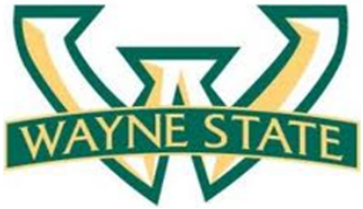


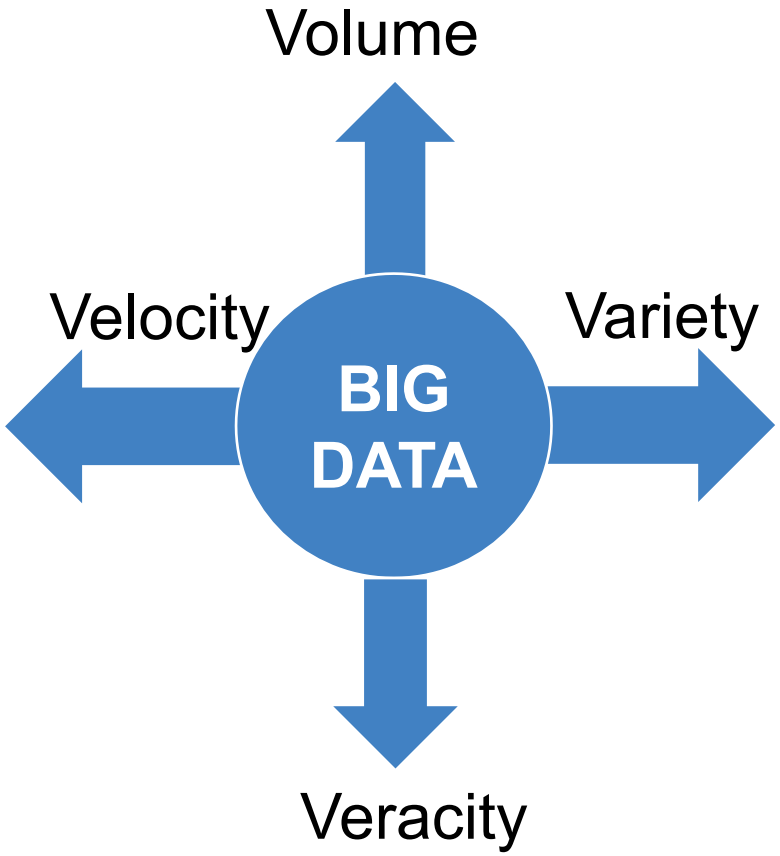
Platforms and Algorithms for Big Data Analytics

Chandan K. Reddy
Department of Computer Science
Wayne State University

http://www.cs.wayne.edu/~reddy/
http://dmkd.cs.wayne.edu/TUTORIAL/Bigdata/

What is Big Data?

A collection of large and complex data sets which are difficult to process using common database management tools or traditional data processing applications.



The four dimensions (V's) of Big Data

Big data is not just about size.

- Finds insights from complex, noisy, heterogeneous, streaming, longitudinal, and voluminous data.
- It aims to answer questions that were previously unanswered.

The challenges include capture, storage, search, sharing & analysis.

Data Accumulation !!!

- Data is being collected at rapid pace due to the advancements in sensing technologies.
- Storage has become extremely cheap and hence no one wants to throw away the data. The assumption here is that they will be using it in the future.
- Estimates show that the amount of digital data accumulated until 2010 has been gathered within the next two years. This shows the growth in the digital world.
- Analytics is still lagging behind compared to sensing and storage developments.

Why Should YOU CARE ?

- **JOBS !!**

- The U.S. could face a shortage by 2018 of 140,000 to 190,000 people with "deep analytical talent" and of 1.5 million people capable of analyzing data in ways that enable business decisions. (McKinsey & Co)
- Big Data industry is worth more than $100 billion
- Growing at almost 10% a year (roughly twice as fast as the software business)

- **Digital World is the future !!**

- The world will become more and more digital and hence big data is only going to get BIGGER !!
- This is an era of big data

Why we need more Powerful Platforms ?

- The **choice of hardware/software platform** plays a crucial role to achieve one's required goals.
- To analyze this **voluminous and complex data**, scaling up is imminent.
- In many applications, analysis tasks need to produce **results in real-time** and/or for large volumes of data.
- It is no longer possible to do real-time analysis on such big datasets using a **single machine** running commodity hardware.
- Continuous research in this area has led to the development of many **different algorithms and big data platforms.**

THINGS TO THINK ABOUT !!!!

- **Application/Algorithm-level requirements…**
  - How quickly do we need to get the results?
  - How big is the data to be processed?
  - Does the model building require several iterations or a single iteration?
- **Systems/Platform-level requirements…**
  - Will there be a need for more data processing capability in the future?
  - Is the rate of data transfer critical for this application?
  - Is there a need for handling hardware failures within the application?

Outline of this Tutorial

- **Introduction**
- **Scaling**
- **Horizontal Scaling Platforms**
  - **Peer to Peer**
  - **Hadoop**
  - **Spark**
- **Vertical Scaling Platforms**
  - **High Performance Computing (HPC) Clusters**
  - **Multicore**
  - **Graphical Processing Unit (GPU)**
  - **Field Programmable Gate Array (FPGA)**
- **Comparison of Different Platforms**
- **Big Data Analytics and Amazon EC2 Clusters**

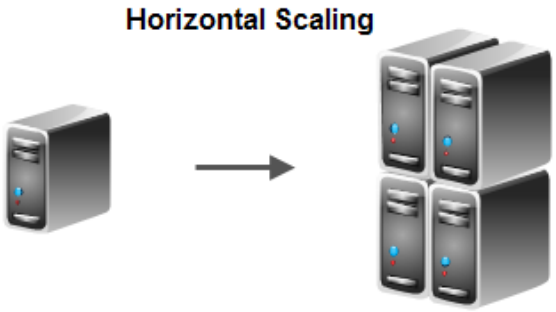
Dilpreet Singh and Chandan K. Reddy, "**A Survey on Platforms for Big Data Analytics**", *Journal of Big Data, Vol.2, No.8, pp.1-20, October 2014.*

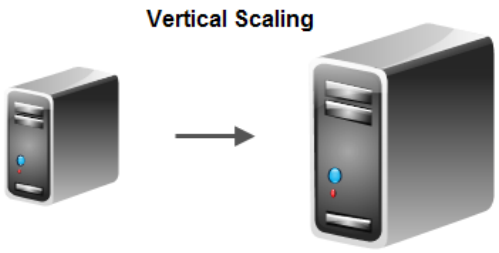
Outline

- Introduction
- Scaling
- Horizontal Scaling Platforms
  - Peer to Peer
  - Hadoop
  - Spark
- Vertical Scaling Platforms
  - High Performance Computing (HPC) clusters
  - Multicore
  - Graphical Processing Unit (GPU)
  - Field Programmable Gate Array (FPGA)
- Comparison of Different Platforms
- Big Data Analytics and Amazon EC2 Clusters

Scaling

- Scaling is the ability of the system to adapt to increased demands in terms of processing
- Two types of scaling :
  - **Horizontal Scaling**
    - Involves distributing work load across many servers
    - Multiple machines are added together to improve the processing capability
    - Involves multiple instances of an operating system on different machines
  - **Vertical Scaling**
    - Involves installing more processors, more memory and faster hardware typically within a single server
    - Involves single instance of an operating system





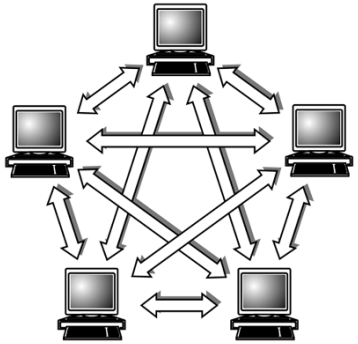
Horizontal vs Vertical Scaling

Scaling	Advantages	Drawbacks
Horizontal Scaling	→ Increases performance in small steps as needed → Financial investment to upgrade is relatively less → Can scale out the system as much as needed	→ Software has to handle all the data distribution and parallel processing complexities → Limited number of software are available that can take advantage of horizontal scaling
Vertical Scaling	→ Most of the software can easily take advantage of vertical scaling → Easy to manage and install hardware within a single machine	→ Requires substantial financial investment → System has to be more powerful to handle future workloads and initially the additional performance goes to waste → It is not possible to scale up vertically after a certain limit

Dilpreet Singh and Chandan K. Reddy, "**A Survey on Platforms for Big Data Analytics**", *Journal of Big Data, Vol.2, No.8, pp.1-20, October 2014.*

Horizontal Scaling Platforms

- Some prominent horizontal scaling platforms:
  - Peer to Peer Networks



  - Apache Hadoop

  - Apache Spark

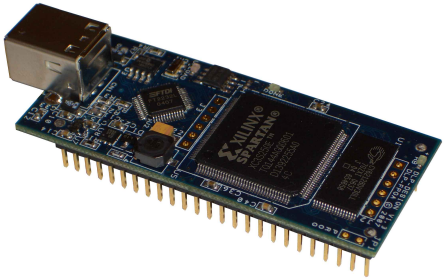
Vertical Scaling Platforms

- Most prominent vertical scaling platforms:
  - High Performance Computing Clusters (HPC)

