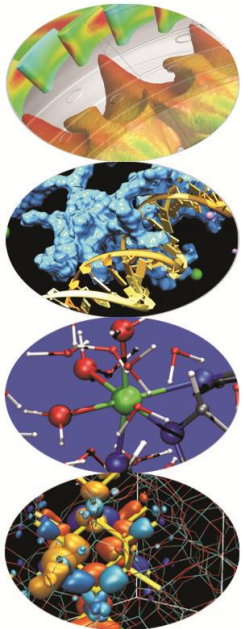


# Hadoop MapReduce and Spark

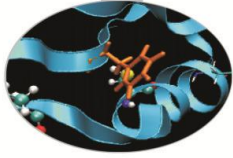
Giorgio Pedrazzi, CINECA-SCAI

*School of Data Analytics and Visualisation*

*Milan, 10/06/2015*

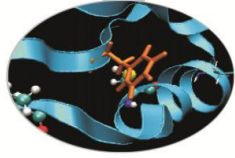


# Outline



- Hadoop
  - Hadoop
  - Import data on Hadoop
- Spark
  - Spark features
  - Scala
- MLlib
  - MLlib algorithms
  - MLlib
  - MLlib on PICO

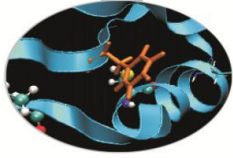
# Hadoop



Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are commonplace and thus should be automatically handled in software by the framework.

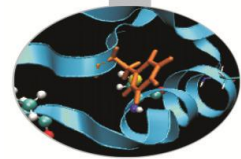
The core of Apache Hadoop consists of a storage part (Hadoop Distributed File System (HDFS)) and a processing part (MapReduce).

# Hadoop



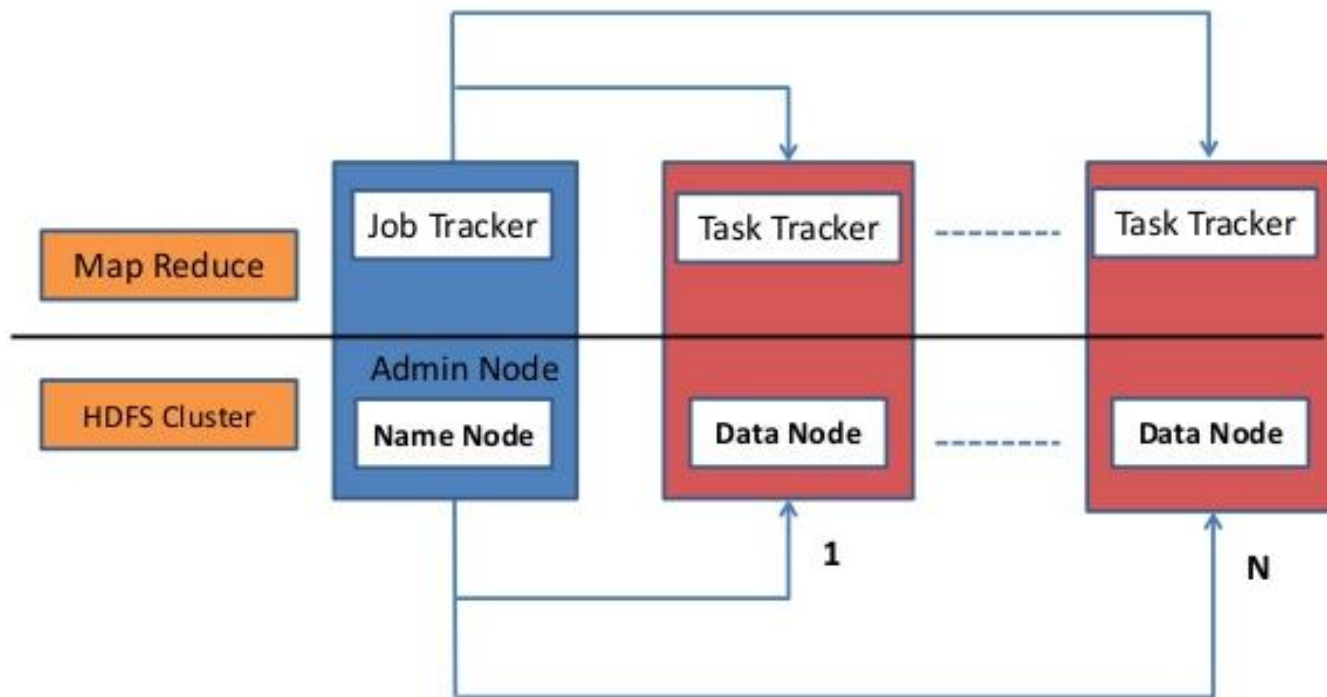
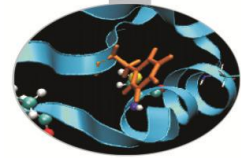
Hadoop splits files into large blocks and distributes them amongst the nodes in the cluster. To process the data, Hadoop MapReduce transfers packaged code for nodes to process in parallel, based on the data each node needs to process. This approach takes advantage of data locality (nodes manipulating the data that they have on hand) to allow the data to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are connected via high-speed networking.

