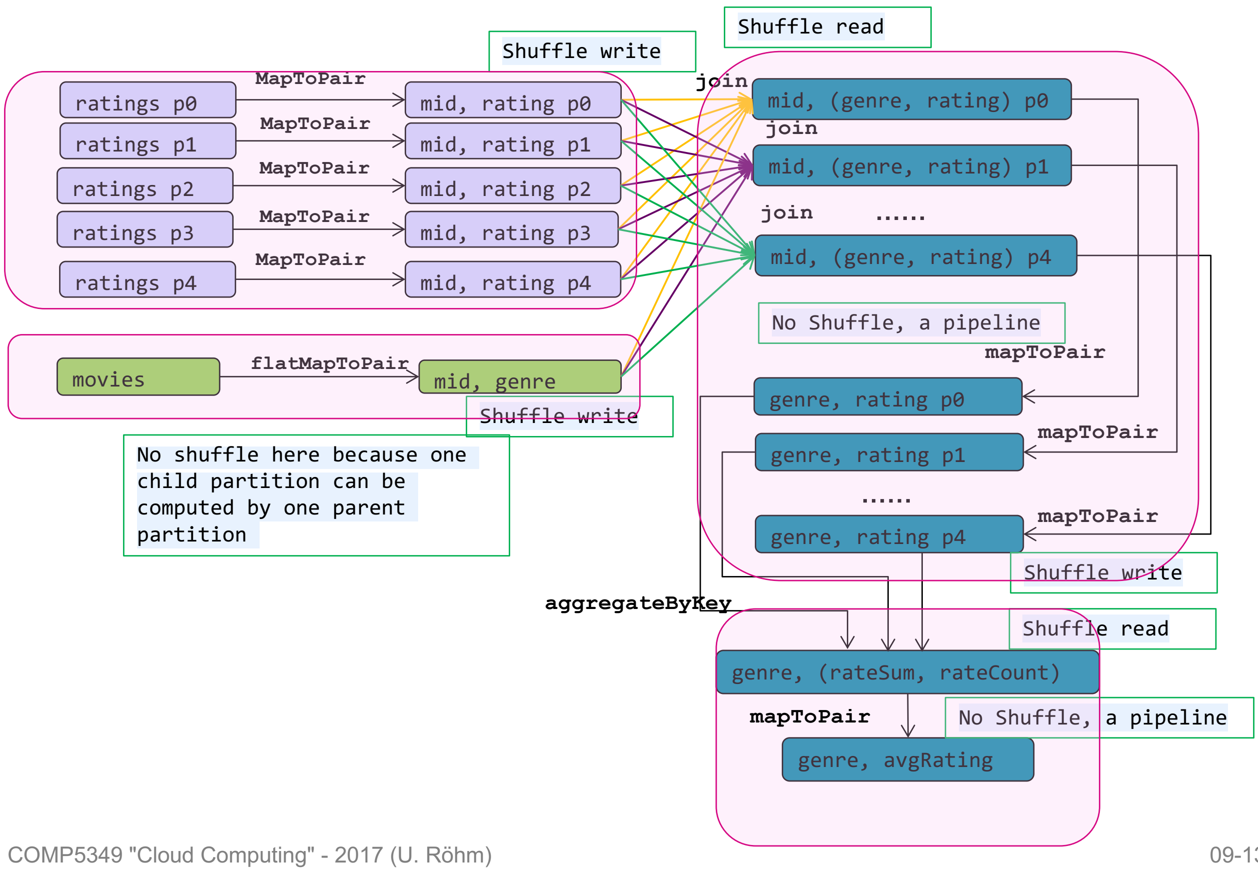


The movies data has **one** block, the ratings data has **five** blocks.
`join` does not specify partition number
`aggregateByKey` specifies to have **one** partition in the result RDD .

Shuffle and its impact

- Shuffles are fairly expensive; all shuffle data must be written to disk and then transferred over the network.
  - there is no pipelining in Spark between different stages
- Design and choose your transformations carefully to avoid shuffling too much data
- Transformations causing shuffle also stress memory if not designed properly
  - Any **`join`**, **`*ByKey`** operation involves holding objects in hashmaps or in-memory buffers to group or sort.
  - It is preferred to have more tasks to reduce memory stress in individual node

