APACHE SPARK

Getting Started

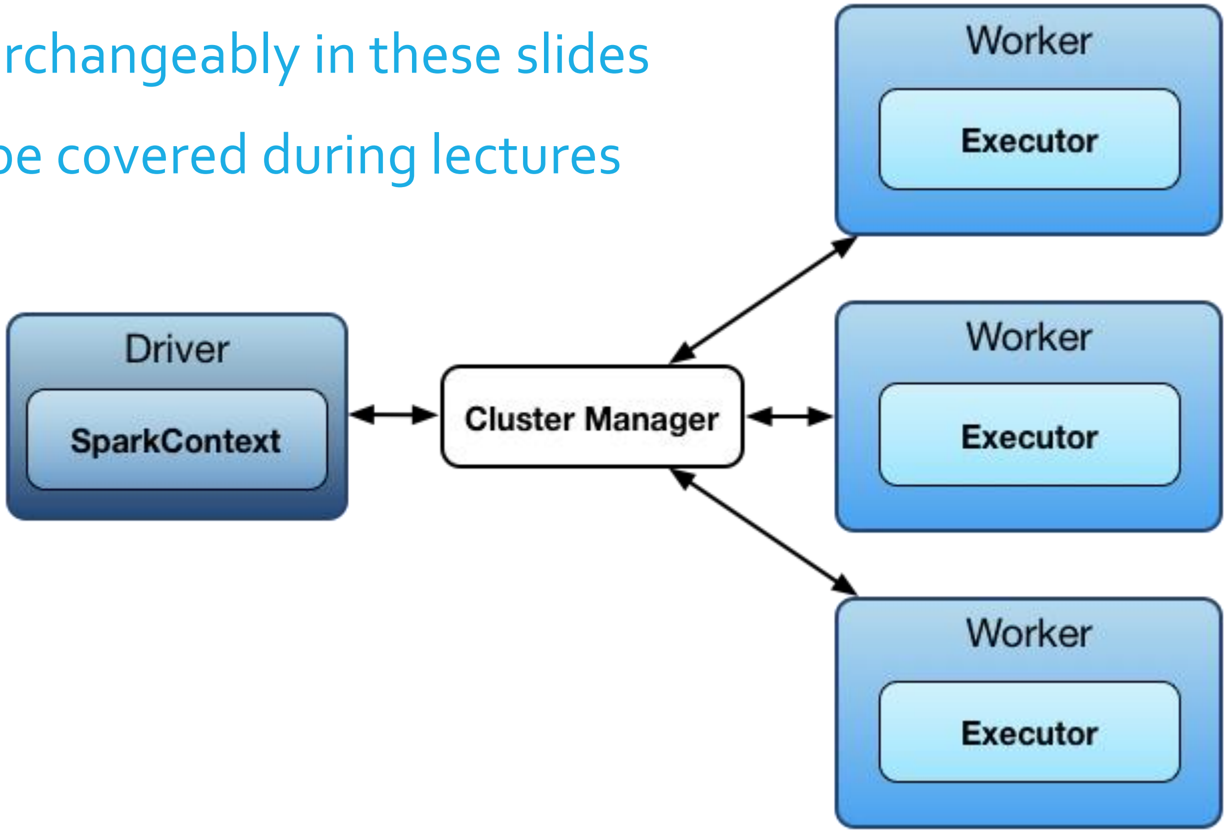
Brian Setz
b.setz@rug.nl

What is Spark?

- Big Data distributed processing platform
  - https://spark.apache.org/
  - Comprehensive guide: https://spark.apache.org/docs/latest/index.html
- Based on the MapReduce programming model
  - https://en.wikipedia.org/wiki/MapReduce
- High performance as computations are executed in-memory
  - Be mindful of operations that force writing to disk e.g. shuffle
    - https://spark.apache.org/docs/latest/rdd-programming-guide.html#shuffle-operations

Spark Architecture

- Worker and Slave are used interchangeably in these slides
- Spark/MapReduce details will be covered during lectures



Spark Execution Modes

- Spark Local
  - Used for development, easier to test and debug since it is ran locally
  - Started from (Scala/Java/Python) code
- Spark Cluster
  - Used for deployment
    - We would like to see a cluster deployment during the demo
      - Preferably using –deploy-mode cluster
  - Different cluster manager modes[1]:
    - Standalone, easiest to set-up: https://spark.apache.org/docs/latest/spark-standalone.html
    - Mesos
    - YARN
    - Kubernetes

[1] https://spark.apache.org/docs/latest/cluster-overview.html#cluster-manager-types