  - Multicore Processors

  - Graphics Processing Unit (GPU)

  - Field Programmable Gate Arrays (FPGA)



Outline

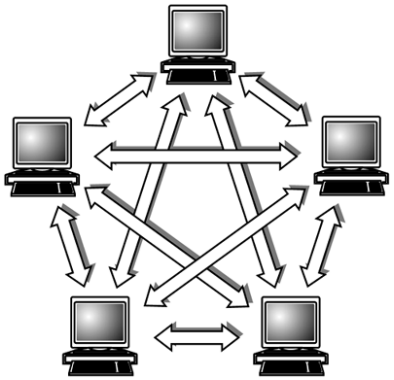
- Introduction
- Scaling
- Horizontal Scaling Platforms
  - Peer to Peer
  - Hadoop
  - Spark
- Vertical Scaling Platforms
  - High Performance Computing (HPC) clusters
  - Multicore
  - Graphical Processing Unit (GPU)
  - Field Programmable Gate Array (FPGA)
- Comparison of Different Platforms
- Big Data Analytics on Amazon EC2 Clusters

Peer to Peer Networks

- Typically involves millions of machines connected in a network
- Decentralized and distributed network architecture
- Message Passing Interface (MPI) is the communication scheme used
- Each node is capable of storing and processing data
- Scale is practically unlimited (can be millions of nodes)

Main Drawbacks

- Communication is the major bottleneck
- Broadcasting messages is cheaper but aggregation of results/data is costly
- Poor Fault tolerance mechanism



Apache Hadoop

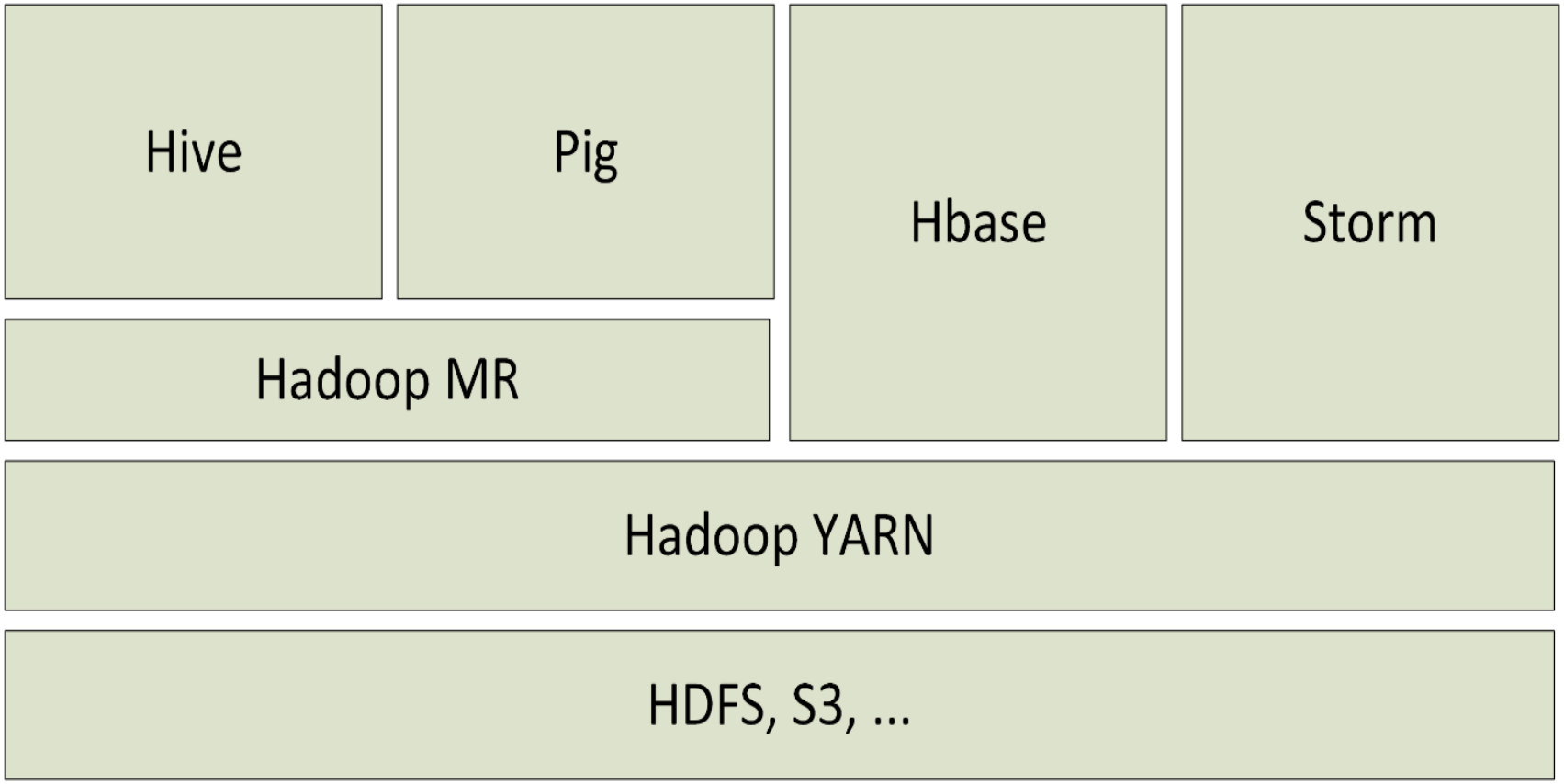
- Open source framework for storing and processing large datasets
- High fault tolerance and designed to be used with commodity hardware

- Consists of two important components:

  - HDFS (Hadoop Distributed File System)
    - Used to store data across cluster of commodity machines while providing high availability and fault tolerance

  - Hadoop YARN
    - Resource management layer
    - Schedules jobs across the cluster

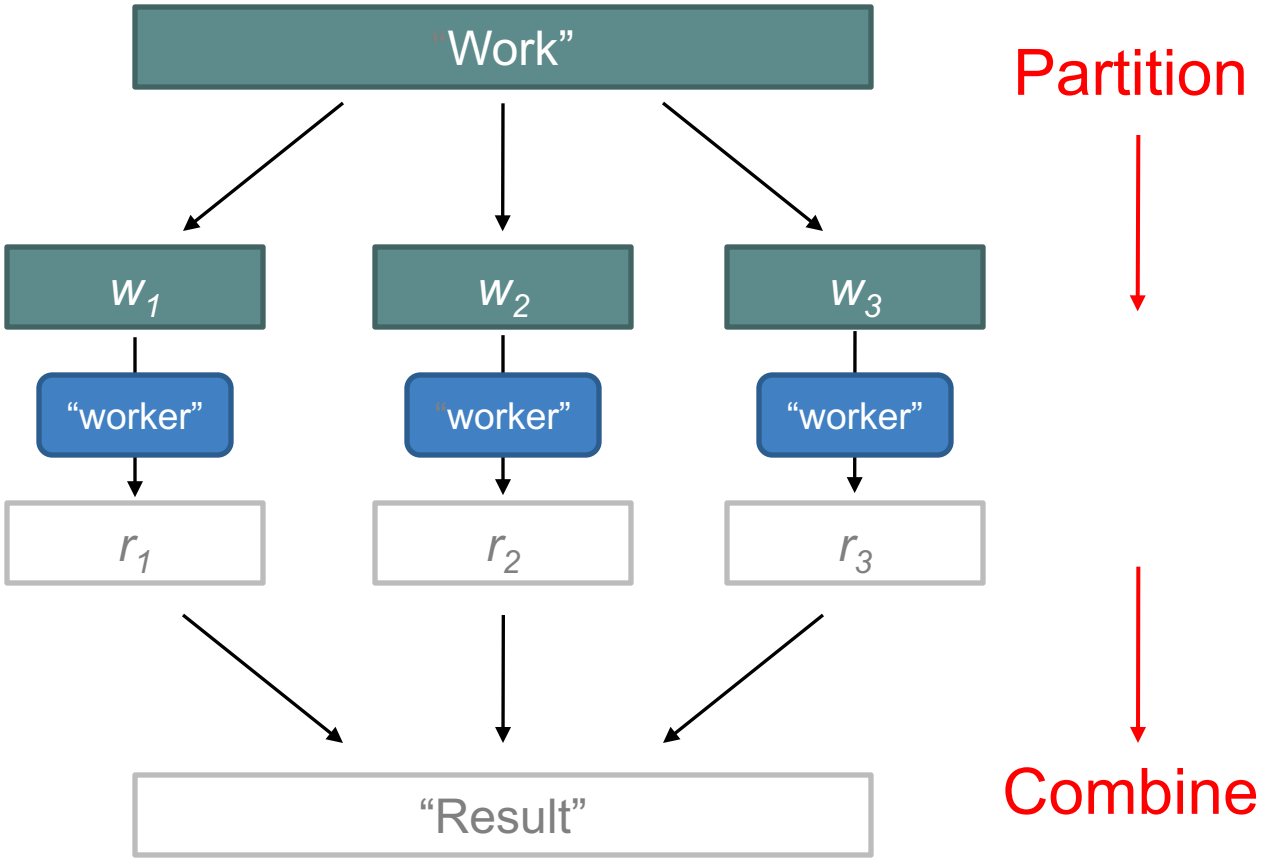
Hadoop Architecture



Hadoop MapReduce

- Basic data processing scheme used in Hadoop
- Includes breaking the entire scheme into mappers and reducers
  - Mappers read data from HDFS, process it and generate some intermediate results
  - Reducers aggregate the intermediate results to generate the final output and write it to the HDFS
- Typical Hadoop job involves running several mappers and reducers across the cluster