When shuffles happen

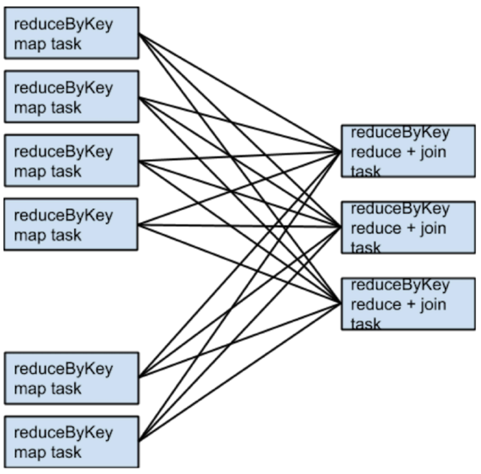


When shuffles do not happen

- There are cases where **`join`** or **`*ByKey`** operation does not involve shuffle and would not trigger stage boundary
  - If child and parent RDDs are partitioned by the same partitioner and/or have the same number of partition

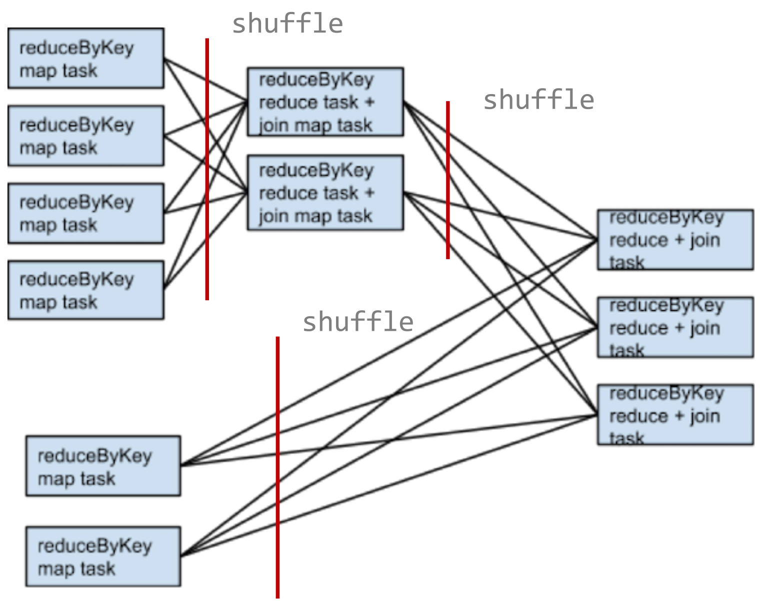
Join with input co-partitioned

```
rdd1 = someRdd.reduceByKey(...)
rdd2 = someOtherRdd.reduceByKey(...)
rdd3 = rdd1.join(rdd2)
```



When shuffles do not happen (cont'd)

- If the parent RDD uses the same partitioner, but result in different number of partitions. Only one parent RDD needs to be re-shuffled for the join operation

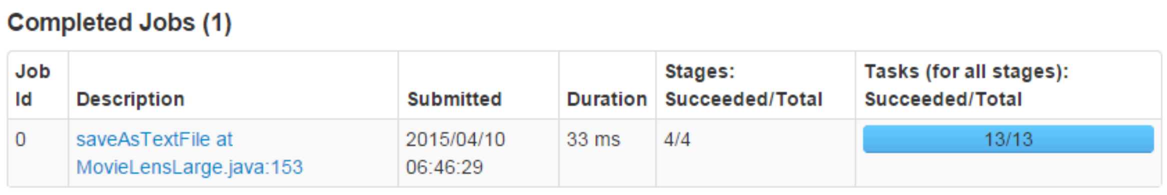

shuffle
reduceByKey map task
reduceByKey map task
reduceByKey map task
reduceByKey map task
reduceByKey reduce task + join map task
reduceByKey reduce task + join map task
shuffle
reduceByKey reduce + join task
reduceByKey reduce + join task
reduceByKey reduce + join task
shuffle
reduceByKey map task
reduceByKey map task

How to submit Spark Application

- Spark applications are usually submitted using **spark-submit** script
  - Specify a few important parameters
- For debugging on local installation, you can set all important parameters in **SparkContext** and run the program as a java application inside an IDE such as Eclipse.

```
spark-submit \
  --class ml.MovieLensLarge \
  --master yarn-cluster \
  --num-executors 2 \
  --num-cores 2 \
  sparkML.jar \
  hdfs://soit-hdp-pro-1.ucc.usyd.edu.au:8020/share/movies/ \
  hdfs://soit-hdp-pro-1.ucc.usyd.edu.au8020/user/ying/spark/
```

