# TIẾP CẬN DỮ LIỆU LỚN

# Nội dung

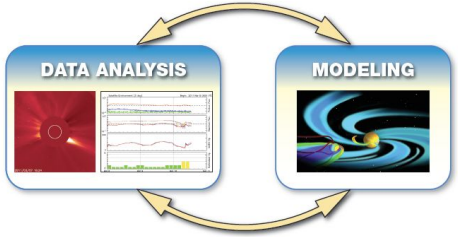
- Vấn đề trong DDL

  (Quản lý lưu trữ, Phân tích tính toán, Trực quan)
- Tính toán song song & phân tán
- Một số nền tảng và công nghệ phổ biến

# CÁC VẤN ĐỀ CHÍNH TRONG DLL

- **Quản lý DLL**: lưu trữ, bảo trì, truy cập.
- **Xử lý, Phân tích DLL**: cố gắng hiểu, khai thác thông tin hữu ích từ dữ liệu.
- **Trực quan**: hiển thị thông tin và hỗ trợ quyết định.

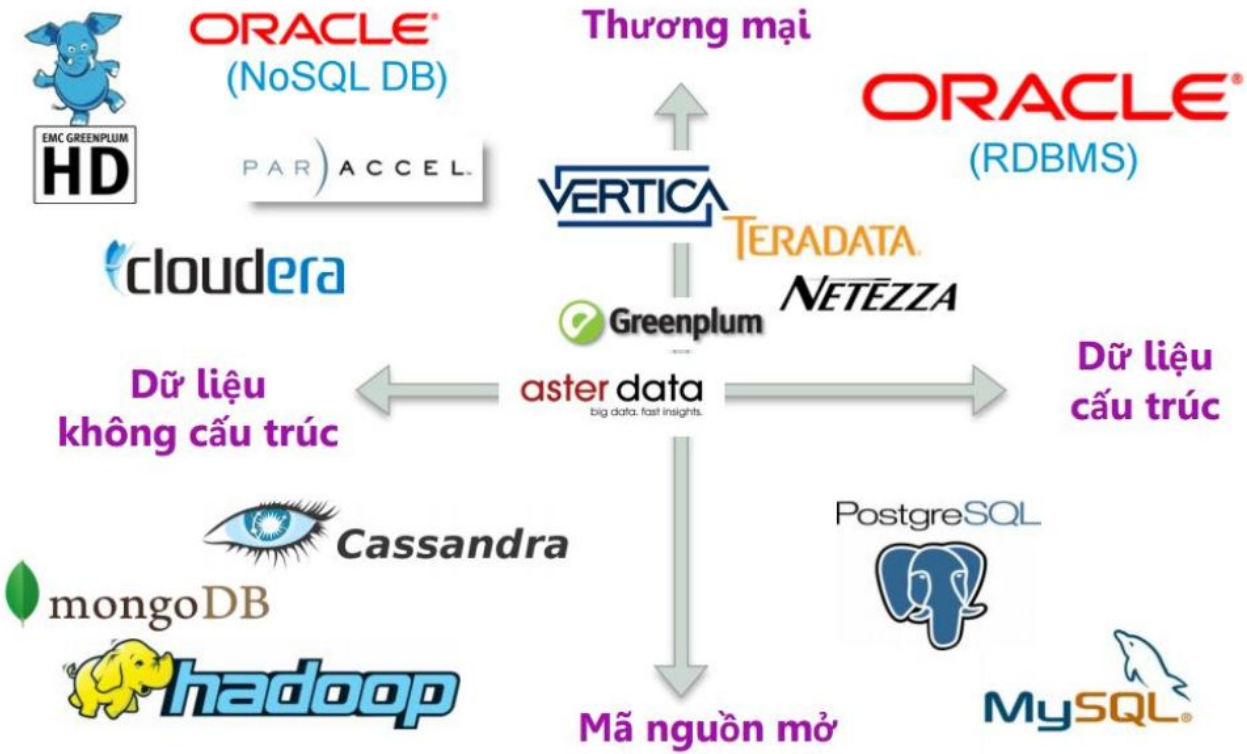
Nguồn: http://www.michellgroup.com/future-data-storage-just-really-long-cassette-tape/