Divide and Conquer Strategy



MapReduce Wrappers

- Provide better control over MapReduce code
- Aid in code development
- Popular map reduce wrappers include:
  - Apache Pig
    - SQL like environment developed at Yahoo
    - Used by many organizations including Twitter, AOL, LinkedIn and more

  - Hive
    - Developed by Facebook

Both these wrappers are intended to make code development easier without having to deal with the complexities of MapReduce coding

Spark

- Next generation paradigm for big data processing
- Developed by researchers at University of California, Berkeley
- Used as an alternative to Hadoop
- Designed to overcome disk I/O and improve performance of earlier systems
- Allows data to be cached in memory eliminating the disk overhead of earlier systems
- Supports Java, Scala and Python
- Can yield upto 100x faster than Hadoop MapReduce

Outline

- Introduction
- Scaling
- Horizontal Scaling Platforms
  - Peer to Peer
  - Hadoop
  - Spark
- Vertical Scaling Platforms
  - High Performance Computing (HPC) clusters
  - Multicore
  - Graphical Processing Unit (GPU)
  - Field Programmable Gate Array (FPGA)
- Comparison of Different Platforms
- Big Data Analytics and Amazon EC2 Clusters

High Performance Computing (HPC) Clusters

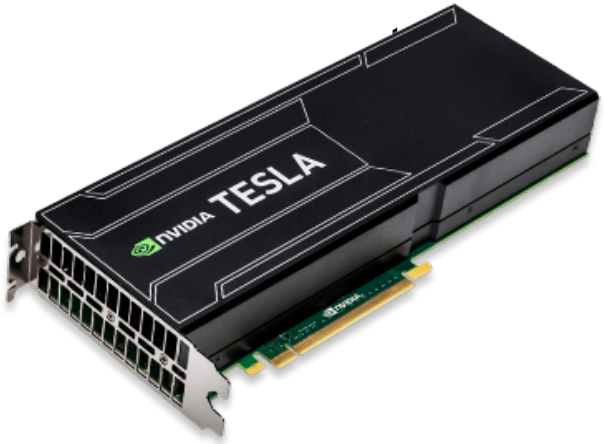
- Also known as Blades or supercomputers with thousands of processing cores
- Can have different variety of disk organization and communication mechanisms
- Contains well-built powerful hardware optimized for speed and throughput
- Fault tolerance is not critical because of top quality high-end hardware
- Not as scalable as Hadoop or Spark but can handle terabytes of data
- High initial cost of deployment
- Cost of scaling up is high
- MPI is typically the communication scheme used

Multicore CPU

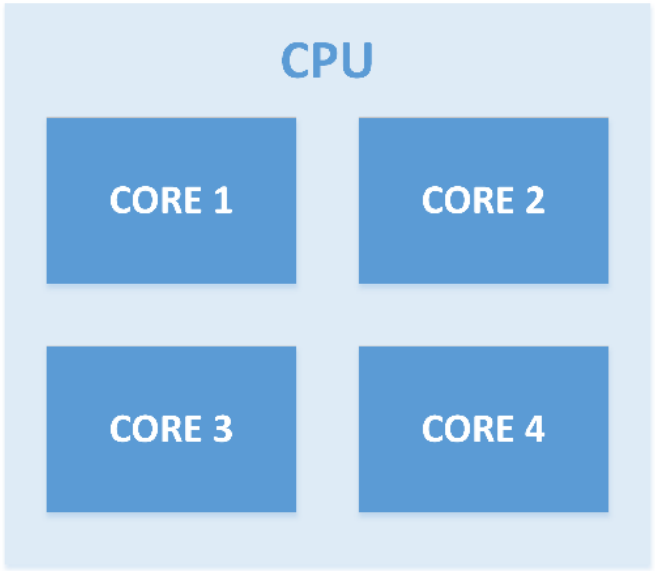
- One machine having dozens of processing cores
- Number of cores per chip and number of operations a core can perform has increased significantly
- Newer breed of motherboards allow multiple CPUs within a single machine
- Parallelism achieved through multithreading
- Task has to be broken into threads

Graphics Processing Unit

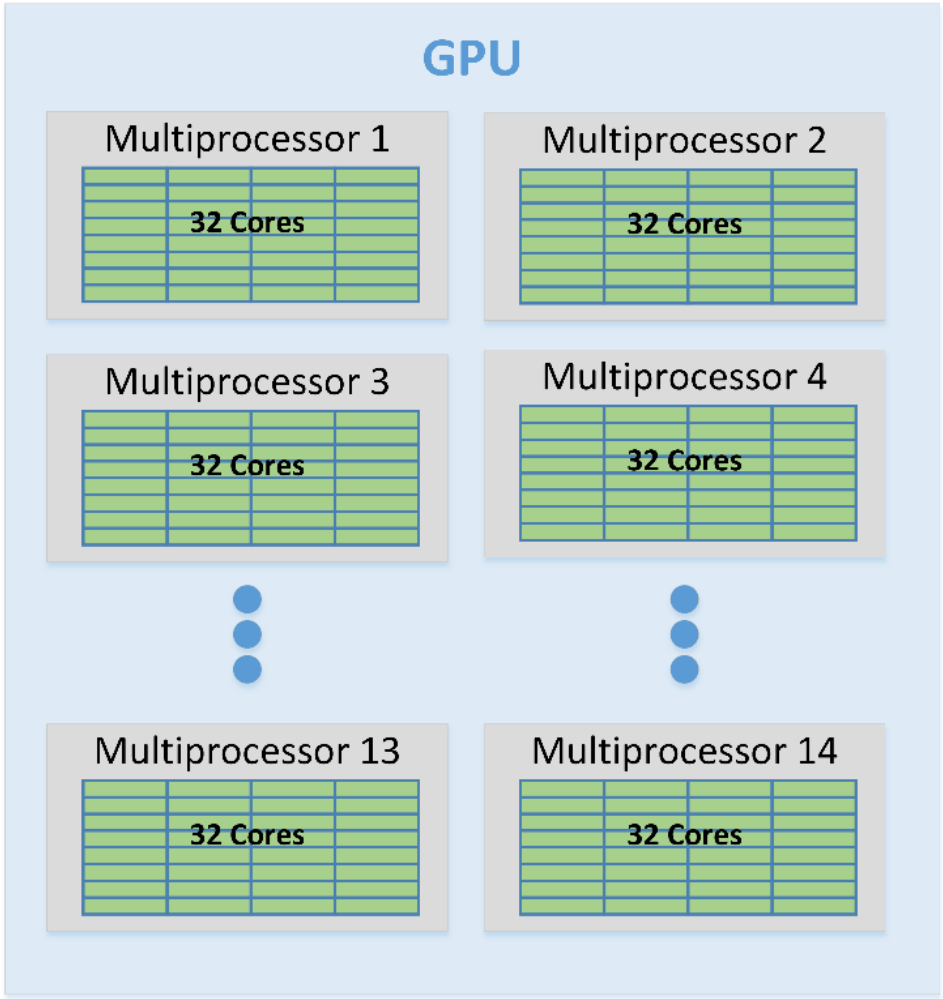
- Specialized hardware with massively parallel architecture
- Recent developments in GPU hardware and programming frameworks has given rise to GPGPU (general purpose computing on graphics processing units)
- Has large number of processing cores (typically around 2500+ currently)
- Has it's own DDR5 memory which is many times faster than typical DDR3 system memory
- Nvidia CUDA is the programming framework which simplifies GPU programming
- Using CUDA, one doesn't have to deal with low-level hardware details



CPU vs GPU Architecture



(a)



