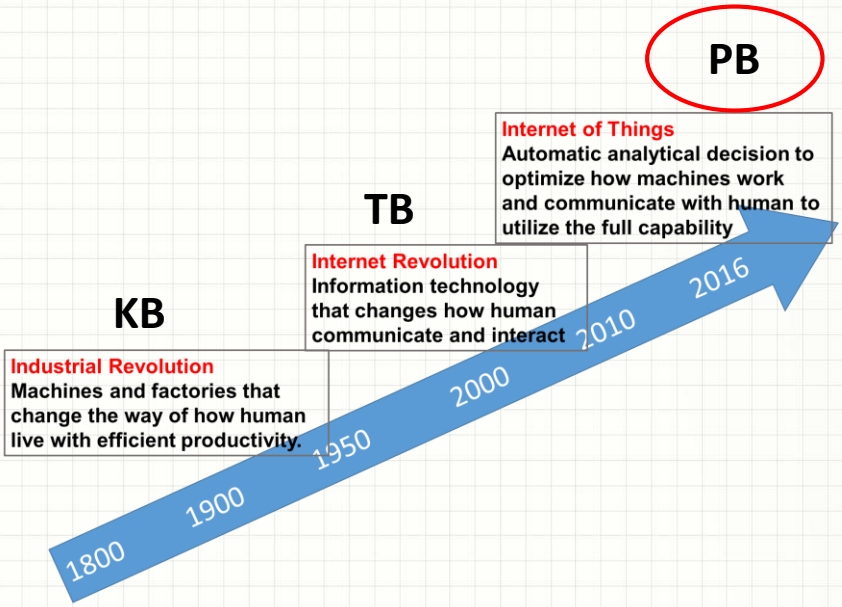
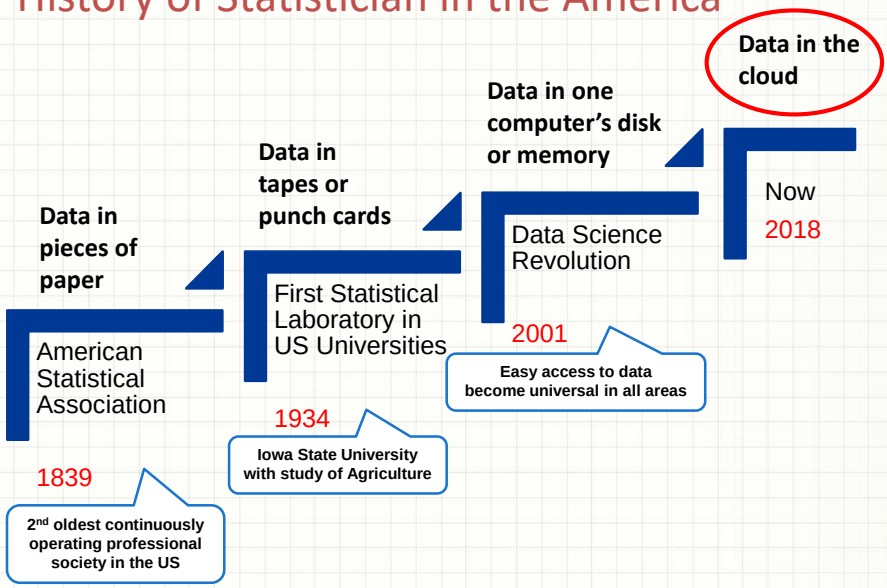

BIG DATA CLOUD PLATFORM
The New Norm
Big Data Platform Notebook: link


A BRIEF INTRODUCTION

New Wave of Industrial Revolution

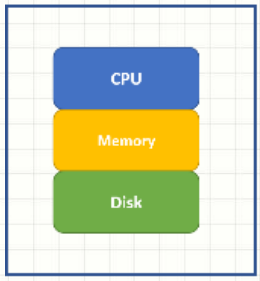


History of Statistician in the America



Big Data Cloud Framework

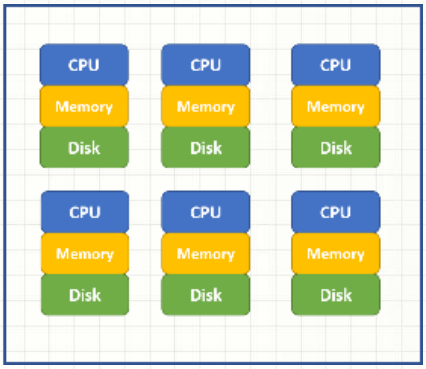
Single Computer (Local)



- Shared disk
- Shared memory
- Parallel computation

- Data redundancy
- Low coast hardware
- Professionally managed

Cluster of Computers (Cloud)