# Hadoop Core Components

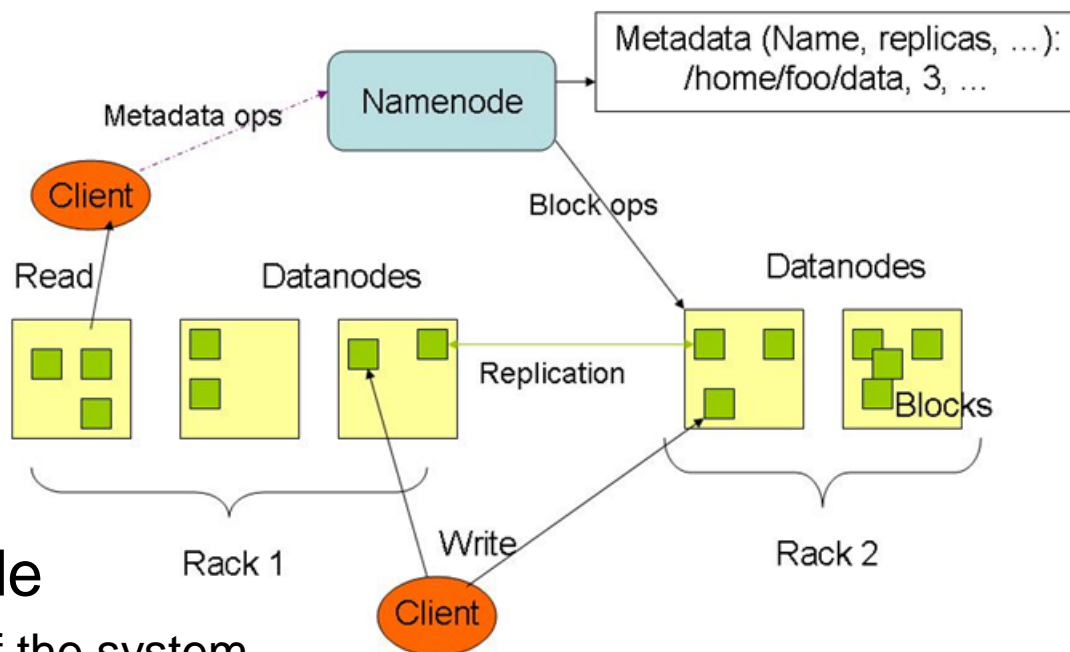
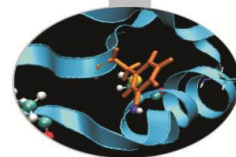


- HDFS –Hadoop Distributed File System(Storage)
  - Distributed across “nodes”
  - Natively redundant
  - Name Node tracks locations.
- MapReduce (Processing)
  - Splits a task across processors
  - “near” the data & assembles results
  - Self-Healing, High Bandwidth
  - Clustered storage
  - JobTracker manages the TaskTrackers

# Hadoop Core Components

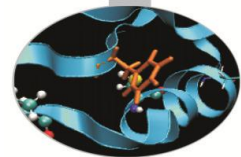


# HDFS Architecture



- **NameNode**
  - master of the system
  - maintains and manages the blocks which are present on the DataNodes
- **DataNodes**
  - slaves which are deployed on each machine and provide the actual storage
  - responsible for serving read and write requests for the clients

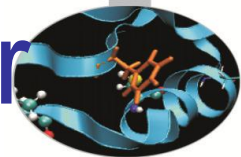
# NameNode Metadata



- Metadata in Memory
  - The entire Metadata is in main memory
  - No demand paging of FS meta-data
- Types of Metadata
  - List of files
  - List of Blocks for each file
  - List of DataNode for each block
  - File attributes, e.g. access time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc.
- Secondary NameNode:
  - Not a hot standby for the NameNode
  - Connects to NameNode every hour\*
  - Housekeeping, backup of NameNode metadata
  - Saved metadata can build a failed NameNode