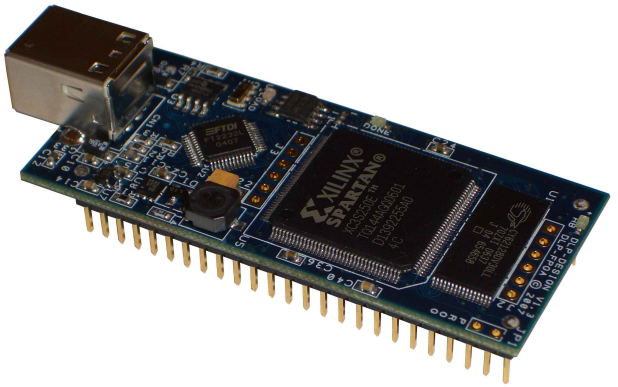
(b)

CPU vs GPU

- Development in CPU is rather slow as compared with GPU
- Number of cores in CPU is still in double digits while a GPU can have 2500+ cores
- Processing power of a current generation CPU is close to 10 Gflops while GPU can have close to 1000 Gflops of computing power
- CPU primarily relies on system memory which is slower than the GPU memory
- While GPU is an appealing option for parallel computing, the number of softwares and applications that take advantage of the GPU is rather limited
- CPU has been around for many years and huge number of software are available which use multicore CPUs

Field Programmable Gate Arrays (FPGA)

- Highly specialized hardware units
- Custom built for specific applications
- Can be highly optimized for speed
- Due to customized hardware, development cost is much higher
- Coding has to be done in HDL (Hardware Description Language) with low level knowledge of hardware
- Greater algorithm development cost
- Suited for only certain set of applications



Outline

- Introduction
- Scaling
- Horizontal Scaling Platforms
  - Peer to Peer
  - Hadoop
  - Spark
- Vertical Scaling Platforms
  - High Performance Computing (HPC) clusters
  - Multicore
  - Graphical Processing Unit (GPU)
  - Field Programmable Gate Array (FPGA)
- Comparison of Different Platforms
- Big Data Analytics and Amazon EC2 Clusters

Comparison of Different Platforms

- Following characteristics are used for comparison:
  - System/Platform dependent
    - Scalability
    - Data I/O performance
    - Fault tolerance
  - Application/Algorithm dependent
    - Real-time processing
    - Data size support
    - Support for iterative tasks

- Comparison is done using the star ratings
- 5 stars correspond to highest possible rating
- 1 star is the lowest possible rating

Comparison of Big Data Platforms

Platforms (Communication Scheme)	System/Platform			Application/Algorithm		
	Scalability	Data I/O Performance	Fault Tolerance	Real-Time Processing	Data Size Supported	Iterative Task Support
Peer to Peer (TCP/IP)	★★★★★	★	★	★	★★★★★	★★
Virtual Clusters (MapRedce/MPI)	★★★★★	★★	★★★★★	★★	★★★★	★★
Virtual Clusters (Spark)	★★★★★	★★★	★★★★★	★★	★★★★	★★★
HPC Clusters (MPI/Mapreduce)	★★★	★★★★	★★★★	★★★	★★★★	★★★★
Multicore (Multithreading)	★★	★★★★	★★★★	★★★	★★	★★★★
GPU (CUDA)	★★	★★★★★	★★★★	★★★★★	★★	★★★★
FPGA (HDL)	★	★★★★★	★★★★	★★★★★	★★	★★★★

Dilpreet Singh and Chandan K. Reddy, "**A Survey on Platforms for Big Data Analytics**", *Journal of Big Data, Vol.2, No.8, pp.1-20, October 2014.*

Scalability

- Ability of the system to handle growing amount of work load in a capable manner or to be enlarged to accommodate that growth.
- It is the ability to add more hardware to improve the performance and capacity of the system

Platforms (Communication Scheme)	System/Platform Scalability
Peer to Peer (TCP/IP)	★★★★★
Virtual Clusters (MapRedce/MPI)	★★★★★
Virtual Clusters (Spark)	★★★★★
HPC Clusters (MPI/Mapreduce)	★★★
Multicore (Multithreading)	★★
GPU (CUDA)	★★
FPGA (HDL)	★

Highly scalable and it is relatively easy to add machines and extend them to any extent

Can only scale up to a certain extent

Limited number of GPUs and CPUs in a single machine

Once deployed, scaling up becomes costly

Data I/O Performance

- The rate at which the data is transferred to/from a peripheral device. In the context of big data analytics, this can be viewed as the rate at which the data is read and written to the memory (or disk) or the data transfer rate between the nodes in a cluster.

Platforms (Communication Scheme)	System/Platform Data I/O
Peer to Peer (TCP/IP)	★
Virtual Clusters (MapRedce/MPI)	★★
Virtual Clusters (Spark)	★★★
HPC Clusters (MPI/Mapreduce)	★★★★
Multicore (Multithreading)	★★★★
GPU (CUDA)	★★★★★
FPGA (HDL)	★★★★★

Disk access and slow network communication

Slower disk access

Uses system memory; minimizes disk access

Uses system memory; usually within a single machine

Use DDR5 memory which is faster than system memory

Fault Tolerance

- The characteristic of a system to continue operating properly in the event of a failure of one or more components

Platforms (Communication Scheme)	System/Platform Fault Tolerance
Peer to Peer (TCP/IP)	★
Virtual Clusters (MapRedce/MPI)	★★★★★
Virtual Clusters (Spark)	★★★★★
HPC Clusters (MPI/Mapreduce)	★★★★
Multicore (Multithreading)	★★★★
GPU (CUDA)	★★★★
FPGA (HDL)	★★★★

Have no fault tolerance mechanism and use of commodity hardware makes them highly susceptible to system failures

Have in-built efficient fault tolerance mechanism

Although these platforms do not have state-of-the-art fault tolerance mechanisms, these have most reliable and well-built hardware which makes hardware failure an extremely rare event

Real-Time Processing

- The system's ability to process the data and produce the results strictly within certain time constraints

Platforms (Communication Scheme)	Application/ Algorithm
	Real-Time
Peer to Peer (TCP/IP)	★
Virtual Clusters (MapRedce/MPI)	★★
Virtual Clusters (Spark)	★★
HPC Clusters (MPI/Mapreduce)	★★★
Multicore (Multithreading)	★★★
GPU (CUDA)	★★★★★
FPGA (HDL)	★★★★★

Slow for real-time data processing because of network overhead and commodity hardware

Slow in terms of data I/O and do not contain optimized and powerful hardware

Have reasonable real-time processing capabilities. They have many processing cores and high memory bandwidth

Well suited for real-time processing with thousands of processing cores and very high speed memory

Data Size Supported

- The size of the dataset that a system can process and handle efficiently

Platforms (Communication Scheme)	Application/ Algorithm Data Size
Peer to Peer (TCP/IP)	★★★★★
Virtual Clusters (MapRedce/MPI)	★★★★
Virtual Clusters (Spark)	★★★★
HPC Clusters (MPI/Mapreduce)	★★★★
Multicore (Multithreading)	★★
GPU (CUDA)	★★
FPGA (HDL)	★★

Can handle Petabytes of data and can scale out to unlimited number of nodes

Can handle around several Terabytes of data

Not suited for large-scale datasets. Multicore relies on system memory which can only be up to few hundred Gigabytes. Similarly, GPU has limited on-board memory.

Iterative Task Support

- This is the ability of a system to efficiently support iterative tasks. Since many of the data analysis tasks and algorithms are iterative in nature, it is an important metric to compare different platforms, especially in the context of big data analytics

Platforms (Communication Scheme)	Application/ Algorithm
	Iterative Tasks
Peer to Peer (TCP/IP)	★★
Virtual Clusters (MapRedce/MPI)	★★
Virtual Clusters (Spark)	★★★
HPC Clusters (MPI/Mapreduce)	★★★★
Multicore (Multithreading)	★★★★
GPU (CUDA)	★★★★
FPGA (HDL)	★★★★

P2P has huge network communication overhead; MapReduce has disk I/O overhead

Reduces the disk I/O overhead

All these other platforms are suited for iterative processing. All the iterative algorithms cannot be easily modified for each of these platforms

Outline

- Introduction
- Scaling
- Horizontal Scaling Platforms
  - Peer to Peer
  - Hadoop
  - Spark
- Vertical Scaling Platforms
  - Graphical Processing Unit (GPU)
  - Multicore
  - High Performance Computing (HPC) clusters
  - Field Programmable Gate Array (FPGA)
- Comparison of Different Platforms
- Big Data Analytics and Amazon EC2 Clusters