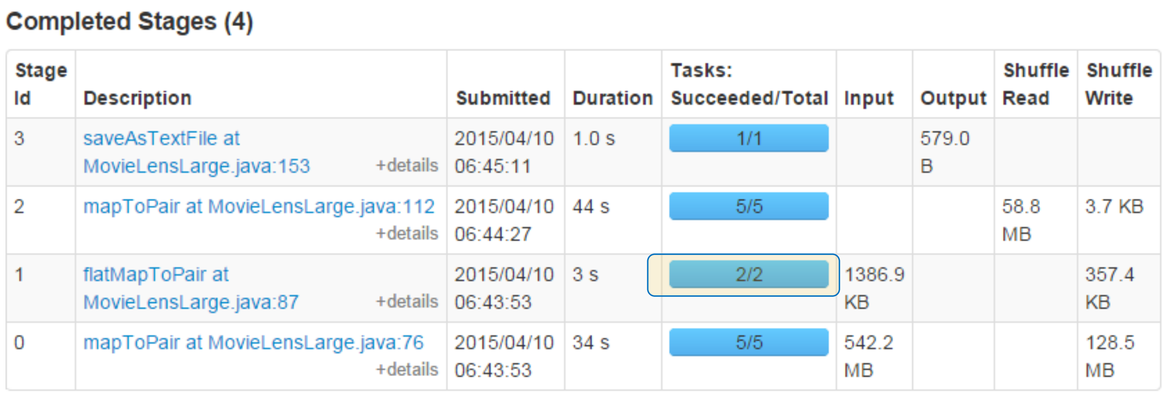
Application History Screen Shot



Completed Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	saveAsTextFile at MovieLensLarge.java:153	2015/04/10 06:46:29	33 ms	4/4	13/13

http://soit-hdp-pro-1.ucc.usyd.edu.au:18080/
http://soit-hdp-pro-1.ucc.usyd.edu.au:8080/
http://soit-hdp-pro-1.ucc.usyd.edu.au:8088/

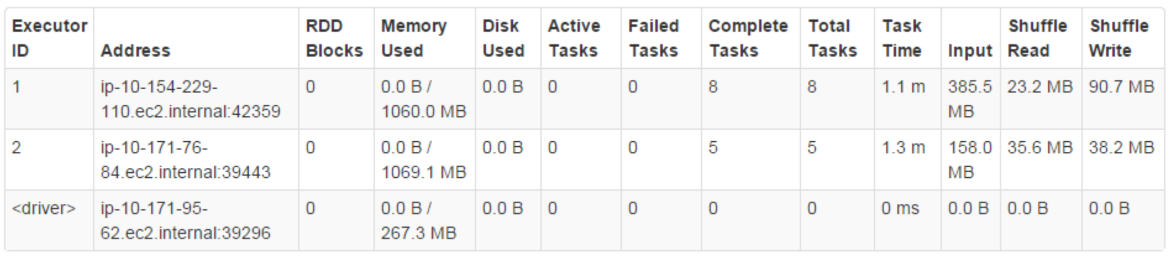


Completed Stages (4)

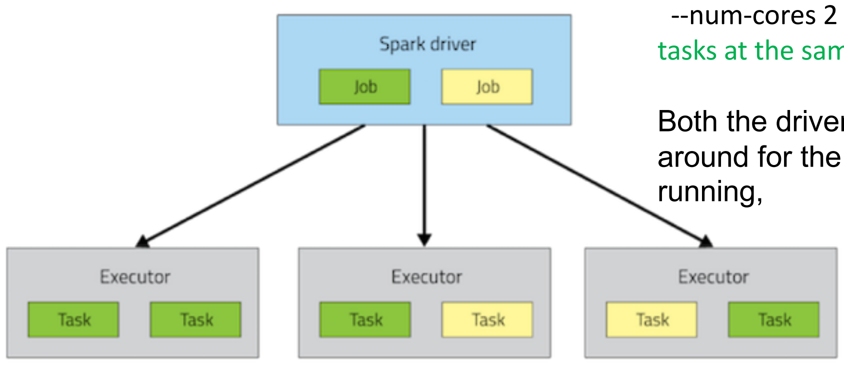
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
3	saveAsTextFile at MovieLensLarge.java:153 +details	2015/04/10 06:45:11	1.0 s	1/1		579.0 B		
2	mapToPair at MovieLensLarge.java:112 +details	2015/04/10 06:44:27	44 s	5/5			58.8 MB	3.7 KB
1	flatMapToPair at MovieLensLarge.java:87 +details	2015/04/10 06:43:53	3 s	2/2	1386.9 KB			357.4 KB
0	mapToPair at MovieLensLarge.java:76 +details	2015/04/10 06:43:53	34 s	5/5	542.2 MB			128.5 MB