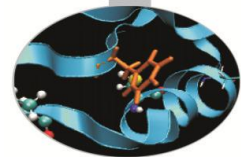


# JobTracker and TaskTracker



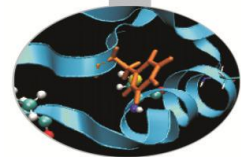
- JobTracker
  - Manages MapReduce jobs, distribute individual tasks (map/reduce) to machines running the...
- TaskTracker
  - Instantiates and monitors individual Map and Reduce tasks
  - When a TaskTracker receives a request to run a task, it instantiates a separate JVM for that task
    - Can run multiple tasks at the same time depending on the hardware resources

# JobTracker



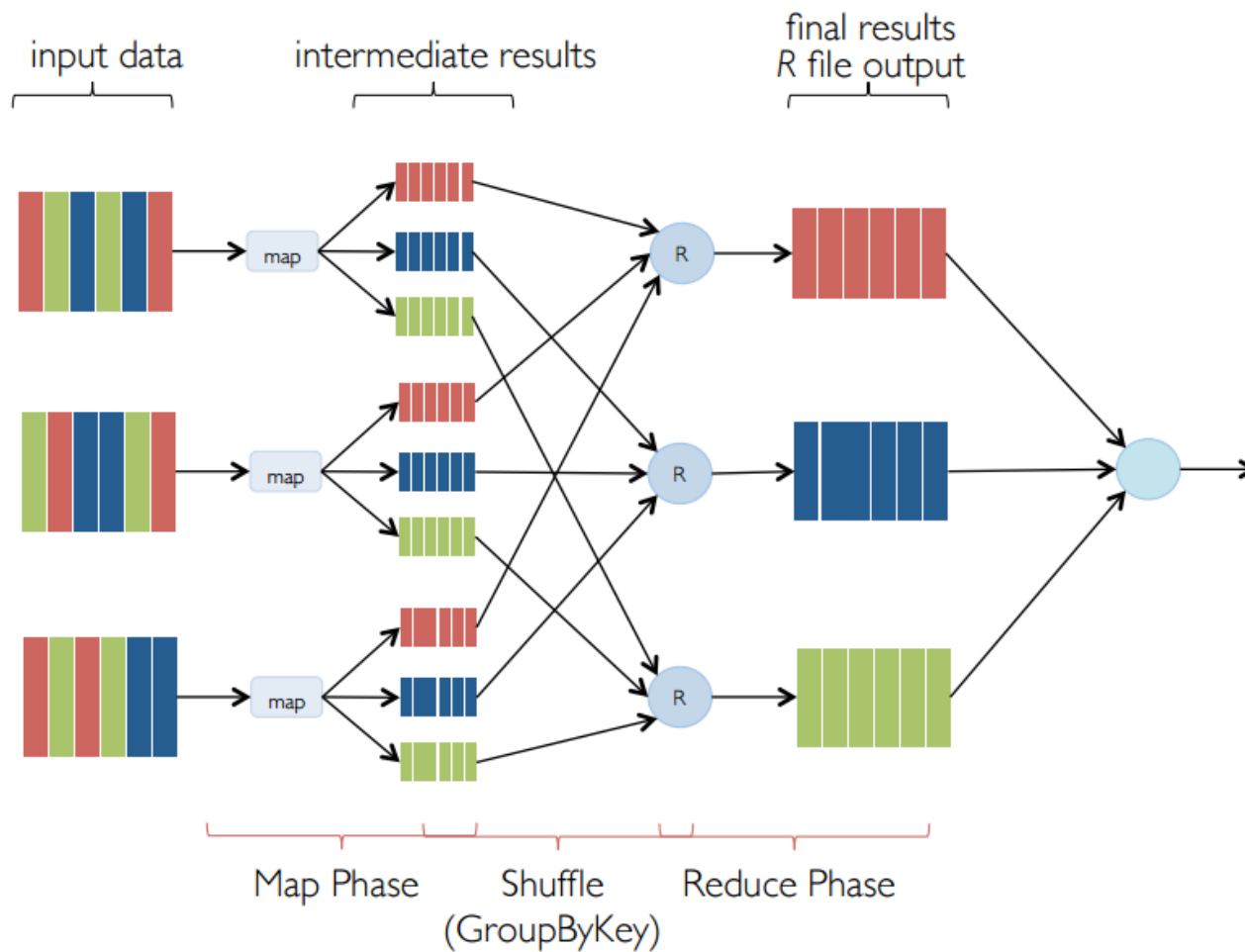
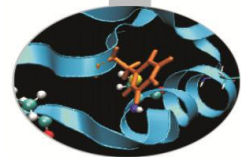
- JobTracker takes care of:
  - task status: (idle, in-progress, completed)
  - scheduling idle tasks as resources (managed by taskTrackers) become available
  - gathering location and size of each intermediate file produced by the Map tasks
  - sending this info to the reducer tasks
- JobTracker pings taskTrackers periodically to detect failures:
  - if a Map failure occurs:
    - Map tasks completed or in-progress are reset to idle
    - Reduce tasks are notified when the map task is rescheduled on another taskTracker
  - if Reduce failure occurs:
    - Only in-progress tasks are reset to idle – JobTracker failure
    - MapReduce task is aborted and client is notified