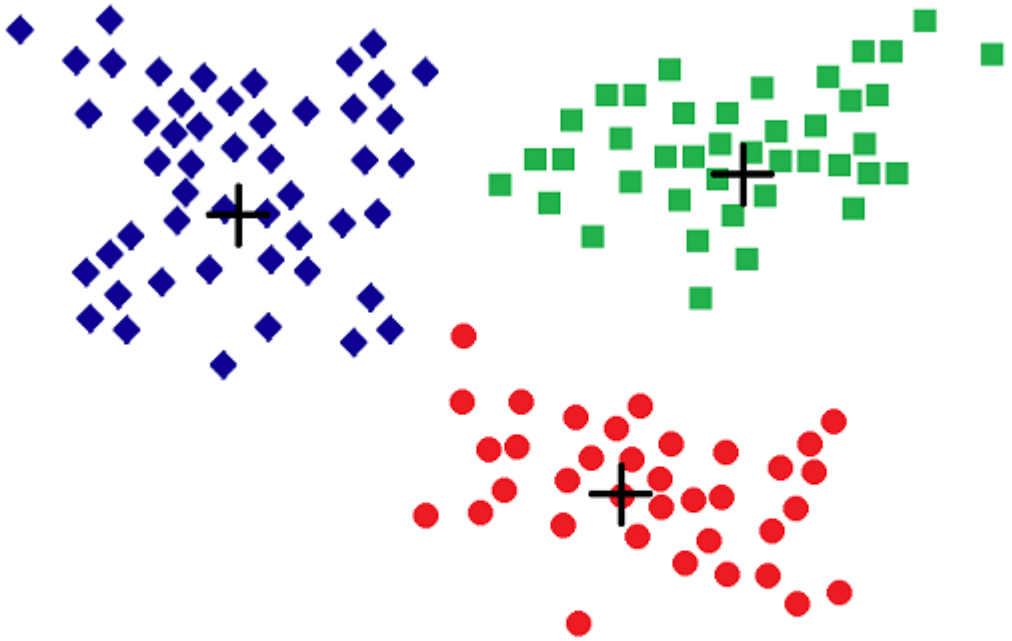
K-Means and K-NN Algorithms

Implementations Available at

http://dmkd.cs.wayne.edu/TUTORIAL/Bigdata/

K-MEANS CLUSTERING ALGORITHM

Basic K-Means Algorithm



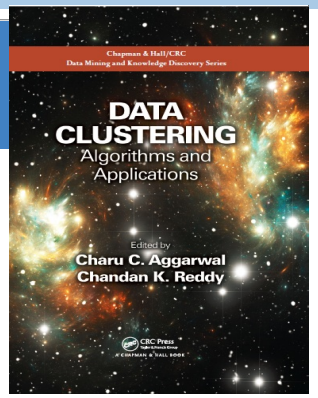
Iteration 5

Basic K-Means Clustering Algorithm

Input: Dataset *D*, Number of clusters *k*
Output: Data points with cluster memberships
1: Initialize random k training data points as centroids
2: Do
3: Compute the distance between each point in D and each point in centroids
4: Sort distances for each data point
5: Associate data points to the nearest centroid
6: Recompute the centroids
7: While No changes in cluster membership

- Starts by initializing the cluster centroids
- Each data point is associated with the nearest centroid in the step 2
- In Step 3, centroids are recalculated
- Step 2 and Step 3 are repeated until the centroids converge or till predefined number of iterations

Data Clustering: Algorithms & Applications



Covers recent advances in Data Clustering

Survey Chapters from prominent researchers

Feature Selection for Clustering

Probabilistic Models for Clustering

Partitional/Hierarchical Clustering

Density Based Clustering

Grid-based Clustering

NMF for Clustering

Spectral Clustering

Clustering High Dimensional Data

Data Stream Clustering

Big Data Clustering

Clustering Categorical Data

Document Clustering

Clustering Multimedia Data

Time Series Data Clustering

Clustering Biological Data

Network Clustering

Uncertain Data Clustering

Visual & Interactive Clustering

Semi-Supervised Clustering

Alternative Clustering

Cluster Ensembles

Clustering Validation

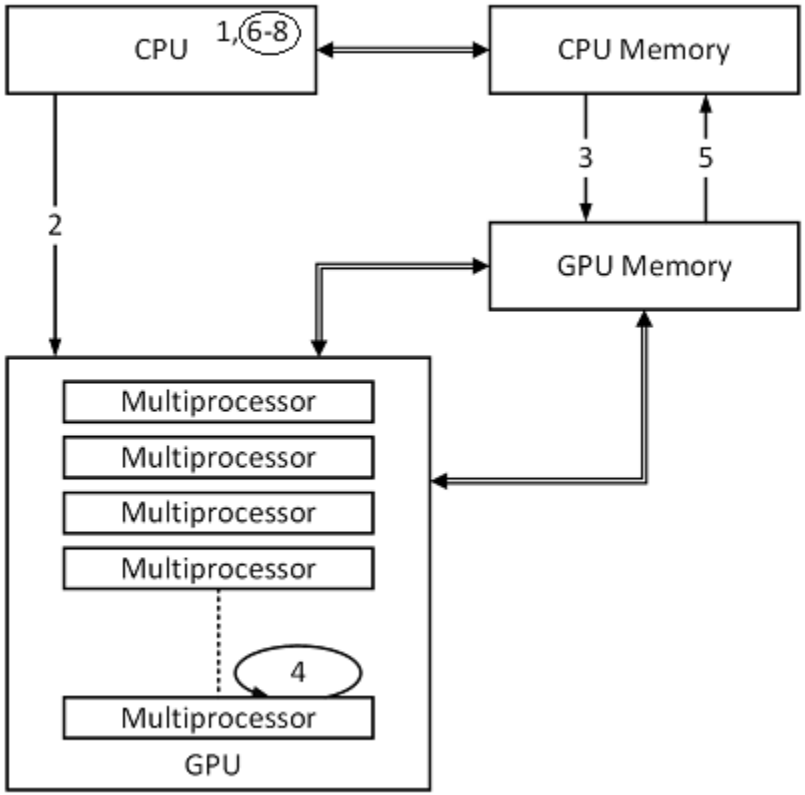
K-Means Clustering on Different Platforms

- Most popular and widely used clustering algorithm
- Contains critical elements that can demonstrate the ability of various platforms
- Characteristics include:
  - Iterative nature of the algorithm wherein the current iteration results are needed before proceeding to the next iteration
  - Compute-intensive task of calculating the centroids from a set of data points
  - Aggregation of the local results to obtain a global solution when the algorithm is parallelized

K-Means GPU Pseudocode

Input: Dataset *D*, Number of clusters *k*
Output: Data points with cluster memberships
1: Initialize first *k* data points as centroids
2: For *iteration* = 1 to *MaxIterations* do
3: Copy D and centroids to GPU shared memory. Split D into threads
4: Kernel process: Compute distance between point in D and each point in centroids
5: Send the distances to CPU
6: CPU process: Sort distances for each data point
7: CPU process: Associate each data point to closest centroid
8: CPU process: Recompute the centroids
9: end For

K-Means on GPU



Step 6: CPU process: Sort distances for each data point

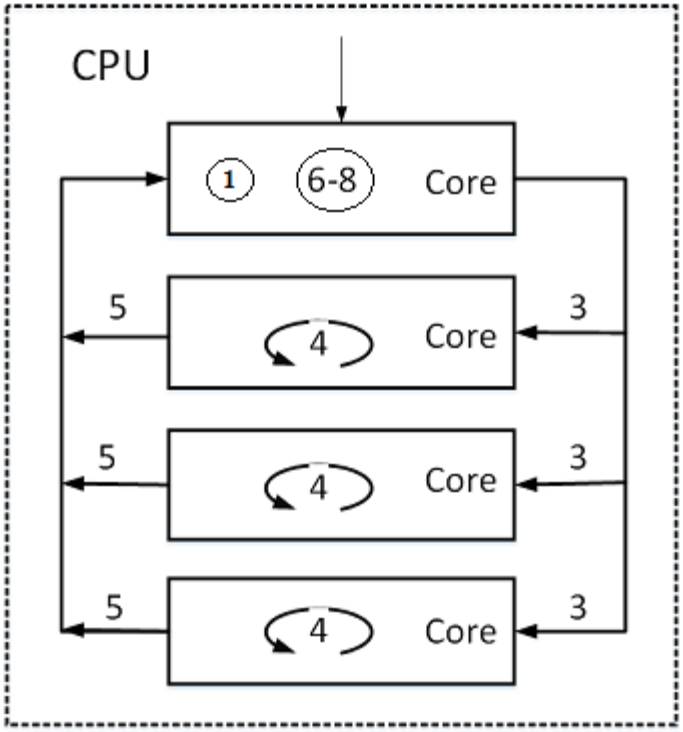
Step 7: CPU process: Associate each data point to closest centroid

Step 8: CPU process: Recompute the centroids

K-Means Multicore CPU Pseudocode

Input: Dataset *D*, Number of clusters *k*
Output: Data points with cluster memberships
1: Initialize first *k* data points as *centroids*
2: For *iteration* = 1 to *MaxIterations* do
3: Split D into multiple cores
4: Compute distance between each point in D and each point in *centroids*
5: Send distances to central core
6: Sort distances for each data point
7: Associate each data point in D with the nearest centroid
8: Recompute the *centroids*
9: end For