Spark starts two tasks for file with only one block

Application History Screen Shot



Executor ID	Address	RDD Blocks	Memory Used	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write
1	ip-10-154-229-110.ec2.internal:42359	0	0.0 B / 1060.0 MB	0.0 B	0	0	8	8	1.1 m	385.5 MB	23.2 MB	90.7 MB
2	ip-10-171-76-84.ec2.internal:39443	0	0.0 B / 1069.1 MB	0.0 B	0	0	5	5	1.3 m	158.0 MB	35.6 MB	38.2 MB
<driver>	ip-10-171-95-62.ec2.internal:39296	0	0.0 B / 267.3 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B



--num-executors 2 // there are two executors to run this application
--num-cores 2 // each executor can run maximum 2 tasks at the same time

Both the driver and the executors typically stick around for the entire time the application is running,

Outline

- Execution Engine
  - Job, Stage, Task
  - Shuffle
- **Additional Features**
  - **Function Closure**
  - **Sharing data among partitions**
- Spark Machine Learning
  - Basic Data Types
  - Machine Learning Libraries
- k-Means Clustering Algorithm

Function Closures

- Closures basic principle
  - A **closure** is a function that has an environment (scope) of its own.
  - inner function is able to access variables defined in outer functions, but when closure is used more than once, bound variables stay same.
- The behaviour in a cluster differs from local execution
  - If the operation needs to be carried out in different partitions, the function closure will be copied and shipped to those partitions (executors)

```
int counter = 0;
JavaRDD<Integer> rdd = sc.parallelize(data);

// Wrong: Don't do this!!
rdd.foreach(x -> counter += x);

System.out.println("Counter value: " + counter);
```

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

Sharing Data

- In **Spark**, data sharing among operations(partitions) are achieved using two types of variables
- Broadcast Variable
  - Used to share small amount of read-only data
  - Similar to distributed cache in MapReduce
- Accumulator
  - Accumulators are variables that are only “added” to through an associative operation
  - Similar to counters in MapReduce

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

Sample Spark code for Broadcast Variable

```
JavaRDD<String> ratingData = sc.textFile(inputDataPath + "ratings.csv"),
                 movieData = sc.textFile(inputDataPath + "movies.csv");

JavaPairRDD<String, Integer> numRatingPerMovie = ratingData.mapToPair(s ->
           {  String[] values = s.split(",");
              return  new Tuple2<String, Integer>(values[1],1);
           }).reduceByKey((n1,n2) -> n1+ n2);

List<Tuple2<String,Integer>> topMovies = numRatingPerMovie.top(topN, new Tuple2Comparator());
List<String> topMovieIds = new ArrayList<String>();

for (Tuple2<String, Integer> t: topMovies)
    topMovieIds.add(t._1); // collects movie IDs (1st field) of the top-k movies by count of ratings

Broadcast<List<String>> topMovieShared = sc.broadcast(topMovieIds);

movieData.filter(m->{
               String movieId = m.split(",")[0];
               return topMovieShared.value().stream().anyMatch(str -> str.equals(movieId));
           }).saveAsTextFile(outputDataPath + "top" + topN + "Movies");
```

Sharing Data with Flink

- In **Flink**, the closure principle applies too; similar solutions:

- Accumulator Objects
  - registered objects with *Accumulator* interface (**add()** operation) that support global result accumulation
- Broadcast Variable
  - Used to share small amount of read-only data
- DistributedCache
  - Similar to distributed cache in Apache Hadoop
- Cross-Join
  - cross transformation combines two DataSets into one DataSet

https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/api_concepts.html

Flink Example of Global Accumulator

- Flink has several built-in accumulators
  - **IntCounter**, **LongCounter**, **DoubleCounter**

- Example: code from a user-defined transformation function

```
// first create an accumulator object (here a counter)
private IntCounter counter = 0;

// next, register accumulator object with runtime env
getRuntimeContext().addAccumulator("num-lines", this.counter);

// you can now use the accumulator anywhere in the operator fct
this.counter.add(1);

// overall global result is in the JobExecutionResult object
myJobExecutionResult.getAccumulatorResult("num-lines");
```

https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/api_concepts.html

Sample Flink code with Broadcast Variable

```
// 1. The DataSet to be broadcasted
DataSet<Integer> toBroadcast = env.fromElements(1, 2, 3);

DataSet<String> data = env.fromElements("a", "b");

data.map(new RichMapFunction<String, String>() {
   @Override public void open(Configuration parameters) throws Exception {
      // 3. Access the broadcasted DataSet as a Collection
      Collection<Integer> broadcastSet = getRuntimeContext().getBroadcastVariable
                                                         ("broadcastSetName");
   }

   @Override public String map(String value) throws Exception {
      ...
   }
}).withBroadcastSet(toBroadcast, "broadcastSetName"); // 2. Broadcast the DataSet
```

https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/batch/index.html#broadcast-variables