# Hadoop MapReduce

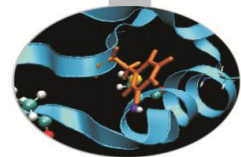


- MapReduce is an high-level programming model and implementation for largescale parallel data processing.
- A MapReduce program consists of two functions (inspired by primitives of functional programming language):
  - **MAP function:**
    - Input: (input key, value)
    - Output: bag of (intermediate key, value)
  - **REDUCE function:**
    - Input: (intermediate key, bag of values)
    - Output: bag of output (values)
- System executes the program in two steps:
  - **the MAP function** is applied in parallel to all (input key, value) pairs in the input file
  - the system will group all pairs with the same intermediate key (“shuffle”), and passes the bag of values to **the REDUCE function**

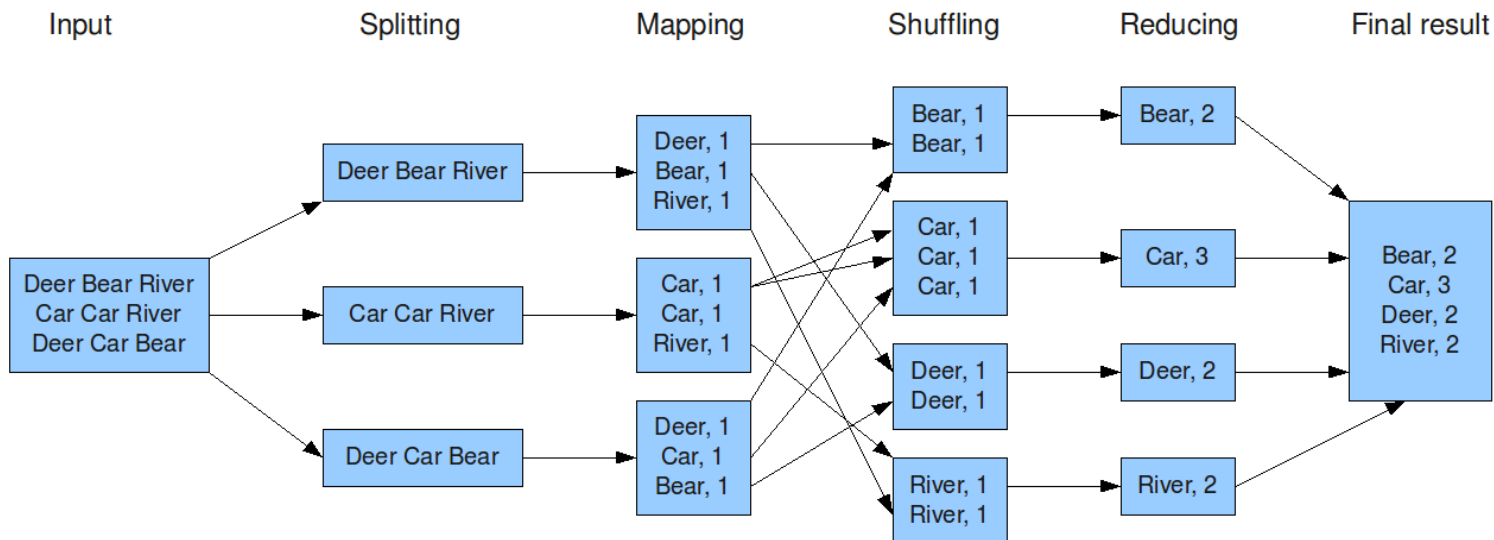
# Word count example

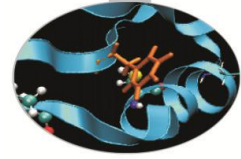


# Word count example



The overall MapReduce word count process



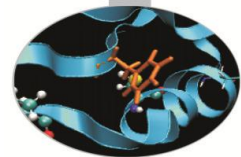


# Importing data in Hadoop

## Copy Commands

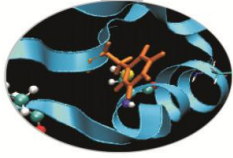
- **put**: Copy file(s) from local file system to destination file system. It can also read from “stdin” and writes to destination file system.
- `hadoop dfs-put iris.txt hdfs://<target Namenode>`
- **copyFromLocal**: Similar to “put” command, except that the source is restricted to a local file reference.
- `hadoop dfs-copyFromLocal iris.txt hdfs://<target Namenode>`
- **distcp**: Distributed Copy to move data between clusters, used for backup and recovery.
- `hadoop distcphdfs://<source NN> hdfs://<target NN>`
-

# Sqoop



- Apache Sqoop(TM) is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured data stores such as relational databases.
- Imports individual tables or entire databases to HDFS.
- Generates Java classes to allow you to interact with your imported data.
- Provides the ability to import from SQL databases straight into your Hive data warehouse

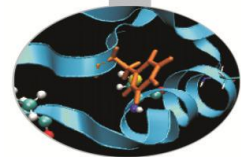
# Apache Spark



- Apache Spark is a fast and general engine for large-scale data processing.
- Apache Spark is a multi-purpose platform for use cases that span investigative, as well as operational analytics.
- Apache Spark is an execution engine that broadens the type of computing workloads Hadoop can handle, while also tuning the performance of the big data framework.
- Apache Spark has numerous advantages over Hadoop's MapReduce execution engine, in both the speed with which it carries out batch processing jobs and the wider range of computing workloads it can handle.