- Scalability to process large dataset
- Centralized data source for easy retrieval
- Robust and reliable for data safety
- Worry-free behind sense maintenance
- Production ready for easy implementation
- Automatic refresh and scheduled update

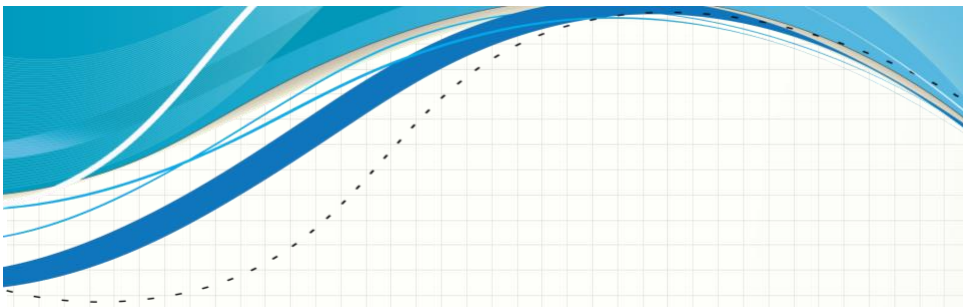
Hadoop / MapReduce / Spark

- Hadoop build on Commodity Hardware
  - ✓ Low cost and scalable by simply added more hardware
  - ✓ Low reliability (such as regular hard disk failures) for each individual computation node
  - ✓ Need robust file system to ensure data safety and integrity
- HDFS (Hadoop Distributed File System)
  - ✓ Cut data into small segments (i.e. 64MB) as blocks
  - ✓ Duplicate each data block N times (i.e. N=3) and save them across computation nodes in the cluster
  - ✓ Master node saves all the data mapping information (i.e. meta data)
  - ✓ Data nodes save actual data blocks
  - ✓ When a few data nodes are down, data is still safe
- MapReduce Operation (Demo)
- Hive and Pig: application software for easy access the data stored in HDFS and apply analytics
- Spark
  - ✓ Build on top of HDFS and other cluster file systems
  - ✓ Provide in-memory computation for faster speed
  - ✓ Provide parallel algorithm for many machine learning methods through MLib
  - ✓ Provide easy to use interface for data scientist
  - ✓ Users do not need to know the details of how data and algorithm are paralleled

Cloud Environments Providers

- Amazon AWS cloud environment
- Microsoft Azure cloud environment
- Google cloud platform
- Databricks Community Edition

- For beginners, Databricks provides an easy to use cloud system for learning purpose
- It provides a user-friendly web-based notebook environment that can create Hadoop/Spark/GPU cluster on the fly to run R/Python/Scala/SQL
- Some reference:
  - https://docs.databricks.com/user-guide/faq/sparklyr.html
  - http://spark.rstudio.com/index.html



DATABRICKS BIG DATA PLATFORM WITH SPARK

Library Installation

- First, we need to install *sparklyr* package which enables the connection between master or local node to Spark cluster environments.
- As it will install more than 10 dependencies, it may take more than 5 minutes to finish. Be patient while it is installing!
- Once the installation finishes, load the *sparklyr* package as illustrated by the following code:

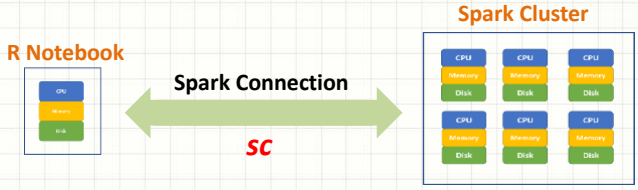
```
if (!require("sparklyr")) {
  install.packages("sparklyr")
}

library(sparklyr)
```

Create Connection

- we need to create a Spark Connection to link the local node running the R notebook to the Spark environment. The options for different environments are different, here we set "method" option to "*databricks*":

```
sc <- spark_connect(method = "databricks")
```

- The created Spark Connection (i.e. ***sc***) will be the pipe that connects R notebook to the Spark Cluster.



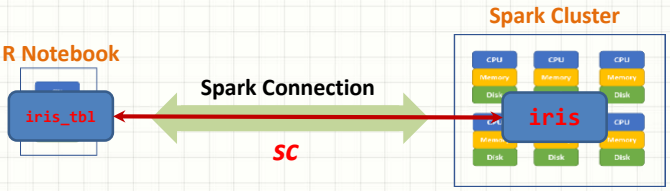
Establish Connection to Spark DataFrame

- For illustration purpose, we will use the **iris** dataset which is part of the dplyr package:

```
library(dplyr)
head(iris)
```

- Copy local R data frame: **iris** to Spark DataFrame **iris** and we can refer Spark DataFrame **iris** through **iris_tbl:**

```
iris_tbl <- sdf_copy_to(sc = sc, x = iris, overwrite = T)
```