Nguồn: https://pics-about-space.com/soho-nasa-data?p=2


Nguồn: https://www.sas.com/en_us/insights/big-data/data-visualization.html

# QUẢN LÝ, LƯU TRỮ DLL




**Nguồn**: Dinh Phung, 2017, VIASM 2017 Data Science Workshop.

# XỬ LÝ (PHÂN TÍCH, TÍNH TOÁN) DLL

**Kỹ thuật chính trong xử lý DLL**

- Scaling up, Scaling out
- Parallel Computing, Distributed Computing

**Một số công nghệ nguồn mở phổ biến**

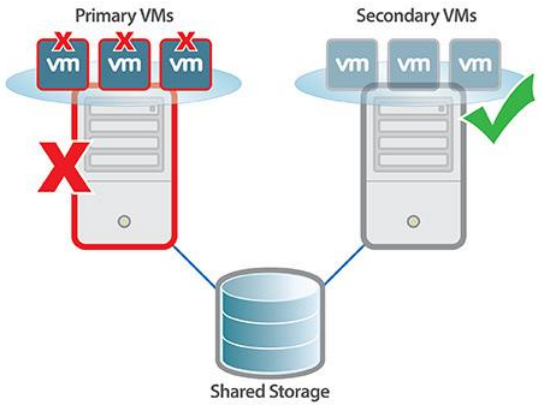
- MapReduce
- Hadoop
- Spark
- Tensor Flow

# XỬ LÝ DLL - MỘT SỐ THUẬT NGỮ

- **Kích thước dữ liệu hỗ trợ**: kích thước tập dữ liệu mà hệ thống có thể xử lý hiệu quả.
- **Xử lý thời gian thực (Real time Processing)**: khả năng xử lý dữ liệu và cho ra kết quả trong khoảng thời gian rất ngắn, gần như thức thời.
  (https://www.techopedia.com/definition/31742/real-time-data-processing)
- **Hiệu quả xuất nhập (I/O performance)**: tỷ lệ dữ liệu vào ra thiết bị ngoại vi

# XỬ LÝ DLL - MỘT SỐ THUẬT NGỮ

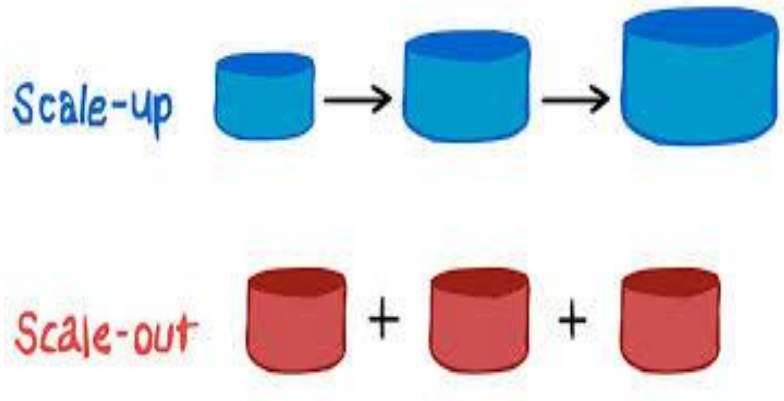
- **Khả năng chịu lỗi (Fault Tolerance):** khả năng hệ thống vẫn tiếp tục hoạt động bình thường khi có vấn đề của một hay nhiều thành phần.
- **Tính khả dụng cao (High availability)** đề cập đến khả năng giảm thiểu downtime của hệ thống trong khi hoạt động, tránh gây gián đoạn dịch vụ




Nguồn: https://www.drj.com/articles/online-exclusive/fault-tolerance-in-virtualized-data-centers.html

# XỬ LÝ DLL - Khả năng mở rộng (Scalability)

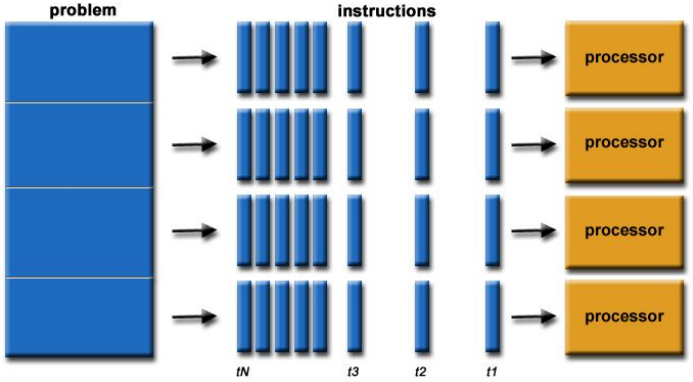
- khả năng hệ thống đối phó với sự gia tăng dữ liệu, độ phức tạp tính toán, mà không ảnh hưởng đến các dịch vụ, chức năng cốt lõi.