Sample Flink code with Distributed Cache

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
// register a file from HDFS
env.registerCachedFile("hdfs:///path/to/your/file", "hdfsFile")
// register a local executable file (script, executable, ...)
env.registerCachedFile("file:///path/to/exec/file", "localExecFile", true)
// define your program and execute
...
DataSet<String> input = ...
DataSet<Integer> result = input.map(new MyMapper());
...
env.execute();
```

```
public final class MyMapper extends RichMapFunction<String, Integer> {
   // access cached file via RuntimeContext and DistributedCache
   File myFile = getRuntimeContext().getDistributedCache().getFile("hdfsFile");
   …
}
```

https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/batch/index.html#distributed-cache

Sample Flink code with Cross Join

```
DataSet<String> ratingData = env.readTextFile(inputDataPath + "ratings.csv");
DataSet<Tuple2<String, Integer>> numRatingPerMovie = ratingData.flatMap(s ->
           {  String[] values = s.split(",");
              return  new Tuple2<String, Integer>(values[1],1);
           }).groupBy(0).sum(1);
DataSet<~~Tuple2<Integer>~~> topMovieIDs = numRatingPerMovie.partitionByRange(0)
                                                          .sortPartition(1, Order.DESCENDING)
                                                          .first(topN)
                                                          .project(1);


DataSet<Tuple2<~~Integer~~,String>> movieData = env.readCsvFile(inputDataPath + " movies.csv")
                                                 .includeFields("110")
                                                 .types(~~Integer~~.class, String.class);
movieData.cross(topMovieIDs)
         .filter( (~~m, t, t~~) -> m.equals(t) )
         .project(1)
         .writeAsCsv (outputDataPath + "top" + topN + "Movies");

env.execute("Find titles of top-N movies");
```

Outline

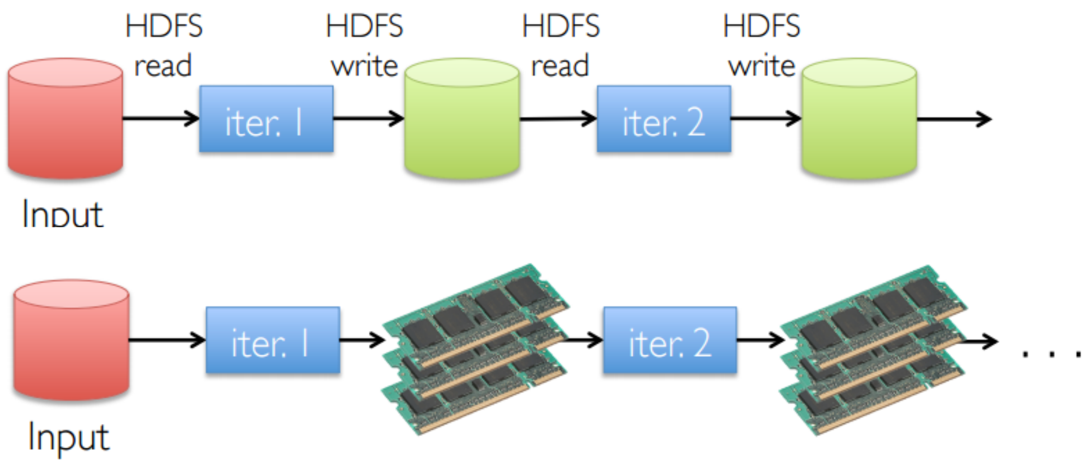
- Execution Engine
  - Job, Stage, Task
  - Shuffle
- Additional Features
  - Function Closure
  - Sharing data among partitions
- **Spark Machine Learning**
  - **Basic Data Types**
  - **Machine Learning Libraries**
- k-Means Clustering Algorithm

Machine Learning and MapReduce

- Big Data framework (MapReduce) was initially designed for simple tasks, especially embarrassingly parallel workload
  - Workload that can be easily partitioned and put to run in parallel because there is little dependencies between the partitioned tasks.
- When it is getting popular, people want to use it to solve more complex problems
  - E.g Machine Learning, data mining algorithm, or graph algorithm, like the PageRank algorithm,
  - The result are not that promising

Iterative Algorithms

- Most machine learning algorithms are iterative ones
- One of the motivations of Spark & Flink is to run iterative algorithms more efficiently than MapReduce framework



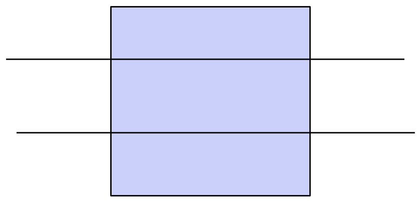
From original spark paper presentation in NSDI'12
https://www.usenix.org/sites/default/files/conference/protected-files/nsdi_zaharia.pdf

Spark ML Data Types -- Local

- Local Vector
  - 0 based index with double typed value
  - Dense vector is backed by a double array
  - Sparse vector is backed by two arrays: indices and values
  - e.g. the vector (0.5, 0.0, 0.3)
    - Dense format: [0.5, 0.0, 0.3]
    - Sparse format: (3,[0,,2],[0.5,0.3])
- Labeled point
  - A local vector with associated label, the label is of double type
    - `LabeledPoint  pos = new LabeledPoint(1.0, Vectors.dense(1.0, 0.0, 3.0));`
- Local Matrix
  - Similar to local vector, it has dense and sparse form
  - Backed by single array or combination of arrays

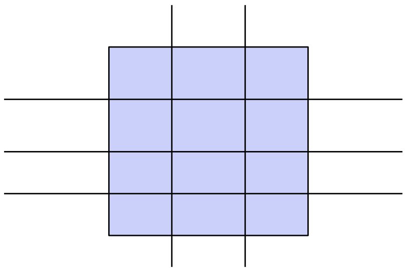
Spark ML Data Types--Distributed

- Distributed Matrix
  - Has long typed row and column indices and double typed values
  - Is stored in RDDs
  - Has several format with different partition strategy, converting a large matrix to a different format may require a global shuffle.
    - Row, coordinate and block
- Row oriented distributed matrix
  - Each row is represented by a local vector
  - The matrix is an RDD of local vectors
  - RowMatrix and IndexedRowMatrix
  - Horizontal partition



Spark ML Data Types--Distributed

- CoordinateMatrix
  - Designed for very sparse matrix
  - Is backed by an RDD of entries (row, column, value)
  - It is partitioned randomly
- BlockMatrix
  - Back by an RDD of MatrixBlocks of the type (int,int,Matrix)



Matrix sample code