Spark Architecture II

- Driver is where the SparkContext is created
  - It can be on your machine, when you run the application locally
  - It can be on the cluster, when you use **spark-submit** to submit the application to the cluster
  - Driver, Workers (slave), and Cluster Manager must be able to communicate via the network
- When using <u>Spark Local</u>, driver is always on your own machine
- When using <u>Spark Cluster</u>, driver can be on your machine or on the cluster
  - deploy-mode client → runs driver on your machine
  - deploy-mode cluster → runs driver on the cluster
- Some operations return all data to the driver (e.g. **collect()**), use these sparingly
- Cluster manager depends on the mode in which the cluster is deployed

Interacting with Spark

- Programming in Scala, Java, or Python
  - Scala is recommended, as it is native to Spark
- You define operations that have to be executed on the data
- Spark executes these operation in a distributed fashion, parallelizing where possible
- Parallelization is done across Workers
  - In Spark Local, as many workers as you have CPU cores
  - In Spark Cluster, as many workers as you have nodes to deploy them on

Development Environment

- Install Java 8 SDK + JDK
  - http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- Install IntelliJ **Ultimate** Edition
  - https://www.jetbrains.com/idea/download/#section=windows
  - Register for a free ultimate edition using your educational e-mail address
  - Make sure to install the Scala Plugin in IntelliJ
- Install sbt (so it is available from the command line)
  - https://www.scala-sbt.org/download.html
- Download Spark
  - http://spark.apache.org/downloads.html
  - Set the SPARK_HOME environment variable
  - You will need the scripts in the bin folder for submitting to the cluster

Spark Bootcamp Project

- Clone the Spark bootcamp repository
  - GitHub repo: https://github.com/BrianSetz/spark-bootcamp
  - Import the project as an **sbt project**, import will take a while (indexing is last step)
  - Run: SparkLocalMain (in src/main/scala/nl/rug/sc/app), follow instructions in the console, and explore the code
  - **Note**: Spark is only compatible with Scala 2.11.x or 2.10.x. Using 2.12.x will fail!
- SparkLocalMain runs the project on your local machine, creating a local cluster with as many workers as you have CPU cores.
- There is also SparkSubmitMain which runs remotely and will be explained later

Spark Bootcamp Project II

- The Spark bootcamp project shows how to use Spark
  - Comments can be found in the files that explain every step
- The project demonstrates:
  - Basic RDD’s
  - Basic DataFrames
  - Basic DataSets
  - Advanced DataSets
  - Realistic example using DataSets
- These examples are demonstrated on a local cluster and a remote cluster
  - SparkLocalMain (run on your local machine)
  - SparkSubmitMain (send to cluster using spark-submit)

Spark Bootcamp Project III

- Algorithms you run on Spark Local will also work on Spark Cluster
  - Note: think about how to access your data, for example when deploying to a remote cluster, all workers need to be able to access the data. This means using a distributed file storage (DFS) or (no)SQL database.
- There are many ways you can connect your Spark application to a database / data source. The project shows one: reading from CSV.
  - Built-in HDFS connector
  - Cassandra connector: https://github.com/datastax/spark-cassandra-connector
  - Built-in JDBC connector
  - MongoDB connector: https://github.com/mongodb/mongo-spark
  - Many more

Spark Cluster (Standalone)

- Standalone cluster requires 1 master, and as many slaves as you need / have nodes
  - http://spark.apache.org/docs/latest/spark-standalone.html
- Launching applications on your cluster: http://spark.apache.org/docs/latest/submitting-applications.html
- Ideally, deploy using Docker to a cloud provider (Google Cloud, Amazon, Azure, Digital Ocean, …), using a Virtual Machine is also possible
  - Tutorial: https://docs.docker.com/get-started/
  - Many pre-existing Docker images to deploy a cluster; https://hub.docker.com/
  - Example Docker image also provided in next slide

Example Standalone Cluster Setup

- Docker image: https://github.com/BrianSetz/docker-spark
  - Based on: https://github.com/P7h/docker-spark
  - Spark 2.2.1, Scala 2.11.12, SBT 1.1.0
- Start Master:

  -
    ```
    docker run -d --rm --name spark-master -p 4040:4040 -p 8080-8081:8080-8081 -p 7077:7077 briansetz/docker-spark:2.2.1 spark/sbin/start-master.sh
    ```
- Start Slave(s) / Worker(s):