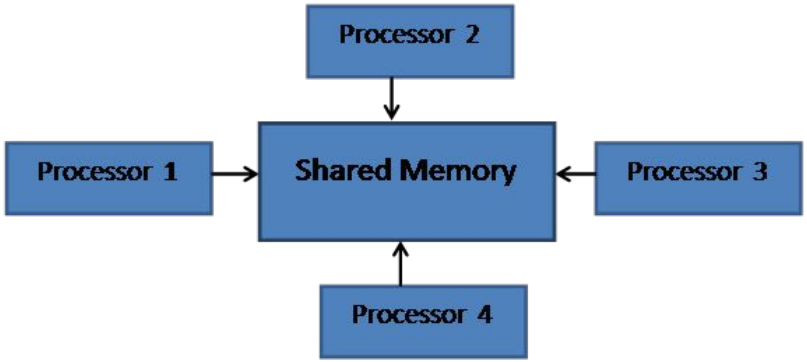

Nguồn: https://turbonomic.com/blog/on-turbonomic/to-scale-up-or-scale-out-that-is-the-question/

# XỬ LÝ DLL - Parallel Computing

- **Tính toán song song (Parallel Computing): bài toán được chia nhỏ vào các bộ xử lý để tính toán song song có cùng bộ nhớ chung.**