```
 JavaRDD<MatrixEntry> entries =
            sc.textFile(inputDataPath+"ratings.csv").map(
                line -> {
                String[] data = line.split(",");
                return new MatrixEntry(
                   Long.parseLong(data[0]),
                   Long.parseLong(data[1]),
                   Double.parseDouble(data[2]));
            });

CoordinateMatrix mat = new CoordinateMatrix(entries.rdd());
// Convert the rating data into RowMatrix
// with the row representing user
// column representing movie
RowMatrix ratings = mat.toRowMatrix();
```

The `mllib` package

- Basic Statistics
  - Summary statistics
    - The library provides API to calculate column-wise summaries such as min, max, mean, variance, nonzero count and total count for row oriented matrix or simply RDD of Vector
  - Correlation
  - Sampling method
  - Hypothesis testing methods
- A library of machine learning methods
  - A lot of them uses matrix factorization, e.g. SVD
  - Many clustering methods
  - Association rule mining methods
  - …

Outline

- **Execution Engine**
  - **Job, Stage, Task**
  - **Shuffle**
- **Additional Features**
  - **Function Closure**
  - **Sharing data among partitions**
- **Spark Machine Learning**
  - **Basic Data Types**
  - **Machine Learning Libraries**
- **k-Means Clustering Algorithm**

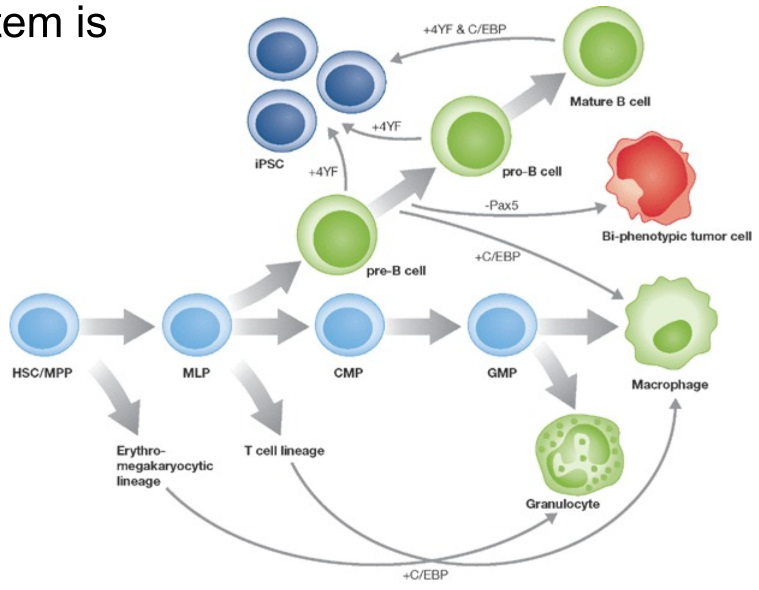
Big Data Analytics for Sciences

- **Scientific research is increasingly data-driven**
- Made possible by modern scientific instruments that allow to automatically conduct large numbers of experiments with massive output data that can be shared by community
- Examples:
  - Physics (LHC)
  - Astronomy (virtual telescopes)
  - Genomics
  - Biology / Immunology (Flow Cytometry)

Motivating Example from Immunology

The Immune system is

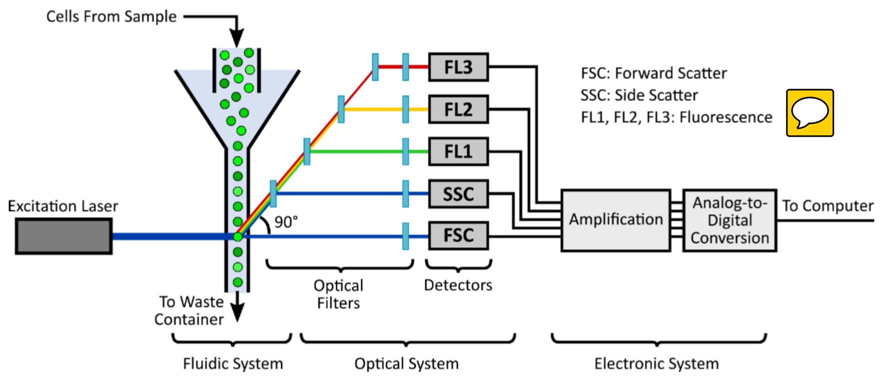
complex



[1]

Flow Cytometry

- laser-based technology to analyse physical and chemical characteristics of particles in a fluid
- used eg. for cell counting, sorting and biomarker detection



[https://benchling.com/pub/tabor-flowcal]

Data Volume with Flow Cytometry

- about 10,000 events / sec (1 event = 1 particle)