K-Means on Multicore CPU



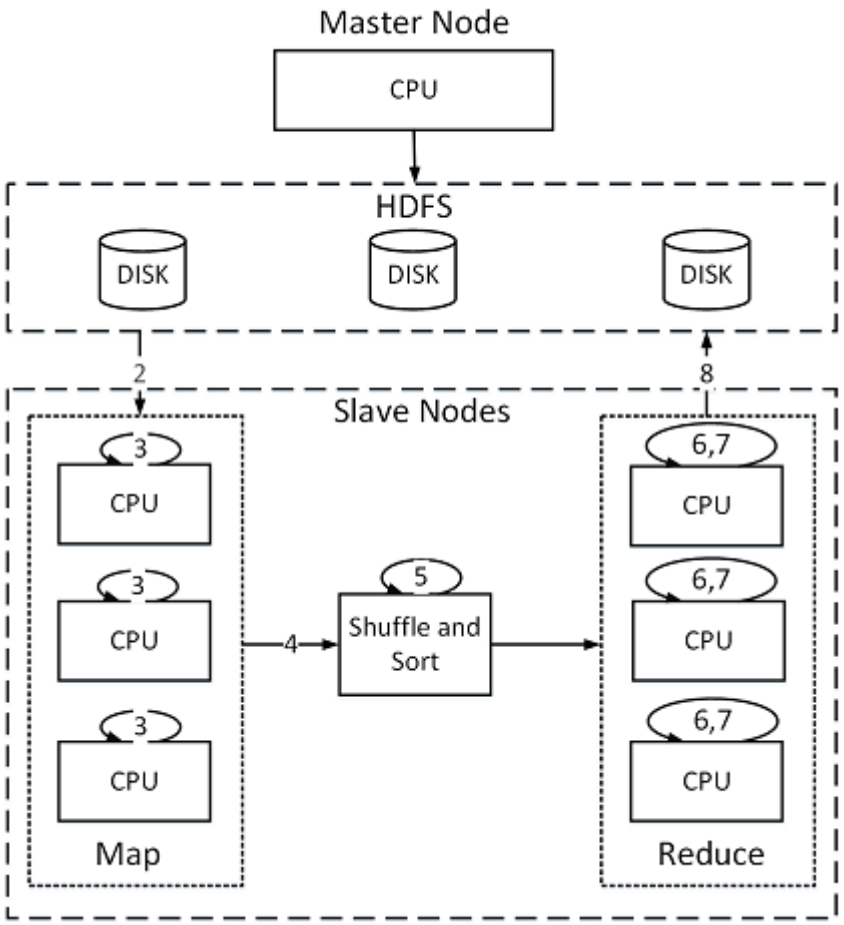
Step 6: Sort distances for each data point
Step 7: Associate each data point in D with the nearest centroid
Step 8: Recompute the *centroids*

K-Means Mapreduce Pseudocode

Input: Dataset *D*, *centroids*
Output: Data points with cluster memberships
1: For *iteration* = 1 to *MaxIterations* do
2: Mapper: Read *D* and *centroids* from HDFS
3: Mapper: Compute the distance between each point in *D* and each point in *centroids*
4: Mapper Output: Key-value pairs with key as centroid id and value as data point id and distance between them
5: Shuffle and Sort: Aggregate for each key (centroid)
6: Reducer: Sort distances and associate data points to the nearest centroid
7: Reducer: Recompute the *centroids*
8: Reducer Output: Write *centroids* to HDFS
9: end For

- Mapper reads the data and centroid from the disk
- Mappers assign data instances to clusters and compute new local centroids and cluster sizes
- Reducers aggregate the local centroids and write the data to the disk for the next iteration
- This shows the disk I/O bottle neck for MapReduce in case of iterative tasks

K-Means on MapReduce



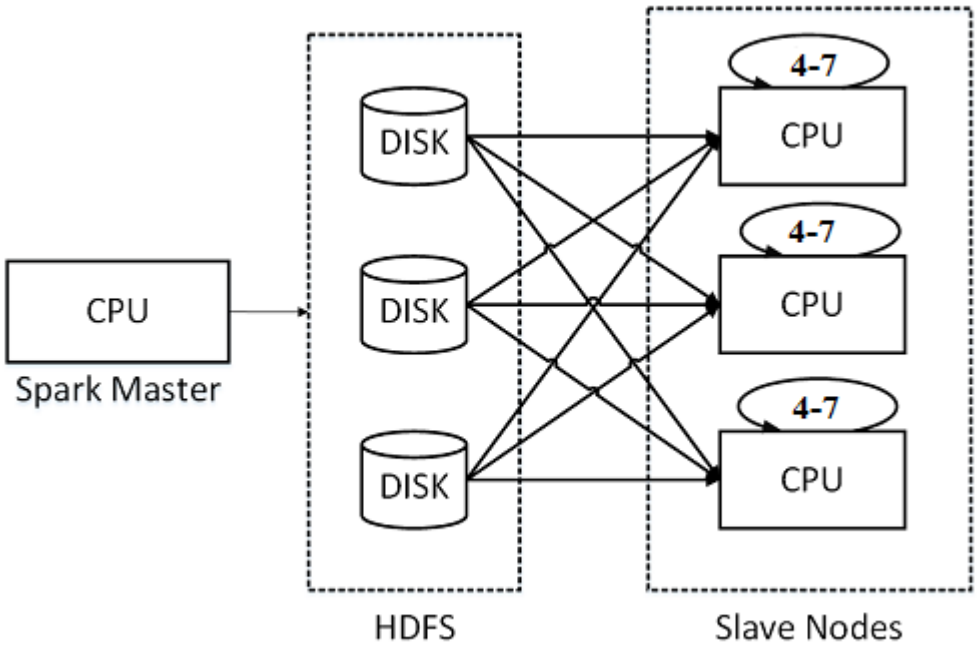
Step 8: Reducer Output: Write *centroids* to HDFS

K-Means Spark Pseudocode

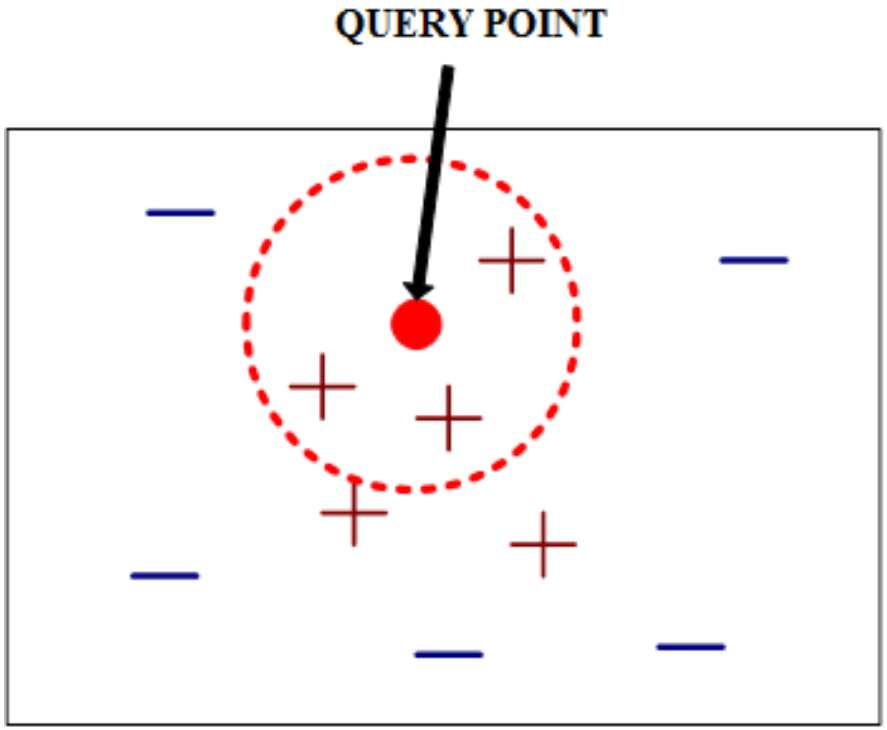
Input: Dataset *D*, Number of clusters *k*
Output: Data points with cluster memberships
1: Read D from HDFS as RDD
2: Initialize first k data points as centroids
3: For *iteration* = 1 to *MaxIterations* do
4: Compute distance between each point in D and each point in centroids
5: For each data point group distances
6: Associate data points to their closest centroid
7: Recompute the *centroids*
8: end For

- K-Means implementation on Spark is similar to K-Means implementation on MapReduce
- Only difference being instead of writing the global centroids to the disk, they are written to the system memory instead
- Data points are also loaded in the system memory for faster access

K-Means on Spark



Step 4: Compute distance between each point in D and each point in centroids
Step 5: For each data point group distances
Step 6: Associate data points to their closest centroid
Step 7: Recompute the centroids