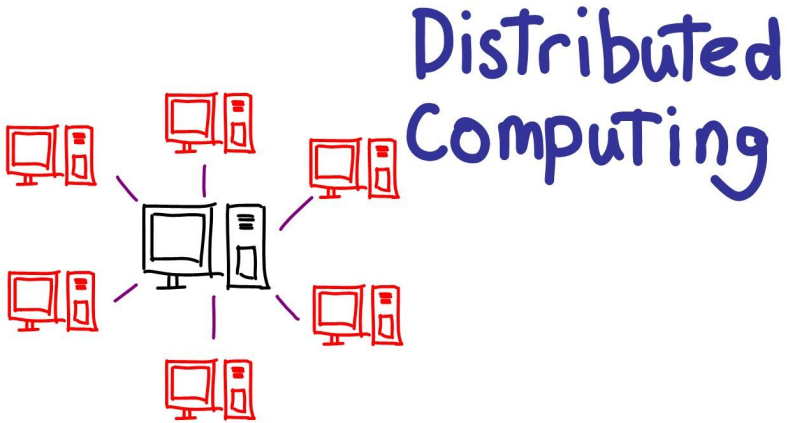

Nguồn: https://computing.llnl.gov/tutorials/parallel_comp/




Nguồn: http://pawangh.blogspot.com/2014/05/mpi-vs-openmp.html

# XỬ LÝ DLL - Distributed computing

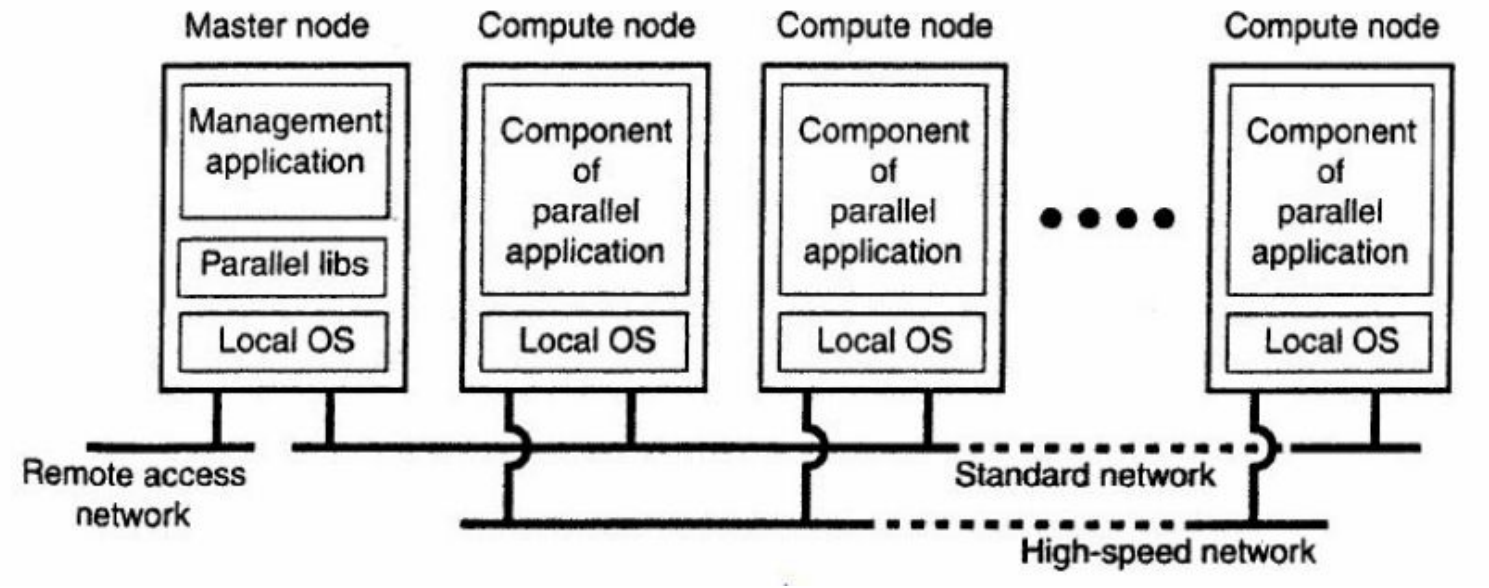
- **Tính toán phân tán: bài toán được chia nhỏ và phân vào nhiều máy khác nhau để tính toán, mỗi máy có bộ nhớ riêng.**




Nguồn: https://www.youtube.com/watch?v=YS-QvfCZWvc

# XỬ LÝ DLL - Kỹ thuật Cluster Computing

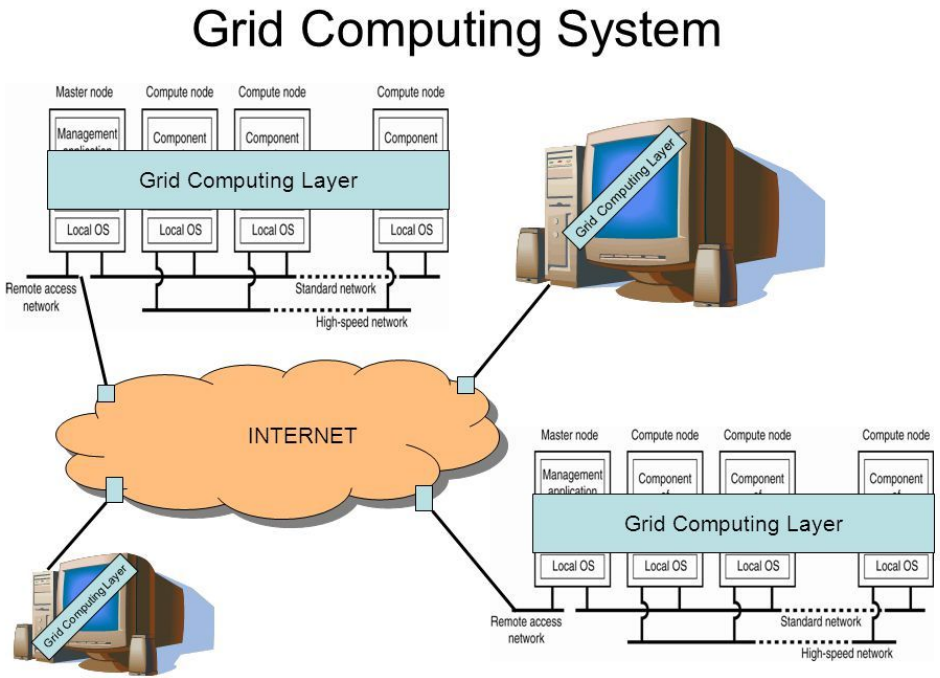
- is used for parallel programing in which a single program is run in parallel on multiple machines.