  -
    ```
    MASTER_IP=$(docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' spark-master)
    ```

  -
    ```
    docker run -d --rm --name spark-slave briansetz/docker-spark:2.2.1 spark/sbin/start-slave.sh spark://${MASTER_IP}:7077
    ```

Example Standalone Cluster Setup II

- The example assumes that you are deploying the master and the slave on the same machine. When deploying slaves on other machines, use the correct master IP.

- You can pass additional command line parameters to start-master.sh and start-slave.sh
  - https://spark.apache.org/docs/latest/spark-standalone.html

- Another good alternative is to run Spark on top of YARN

Spark Bootcamp Project IV

- Now that you have created the cluster, you can submit a job to the cluster
  - Unlike local deployment, where you run the application in IntelliJ, cluster deployment requires you to build a JAR file that has to be submitted.

- To build a (fat) JAR file, use sbt-assembly
  - https://github.com/sbt/sbt-assembly
  - It is included in the example project, you can run the following command in the root of the project to create the far JAR:
    - `sbt assembly`
  - After running the command, the console will show the location of the fat jar
  - Submit the far jar:
    - `./spark-submit --class nl.rug.sc.app.SparkSubmitMain --deploy-mode client --master spark://localhost:7077 <your-path-to>/spark-bootcamp/target/scala-2.11/spark-bootcamp-assembly-0.1.jar`

Spark Bootcamp Project V

- When submitting to remote clusters:
  - Make sure that all workers can access the required files / data which means using a database or distributed file system
  - Think about where your driver is running (deploy-mode) and how this affects, for example, directories
  - Submit a fat JAR, a jar containing all dependencies in your project, otherwise you experience **ClassNotFoundException**'s as the slaves do not have these dependencies
  - Be careful when returning data to the driver, if you return a lot of data to a driver that is running on your local machine, this will be slow as data has to be transferred from your remote cluster to the local machine. Better would be to run the driver in the cluster itself, reducing the latency

Tips

- Avoid working with Resilient Distributed Datasets (RDDs) directly
  - Prefer working with:
    - Data Frames (Spark SQL)
    - Data Sets
  - Frames and Sets offer much higher levels of abstraction
  - Use RDDs only if you need low-level transformations / control
- Use Spark's web-based User Interface to assist with debugging
  - Running on port 4040
- Do **NOT** use Spark 1.6 or lower, only use the latest version (=2.2.1), be mindful of online tutorials that still use Spark 1.6 or older.
- For the online/streaming data of your algorithm, look at Spark Streaming:
  - https://spark.apache.org/streaming/
- Start on time with migrating from Spark Local to Spark Cluster
- Recommended programming language is Scala, it is native to Spark

Troubleshooting

- If you cannot import the project as an SBT project, you are either missing the Scala plugin, or the Scala plugin is out of date. Make sure you also use the latest version of IntelliJ

- In case you get a **ClassNotFoundException** when running “SparkLocalMain”, try updating and/or restarting IntelliJ

- A **FileNotFoundException** is expected when submitting “SparkSubmitMain”

Troubleshooting II

- sbt-assembly & fat JARs
  - Creating a fat JAR when your project has many dependencies can be an issue
    - Dependencies can have configuration files or directories (META-INF) which have to be merged
    - Dependencies can depend on different versions of a particular dependency
  - Solutions:
    - Specify a merge strategy to deal with merging META-INF, config files, etc.
      - Example: https://coderwall.com/p/6gr84q/sbt-assembly-spark-and-you
    - Use shading to overcome different versions of the same dependency
      - Example: https://stackoverflow.com/questions/45989052/sbt-assembly-shading-to-create-fat-jar-to-run-on-spark
    - Alternative but not recommended: copy the individual JAR files of your dependencies to each of the workers so that you do not have to build a fat JAR

(Spark) Project Goals

- Deadlines: Create Pull Request on Wednesday before the deadline at 12:00
- Parallelize a non-trivial algorithm
  - Your algorithm processes offline batch data and online streaming data
    - Ideally the same type of algorithm is used for offline/online
  - Example of a trivial algorithm: word count. Demonstrate some complexity
- Architecture
  - Data Source & Sink (database / distributed file systems)
  - Message Queue (streaming data)
  - MapReduce framework (e.g. Spark)
- Excellent projects
  - All components of your architecture are distributed (running on different VM's, nodes, machines)
- Demo → you should be able to visualize how work is parallelized, e.g. via the Spark UI

Resources

- RDD vs DataFrame vs DataSet: http://why-not-learn-something.blogspot.nl/2016/07/apache-spark-rdd-vs-dataframe-vs-dataset.html
- Common mistakes: https://www.slideshare.net/cloudera/top-5-mistakes-to-avoid-when-writing-apache-spark-applications
- Spark Streaming: https://www.slideshare.net/prakash573/spark-streaming-best-practices
- Git tutorial: https://try.github.io/levels/1/challenges/1
- Spark Cluster Deployment (outdated but useful): http://mkuthan.github.io/blog/2016/03/11/spark-application-assembly/
- **Mining of Massive Datasets**, Jure Leskovec, Anand Rajaraman, Jeff Ullman, available for free at http://www.mmds.org/
- **Data-Intensive Text Processing with MapReduce**, Jimmy Lin and Chris Dyer, available for free at http://lintool.github.io/MapReduceAlgorithms/

Questions?

- Ask your TA's during the lab session
- Create an issue on GitHub and assign your TA's + BrianSetz