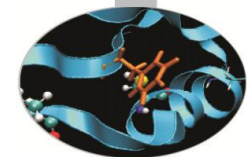


# Apache Spark Features



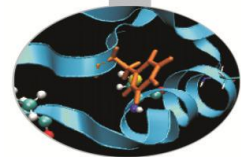
- Spark has a number of features that make it a compelling cross over platform for investigative as well as operational analytics:
- Spark comes with a Machine-Learning Library, MLlib
- Being Scala-based, Spark embeds in any JVM-based operational system, but can also be used interactively in a way that will feel familiar to R and Python users.
- For Java programmers, Scala still presents a learning curve. But at least, any Java library can be used from within Scala.
- Spark's RDD (Resilient Distributed Dataset) abstraction resembles Crunch's PCollection, which has proved a useful abstraction in Hadoop that will already be familiar to Crunch developers.(Crunch can even be used on top of Spark.)
- Spark imitates Scala's collections API and functional style, which is a boon to Java and Scala developers, but also some what familiar to developers coming from Python. Scala is also a compelling choice for statistical computing.

# Spark vs. MapReduce



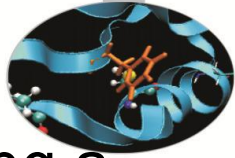
Criteria	Map Reduce	Spark
Conciseness	Plain MR has a lot of boiler plate	Almost no boilerplate
Performance	High latency	very fast compared to MR
Testability	Possible via libraries, but non trivial	Very much easy
Iterative processing	Non trivial	straight forward
Exploration of data	Not possible easily	Spark shell allows quick and easy data exploration
SQL like interface	Via Hive	Build in as SparkSQL
Fault Tolerance	Inheranlty able to handle fault tolerance via persisting the results of each of phases	Exploits immutability of RDD to enable fault tolerance
Eco system	lots of tools available but integration is not quite seamless, requiring lot of effort for their seamless integration	Unifies lot of interfaces like SQL, stream processing etc into single abstraction of RDD
In memory computations	not possible	possible

# Apache Spark Architecture



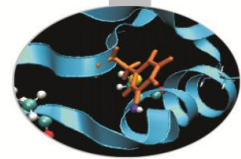
- Spark solves similar problems as Hadoop MapReduce does but with a fast in memory approach and a clean functional style API. With its ability to integrate with Hadoop and inbuilt tools for interactive query analysis(Shark), large-scale graph processing and analysis(Bagel), and real-time analysis(Spark Streaming), it can be interactively used to quickly process and query big datasets.
- Fast Data Processing with Spark covers how to write distributed mapreduce style programs with Spark.