Nguồn: Tanenbaum, A. S., & Van Steen, M. (2007). Distributed systems: principles and paradigms. Prentice-Hall.

# XỬ LÝ DLL - Kỹ thuật Grid Computing

- resources from different organizations are brought together to allow collaboration of a group of people (virtual organization)

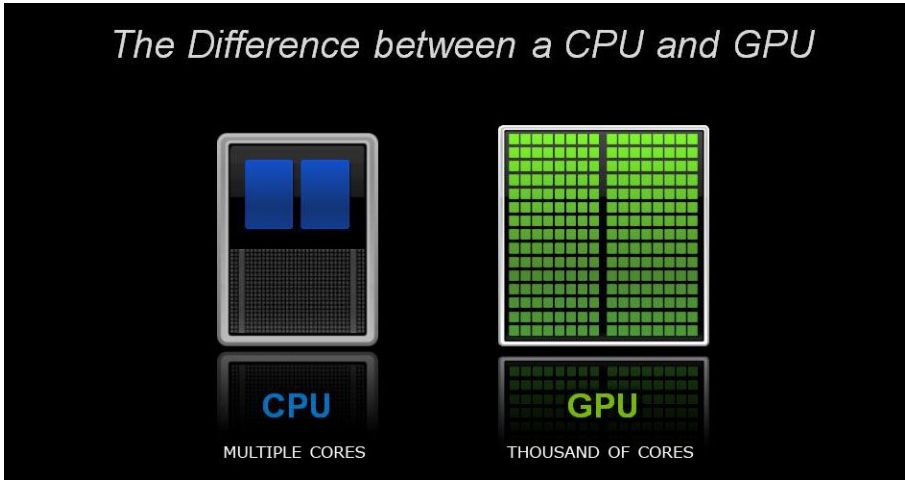


# SONG SONG VỚI CPU ĐA NHÂN

- **CPU đa nhân**
  - Dùng máy CPU nhiều cores
  - Dùng multithreading
  - Hạn chế:
    - Số lượng Cores của CPU không nhiều.
    - Bộ nhớ hiện giới hạn vài trăm GB.

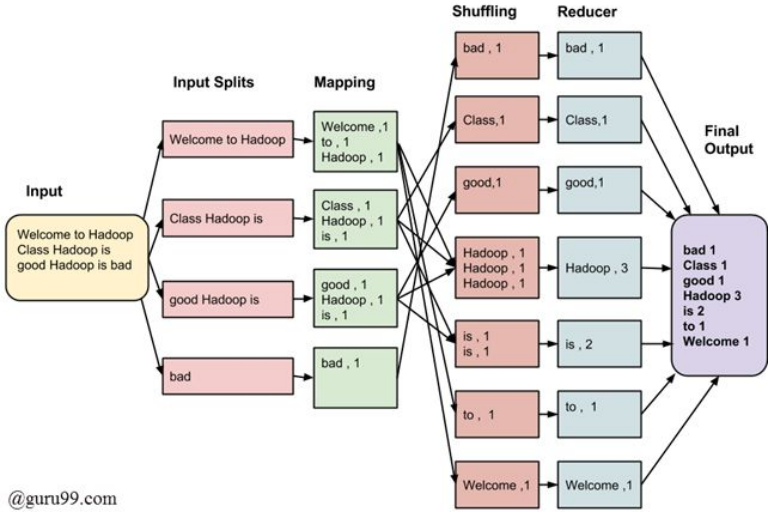
# SONG SONG VỚI GPU

- **GPU**
  - Số cores rất lớn (2K+).
  - Khả năng tăng tốc lớn hơn so với CPU đa nhân
  - Hạn chế:
    - Bộ nhớ hạn chế (khoảng 12GB trên GPU).
    - Ít thuật toán, phần mềm hỗ trợ trên GPU.