- ca. 1 to 2 million raw events per experiment

- example from immunology:
  - interested in the immune reaction of an organism to a virus
  - 4 individuals tested each day for a duration of 7 days + 1 control
    - 4 x 8 x 90,000 =~ 2.9 million data points
    - high dimensional space of up-to 34 dimensions

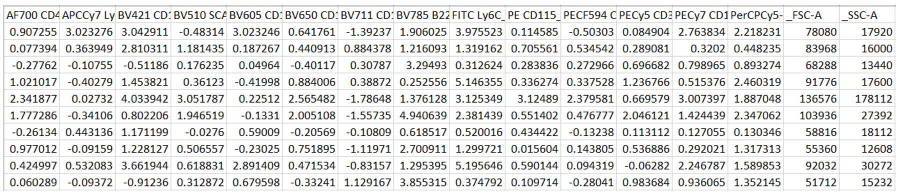
[collaboration with Nicholas King, Thomas Ashhurst, Mark Read]

Example Data Set from CPC

- Mice injected with WNV at Charles Perkins Centre
- Data extracted for several days from 4 mice each day
- Day 0 shows a disease free stage
- Days 1 – 7 show progressive states of the immune system under disease
- All samples processed through Flow Cytometry to produce data for each day with 16 dimensions and ~ 360,000 cells

Assignment Data Set

Flow Cytometry can produce up to 34 cell parameters

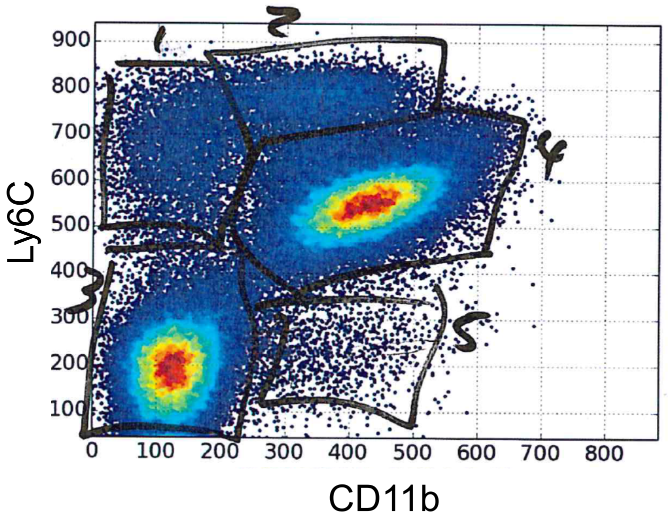


AF700 CD4	APCCy7 Ly	BV421 CD1	BV510 SCA	BV605 CD1	BV650 CD1	BV711 CD1	BV785 B22	FITC Ly6C_	PE CD115_	PECF594 C	PECy5 CD3	PECy7 CD1	PerCPCy5-	_FSC-A	_SSC-A
0.907255	3.023276	3.042911	-0.48314	3.023246	0.641761	-1.39237	1.906025	3.975523	0.114585	-0.50303	0.084904	2.763834	2.218231	78080	17920
0.077394	0.363949	2.810311	1.181435	0.187267	0.440913	0.884378	1.216093	1.319162	0.705561	0.534542	0.289081	0.3202	0.448235	83968	16000
-0.27762	-0.10755	-0.51186	0.176235	0.04964	-0.40117	0.30787	3.29493	0.312624	0.283836	0.272966	0.696682	0.798965	0.893274	68288	13440
1.021017	-0.40279	1.453821	0.36123	-0.41998	0.884006	0.38872	0.252556	5.146355	0.336274	0.337528	1.236766	0.515376	2.460319	91776	17600
2.341877	0.02732	4.033942	3.051787	0.22512	2.565482	-1.78648	1.376128	3.125349	3.12489	2.379581	0.669579	3.007397	1.887048	136576	178112
1.777286	-0.34106	0.802206	1.946519	-0.1331	2.005108	-1.55735	4.940639	2.381439	0.551402	0.476777	2.046121	1.424439	2.347062	103936	27392
-0.26134	0.443136	1.171199	-0.0276	0.59009	-0.20569	-0.10809	0.618517	0.520016	0.434422	-0.13238	0.113112	0.127055	0.130346	58816	18112
0.977012	-0.09159	1.228127	0.506557	-0.23025	0.751895	-1.11971	2.700911	1.299721	0.015604	0.143805	0.536886	0.292021	1.317313	55360	12608