# MapReduce issues



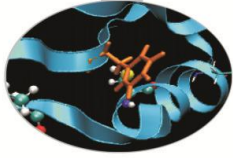
- MapReduce let users write parallel computations using a set of high-level operators without having to worry about:
  - distribution – fault tolerance • abstractions for accessing a cluster's computational resources
- • but lacks abstractions for leveraging distributed memory
- between two MR jobs writes results to an external stable storage system, e.g., HDFS ! Inefficient for an important class of emerging applications:
- • iterative algorithms – those that reuse intermediate results across multiple computations – e.g. Machine learning and graph algorithms • interactive data mining – where a user runs multiple ad-hoc queries on the same subset of the data

# Improving over MapReduce



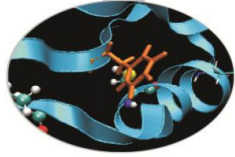
- Not relying on a rigid map-then-reduce format, its engine can execute a more general directed acyclic graph (DAG) of operators. Spark can pass them directly to the next step in the pipeline in situations where MapReduce must write out intermediate results to the distributed filesystem.
- It extends this capability with a rich set of transformations that enable users to express computation more naturally. It has a strong developer focus and streamlined API that can represent complex pipelines in a few lines of code.

# Improving over MapReduce



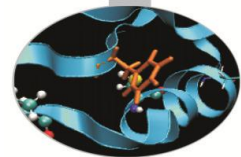
- Spark extends its predecessors with in-memory processing. Its Resilient Distributed Dataset (RDD) abstraction enables developers to materialize any point in a processing pipeline into memory across the cluster.
- Hadoop workloads are typically centered around the concept of doing batch jobs on large amounts of data, and typically it's not used for interactive querying. Spark, in contrast, has the ability to chainsaw through large amounts of data, store pared-down views of the data in structures called Resilient Distributed Datasets (RDDs), and do interactive queries with sub-second response times.

# MLlib



- MLlib is a Spark subproject providing machine learning primitives:
  - initial contribution from AMPLab, UC Berkeley
  - shipped with Spark since version 0.8
- MLlib's goal is to make practical machine learning (ML) scalable and easy. Besides new algorithms and performance improvements that we have seen in each release, a great deal of time and effort has been spent on making MLlib *easy*.

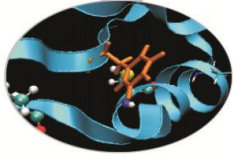
# Why MLlib



- It is built on Apache Spark, a fast and general engine for large-scale data processing.
- Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
- MLlib provides APIs in three languages: Python, Java, and Scala, along with user guide and example code, to ease the learning curve for users coming from different backgrounds

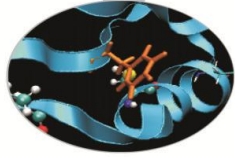


# MLlib Algorithms



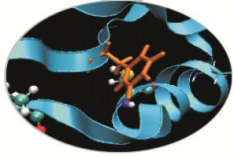
- classification: logistic regression, naive Bayes, decision tree, ensemble of trees (random forests)
- regression: generalized linear regression (GLM)
- collaborative filtering: alternating least squares (ALS)
- clustering: k-means, gaussian mixture, power iteration clustering, latent Dirichlet allocation
- decomposition: singular value decomposition (SVD), principal component analysis, singular value decomposition
- ...

# Examples with MLlib



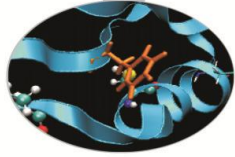
- Clustering
  - KDD 99 cup
- Decision tree
  - covtype

# KDD Cup 99 data



This is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between ``bad" connections, called intrusions or attacks, and ``good" normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment.

# Covtype data



- The data set used in this chapter is the well-known Covtype data set, available [online](#) as a compressed CSV-format data file, *covtype.data.gz*, and accompanying info file, *covtype.info*.
- The data set records the types of forest covering parcels of land in Colorado, USA. Each example contains several features describing each parcel of land, like its elevation, slope, distance to water, shade, and soil type, along with the known forest type covering the land. The forest cover type is to be predicted from the rest of the features, of which there are 54 in total.
- This data set has been used in research, and even a [Kaggle competition](#). It is an interesting data set to explore in this chapter because it contains both categorical and numeric features. There are 581,012 examples in the data set, which does not exactly qualify as big data, but is large enough to be manageable as an example and still highlight some issues of scale.