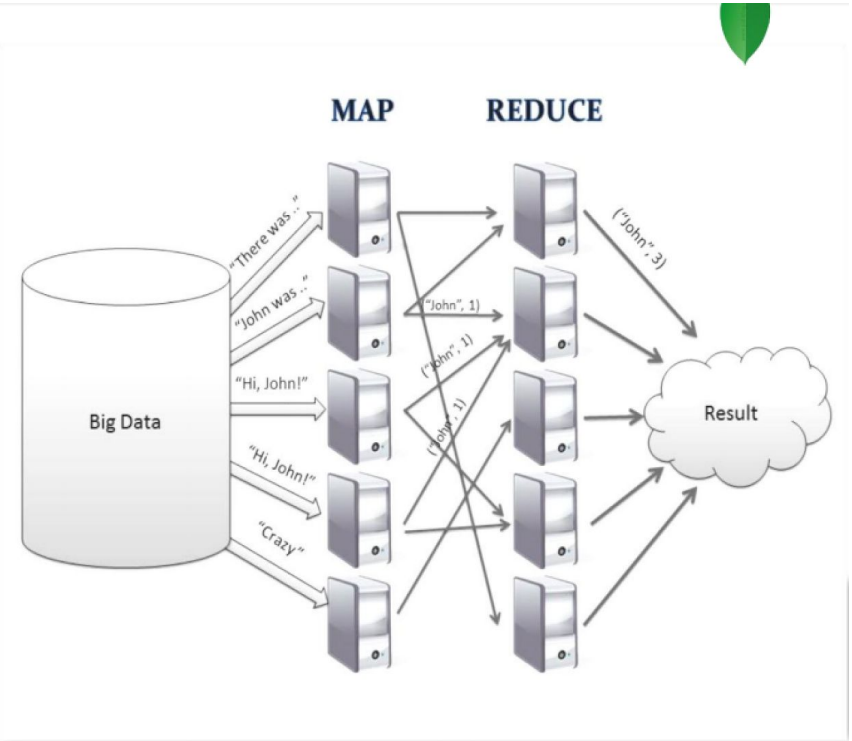
# XỬ LÝ DLL - MỘT SỐ CÔNG NGHỆ QUAN TRỌNG

- MapReduce
- Hadoop
- Spark
- TensorFlow



# NỀN TẢNG CÔNG NGHỆ

- Nhu cầu xử lý DLL tăng nhanh
- MapReduce
  - invented by Google in 2004 and used in Hadoop.
  - breaking the entire task into two parts: mappers and reducers.
  - mappers: read the data from HDFS, process it and generate some intermediate results.
  - reducers: aggregate the intermediate results to generate the final output.
- Key Limitations
  - inefficiency in running iterative algorithms.
  - Mappers read the same data again and again from the disk.

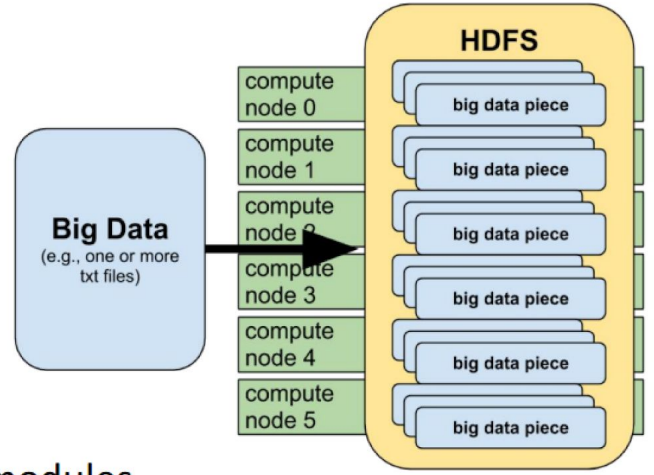



Data **parallelization** **Distributed** execution Outcome **aggregation**

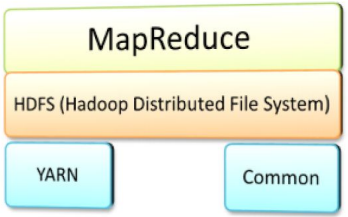
[Nguồn: https://www.supinfo.com/articles/single/2807-introduction-to-the-mapreduce-life-cycle]

# Apache Hadoop

- Apache Hadoop:
  - an open source framework for storing and processing large datasets using clusters of commodity hardware
  - highly fault tolerant
  - scaling up to 100s or 1000s of nodes