0.424997	0.532083	3.661944	0.618831	2.891409	0.471534	-0.83157	1.295395	5.195646	0.590144	0.094319	-0.06282	2.246787	1.589853	92032	30272
0.060289	-0.09372	-0.91236	0.312872	0.679598	-0.33241	1.129167	3.855315	0.374792	0.109714	-0.28041	0.983684	0.936065	1.352145	51712	15232

For assignment 2, we will provide you with a flow-cytometry data set which is high dimensional (16) and has about 360k data points

Problem Motivation

Manual analysis (Gating) conducted by immunologists
- Time consuming
- Requires expertise
- Highly subjective



Gating example by Ashhurst, CPC

Problem Statement

- Gather data distribution statistics
- Identify cell populations
  1. prepare data for processing (ignore incomplete or inconsistent measurements)
  2. cluster cell measurements using *k*-means clustering
  3. output cluster centroids and sizes (number of points) as CSV

k-Means Clustering

k-Means:

- Partitional clustering approach
- Each cluster is associated with a centroid (center point)
- Each point is assigned to the cluster with the closest centroid
- Number of clusters, *k*, must be specified
- The basic algorithm is very simple

```
1: Select $K$ points as the initial centroids.
2: repeat
3:     Form $K$ clusters by assigning all points to the closest centroid.
4:     Recompute the centroid of each cluster.
5: until The centroids don't change
```

k-Means Clustering Details

- Initial centroids are often chosen randomly
- Clusters produced vary from one run to another
- *k*-Means will converge for common similarity measures
- Most of the convergence happens in the first few iterations
- Complexity is O(*n* * *k* * *i* * *d*)
  - *n* = number of points, *k* = number of clusters, *i* = number of iterations, *d* = number of attributes (or dimensions)

Assignment 2

- Assignment 2 to be released tomorrow or on the weekend
- We plan to extend the deadline until Wk11 to give you again 2 weeks to work on it
- Choice of either Spark or Flink for assignment 2
- Should use then the other system in assignment 3 which will be similar in complexity
  - check with tutor on in which order you prefer to work on these systems
- A2 marking still ongoing. Results should be ready by next week