K-NEAREST NEIGHBOR ALGORITHM

Basic K-NN Algorithm

Input: Train Data D, Test Data X, Number of nearest neighbors k

Output: Predicted class labels of X

1: Compute the distance between each $d_i \in D$ and each $x_j \in X$

2: For each test instance sort the distances

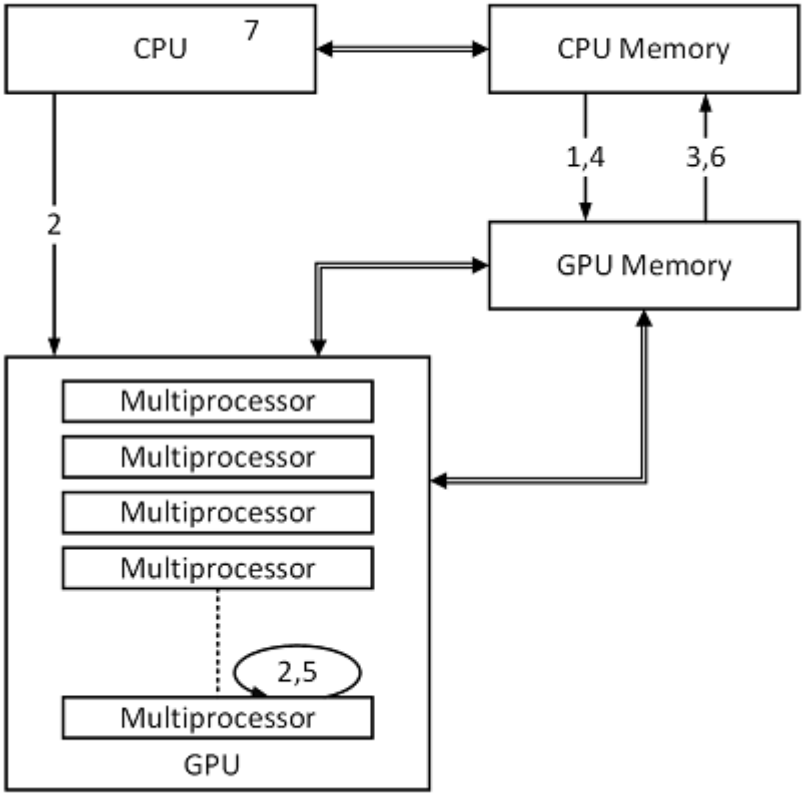
3: Take first k train data points as nearest neighbors

4: Assign the most frequent class label from nearest neighbors as predicted class label

K-NN GPU Pseudocode

Input: Train Data *D*, Test Data *X*, Number of nearest neighbors *k*
Output: Predicted class labels of *X*
1: Copy D and X to the GPU shared memory. Split D into threads
2: Kernel 1: Compute the distance between each $d_i \in D$ and each $x_j \in X$
3: Send the distances to CPU
4: Copy distances to GPU shared memory, split into threads
5: Kernel 2: Sort distances for each test instance
6: Send indices of *k* nearest neighbors to CPU
7: CPU Process: Assign most frequent class label from nearest neighbors as predicted class label

K-NN on GPU

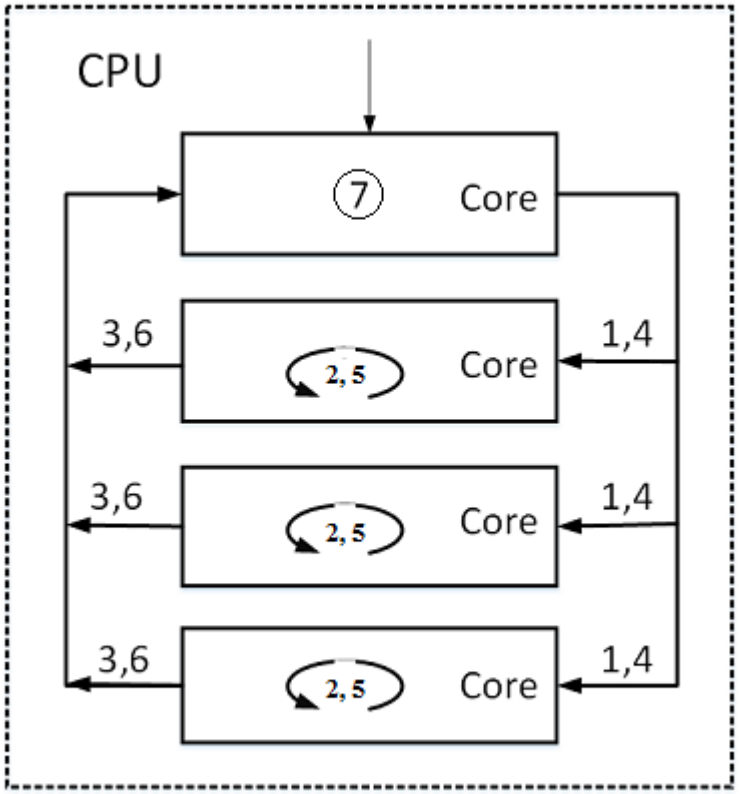


Step 7: CPU Process: Assign most frequent class label from nearest neighbors as predicted class label

K-NN MultiCore CPU Pseudocode

Input: Train Data D, Test Data X, Number of nearest neighbors k
Output: Predicted class labels of X
1: Split D into multiple cores
2: Calculate the distance between each $d_i \in D$ and each $x_j \in X$
3: Send distances to central core
4: Split the distances into multiple cores
5: For each test instance sort the distances
6: Send indices of k nearest neighbors to central core
7: Assign most frequent class label from nearest neighbors as predicted class

K-NN on MultiCore CPU

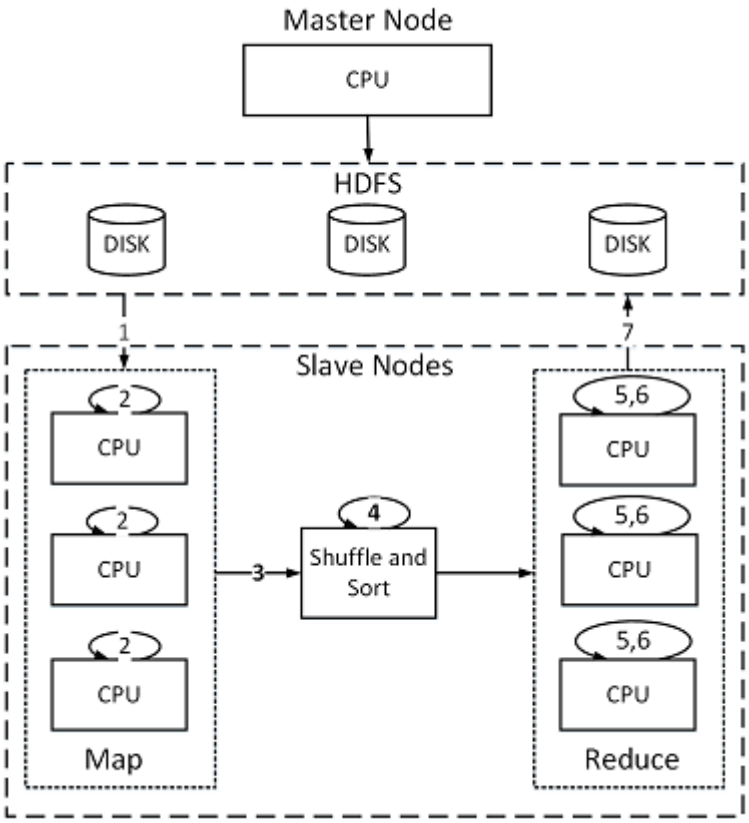


Step 7: Assign most frequent class label from nearest neighbors as predicted class label

K-NN MapReduce Pseudocode

Input: Train Data D, Test Data X, Number of nearest neighbors k
Output: Predicted class labels of X
1: Mapper: Read D and X from HDFS
2: Compute the distance between each $d_i \in D$ and each $x_j \in X$
3: Mapper Output: Key-value pairs with key as test instance Id and value as train instance ID and the distance between them
4: Shuffle and Sort: Aggregate for each key (test instance)
5: Reducer: Sort the distances and take first k train instances as nearest neighbors
6: Reducer: Take majority voting of class labels of nearest neighbors
7: Reducer Output: Class labels of test instances