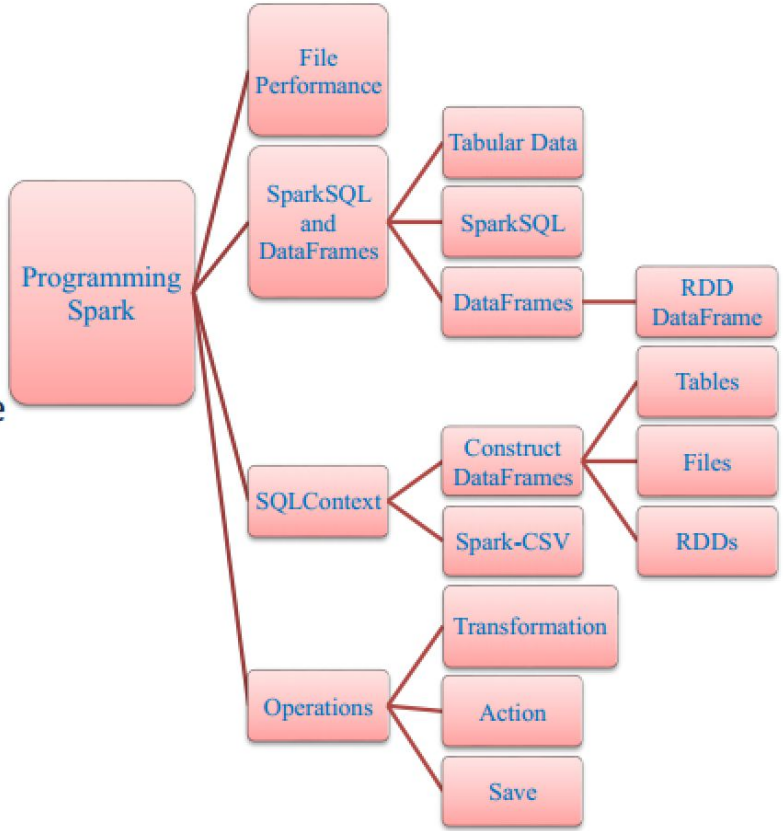



- Hadoop components:
  - **Common**: utilities that support the other Hadoop modules
  - **YARN**: a framework for job scheduling and cluster resource management.
  - **HDFS**: a distributed file system
  - **MapReduce**: computation model for parallel processing of large datasets.



# Apache Spark

- Key motivation: suitable for iterative-convergent algorithms!
- Spark key features:
  - Resilient Distributed Datasets (RDD)
    - Read-only, partitioned collection of records distributed across cluster, stored in memory or disk.
    - Data processing = graph of transforms where nodes = RDDs and edges = transforms.
  - Benefits:
    - Fault tolerant: highly resilient due to RDDs
    - Cacheable: store some RDDs in RAM, hence faster than Hadoop MR for iteration.
    - Support MapReduce.



# Spark & Hadoop

Sorted 100 TB of data on disk in 23 minutes; Previous world record set by Hadoop MapReduce used 2100 machines and took 72 minutes.

**This means that Apache Spark sorted the same data 3X faster using 10X fewer machines.**

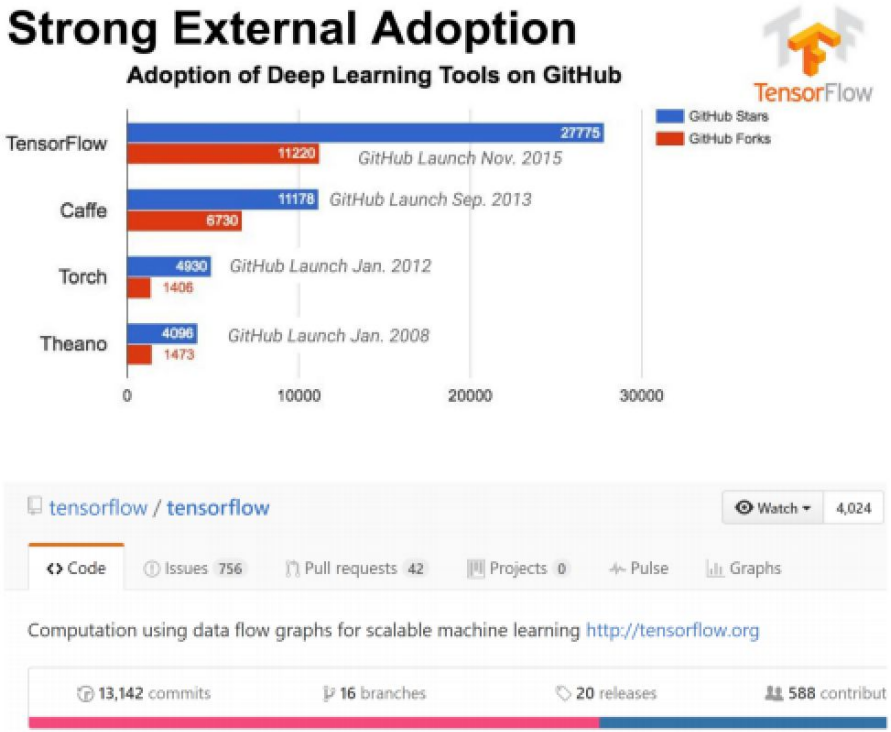
"Winning this benchmark as a general, fault-tolerant system marks an important milestone for the Spark project".