- We can also **establish connection** between local R object ***my_df_tbl*** to **existing** Spark DataFrame ***my_df***: ***my_df_tbl <- tbl(sc, my_df)***

Manipulate Spark DataFrame Through R

- Once connection is established, we can always refer **iris_tbl** in R notebook to operate on Spark DataFrame **iris** using the scalable computation power of Spark cluster

```
iris_tbl %>%
  mutate(Sepal_Width = ROUND(Sepal_Width * 2) / 2) %>%
  group_by(Species, Sepal_Width) %>%
  summarize(count = n(), Sepal_Length = mean(Sepal_Length),
            stdev = sd(Sepal_Length))
```

- With the ***sparklyr*** packages, we can use many functions in ***dplyr*** to Spark DataFrame directly to **iris_tbl**, same as we are applying ***dplyr*** functions to a local R data frame.
- The advantage is more significant when the Spark DataFrame is huge (such as the data take 20GB+ of memory to store)

Collect Result Back to R Notebook

- Even though we can run many of the ***dplyr*** functions on Spark DataFrame, we cannot apply functions from other packages to Spark DataFrame direction (such as ggplot()).
- To use functions from other packages, we need to bring data (usually aggregated and much smaller in size) back to R Notebook using ***collect()*** function.
- Once the data is brought back to R Notebook, we can use any library. With code below, ***iris_summary*** is a local R object

```
iris_summary <-
  iris_tbl %>%
  mutate(Sepal_Width = ROUND(Sepal_Width * 2) / 2) %>%
  group_by(Species, Sepal_Width) %>%
  summarize(count = n(), Sepal_Length = mean(Sepal_Length),
            stdev = sd(Sepal_Length)) %>%
  collect
```

Apply Statistical and Machine Learning Models to Spark DataFrame Through R

- One of the BIGGEST advantage is that there are many popular statistical and machine learning models developed in Spark system (i.e. MLlib) for Spark DataFrame to leverage the scalable computation power of Spark Cluster.
- These models include: linear regression, logistic regression, Survival Regression, Generalized Linear Regression, Decision Trees, Random Forests, Gradient-Boosted Trees, Principal Components Analysis, Naive-Bayes, K-Means Clustering.
- Conveniently, we can use R notebook to call functions to apply these machine learning algorithms to Spark DataFrame which enable minimum effort for statistician to leverage the Spark environment.

Fit Regression to Spark DataFrame

- We can call `ml_linear_regression` to fit a linear regression model to Spark DataFrame `iris` by referring to `iris_tbl`
- The response variable is `"Sepal_Length"`
- The explanatory variables are `"Sepal_Width"`, `"Petal_Length"`, `"Petal_Width"`
- The fitted results are stored in `fit1`

```
fit1 <-  ml_linear_regression(x = iris_tbl, response =
"Sepal_Length", features = c("Sepal_Width", "Petal_Leng
th", "Petal_Width"))


summary(fit1)
```

Fit K-means Cluster to Spark DataFrame

- We can call `ml_kmeans()` to fit a k-mean clustermodel to Spark DataFrame `iris` by referring to `iris_tbl`
- The variables are `"Petal_Length"`, `"Petal_Width"`
- The fitted results are stored in `fit2`
- We can then apply the model to a specific dataset (in this case the original dataset) using `ml_predict()` function
- Finally we can use `collect()` function to bring results back to R Notebook for further analysis

```
fit2 <- ml_kmeans(x = iris_tbl, k = 3,
                  features = c("Petal_Length", "Petal_Width"))
print(fit2)

prediction = collect(ml_predict(fit2, iris_tbl))
```

Quick Summary

In the above a few sub-sections, we illustrated:

- ❑ The relationship between local node (i.e. where R notebook is running) and Spark Clusters (i..e where massive data are stored and computation are done in parallel)
- ❑ ***Sparklyr*** library to established the connection between local node and Spark Cluster
- ❑ How to copy a local data frame to a Spark DataFrames (please note if your data is already in Spark environment, there is no need to copy).
- ❑ Establish connection between local R object ***my_df_tbl*** to existing Spark DataFrame ***my_df***: ***my_df_tbl <- tbl(sc, my_df)***
- ❑ How to manipulate Spark DataFrames for data cleaning and preprocessing through ***dplyr*** functions
- ❑ How to fit statistical and machine learning models using R notebook to Spark DataFrame in a truly parallel manner
- ❑ How to collect information from Spark DataFrames back to a local R object (i.e. local R data frame) for future analysis
- ❑ List of functions in ***Sparklyr***: http://spark.rstudio.com/reference/index.html

THANK YOU!