| | Hadoop MR Record | Spark Record | Spark 1 PB |
|---|---|---|---|
| Data Size | 102.5 TB | 100 TB | 1000 TB |
| Elapsed Time | 72 mins | 23 mins | 234 mins |
| # Nodes | 2100 | 206 | 190 |
| # Cores | 50400 physical | 6592 virtualized | 6080 virtualized |
| Cluster disk throughput | 3150 GB/s (est.) | 618 GB/s | 570 GB/s |
| Sort Benchmark Daytona Rules | Yes | Yes | No |
| Network | dedicated data center, 10Gbps | virtualized (EC2) 10Gbps network | virtualized (EC2) 10Gbps network |
| **Sort rate** | **1.42 TB/min** | **4.27 TB/min** | **4.27 TB/min** |
| **Sort rate/node** | **0.67 GB/min** | **20.7 GB/min** | **22.5 GB/min** |

[Nguồn: https://databricks.com]

# TensorFlow

- TensorFlow
  - open-source framework for deep learning, developed by the GoogleBrain team.
  - provides primitives for defining functions on tensors and automatically computing their derivatives.



# Mô hình thống kê & học máy cho dữ liệu lớn

MLlib

- MLlib history
  - A platform on Spark providing scalable machine learning and statistical modelling algorithms.
  - Developed from AMPLab, UC Berkeley and shipped with Spark since 2013.
- MLlib algorithms
  - Classification
    - Linear models (linear SVMs, logistic regression)
    - Naïve Bayes
    - Least squares
    - Classification tree
    - Ensembles of trees (Random Forests and Gradient-Boosted Trees)
  - Regression
    - Generalized linear models (GLMs)
    - Regression tree
    - Isotonic regression
  - Clustering
    - K-means
    - Gaussian mixture
    - Power iteration clustering (PIC)
    - Latent Dirichlet Allocation (LDA)
    - Streaming k-means
  - Collaborative filtering (recommender system)
    - Alternating least squares (ALS),
    - Non-negative matrix factorization (NMF)
  - Dimensionality reduction
    - Singular value decomposition (SVD)
    - Principal component analysis (PCA)
  - Optimization
    - SGD, L-BFGS

# Tóm tắt

1. DLL là những tập rất lớn vượt quá khả năng lưu trữ, xử lý, tính toán của những công nghệ truyền thống.
2. Dữ liệu lớn đến từ nhiều nguồn khác nhau và không ngừng biến đổi.
3. Đặc trưng chính: Volume (Kích thước), Velocity (Tốc độ), Variety (Đa dạng).
4. DLL đem lại nhiều cơ hội, đồng thời có không ít thách thức.
5. Khi làm việc với dữ liệu lớn: (1) Lưu trữ; (2) Xử lý; (3) Công cụ mô hình để phân tích.
6. Nền tảng công nghệ: MapReduce, Hadoop, Spark, TensorFlow (chủ yếu Deep Learning).

# References


1. Tanenbaum, A. S., & Van Steen, M. (2007). Distributed systems: principles and paradigms. Prentice-Hall.
2. GS. Phùng Quốc Định, Khoa học dữ liệu lớn, Mini-Course Data Science, VIASM, 2017 (http://www.jaist.ac.jp/~bao/DS2017/)