K-NN on MapReduce

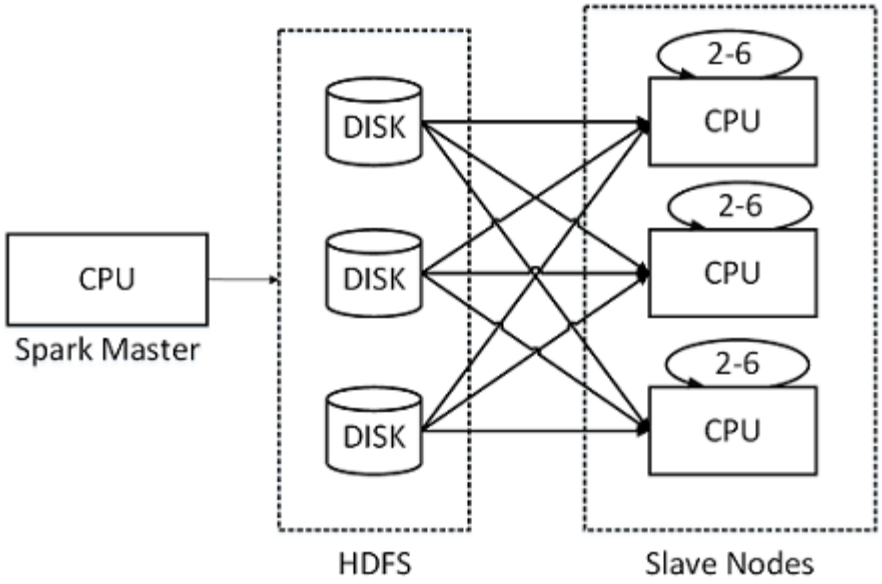


Step 7: Reducer Output: Class labels of test instances

K-NN Spark Pseudocode

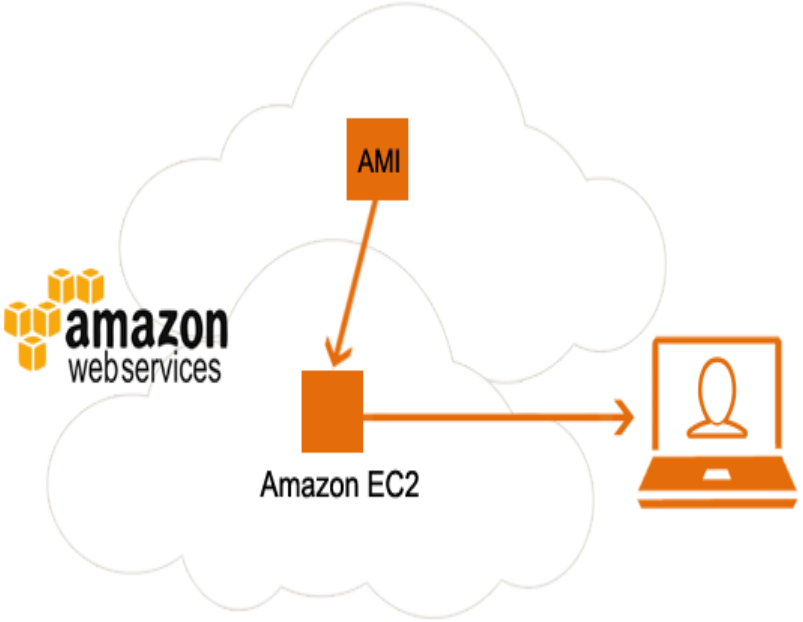
Input: Train Data *D*, Test Data *X*, Number of nearest neighbors *k*
Output: Predicted class labels of *X*
1: Read X as RDD_X and *D* from HDFS
2: Broadcast *D* to all the worker nodes
3: Calculate the distance between each point in RDD_X and *D* as $RDD_{distance}$
4: Find the indices of *k* smallest distances as nearest neighbours
5: Assign most frequent class label from nearest neighbours as predicted class label
6: Write predicted class labels to HDFS

K-NN on Spark



Step 2: Broadcast D to all the worker nodes
Step 3: Calculate the distance between each point in RDD_X and D as $RDD_{distance}$
Step 4: Find the indices of k smallest distances as nearest neighbours
Step 5: Assign most frequent class label from nearest neighbours as predicted class label
Step 6: Write predicted class labels to HDFS

Amazon Web Services



Amazon EC2

- Amazon Elastic Compute Cloud (Amazon EC2) is a **web service** that provides resizable compute capacity in the cloud.
- Designed to make **web-scale computing easier** for developers.
- **Simple web service interface** allows you to obtain and configure capacity with minimal friction.
- Provides you **with complete control** of your computing resources and lets you run on Amazon's proven computing environment.
- **Reduces the time required** to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change.
- Changes the economics of computing by allowing you to **pay only for capacity** that you actually use.
- Provides developers the tools to **build failure resilient** applications and isolate themselves from common failure scenarios.

amazon web services™ EC2

Benefits

- **Elastic Web-Scale Computing**
  - Enables you to increase or decrease capacity within minutes.
  - You can commission thousands of server instances simultaneously.
  - Applications can automatically scale itself up and down depending on its needs.
- **Completely Controlled**
  - You have root access to each instance
  - You can stop your instance while retaining the data.
  - Instances can be rebooted remotely using web service APIs.
  - You also have access to console output of your instances.
- **Flexible Cloud Hosting Services**
  - You have the choice of multiple instance types, operating systems, and software packages.
  - It allows you to select a configuration of memory, CPU, instance storage, and the boot partition size that is optimal for your choice of operating system and application.
- **Reliable**
  - The service runs within Amazon's proven network infrastructure and data centers.
  - The Amazon EC2 Service Level Agreement commitment is 99.95% availability for each Amazon EC2 Region.

Benefits

- **Secure**
  - Amazon EC2 works in conjunction with Amazon VPC to provide security and robust networking functionality
  - Instances are located in a Virtual Private Cloud (VPC) with an IP range that you specify.
  - You decide which instances are exposed to the Internet and which remain private.
  - Security Groups and networks ACLs allow you to control inbound and outbound network access.
  - You can provision your EC2 resources as Dedicated Instances. Dedicated Instances are Amazon EC2 Instances that run on hardware dedicated to a single customer for additional isolation.
- **Inexpensive**
  - Pay only for what is used, without up-front or long-term commitments
  - **On-Demand Instances** let you pay for compute capacity by the hour with no long-term commitments.
  - **Reserved Instances** give you the option to make a low, one-time payment for each instance and in turn receive a significant discount on the hourly charge for that instance.
  - **Spot Instances** allow customers to bid on unused Amazon EC2 capacity and run those instances for as long as their bid exceeds the current Spot Price.
- **Easy to Start**
  - Choosing preconfigured software on Amazon Machine Images (AMIs), you can quickly deploy softwares to EC2 via 1-Click launch or with the EC2 console.

amazon web services™ EC2

Using EC2 Services

- Instances are priced depending upon the configurations

Instance Type	Usage	Use cases	Price range
T2	General Purpose	Development environments, build servers, code repositories, low-traffic web applications, early product experiments, small databases.	$0.013 - $0.520 per hour
M3	General Purpose	Small and mid-size databases, backend servers for SAP, Microsoft SharePoint	$0.070 - $0.560 per hour
C3	Compute Optimized	High performance front-end fleets, web-servers, on-demand batch processing, distributed analytics, high performance science and engineering applications, ad serving, batch processing and distributed analytics.	$0.105 - $1.680 per hour
R3	Memory Optimized	High performance databases, distributed memory caches, in-memory analytics, genome assembly and analysis, larger deployments of SAP, Microsoft SharePoint, and other enterprise applications.	$0.175 - $2.800 per hour
G2	GPU	Game streaming, video encoding, 3D application streaming, GPGPU, and other server-side graphics workloads.	$0.650 per hour
I2	Storage Optimized	NoSQL databases, scale out transactional databases, data warehousing and cluster file systems.	)

EC2 Best Practices

- Make sure you choose the correct instance type, depending upon your use case
- Make sure you choose the correct OS and the appropriate amount of storage for your use case
- For Development purposes, choose the configurations which are provided for free. AWS provides a free tier for 750 hours each month for some configurations
- While using On-demand instances, make sure to stop the instances if not in use and restart them later as required
- Terminate the instances which you won't be needing anymore to avoid being charged.

amazon web services™ EC2

Big Data Platform Instance Configurations for AWS

AWS GPU Instance

- Type : g2.2xlarge (GPU)
- Instances used: 1
- Processor: Intel Xeon-E5-2670 (Sandy Bridge)
- CPU cores: 8
- Memory : 15 GiB

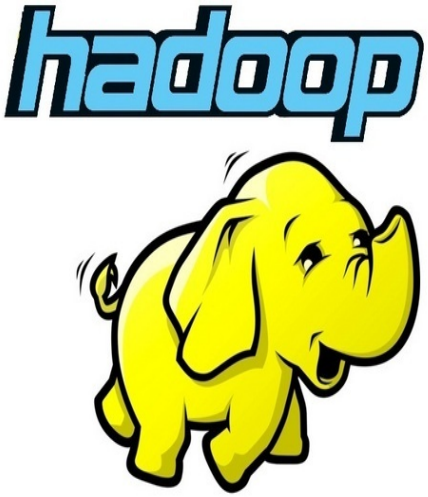
- GPU: NVIDIA (Kepler GK104) with 1,536 CUDA cores and 4 Gb of video memory
- Instance storage: 60GB SSD
- Cost per hour: $0.65

AWS Multicore Instance

- Type : c3.4xlarge (Compute optimized)
- Instances used: 1
- Processor: Intel Xeon E5-2666 v3 (”Haswell”)
- CPU cores: 16
- Memory: 30 GiB
- Instance storage: 320 GB SSD
- Cost per hour: $0.84

AWS Hadoop Instances

- Type : m3.large
- Instances used: 5
- Processor: Intel Xeon E5-2670 v2 (Ivy Bridge)
- CPU cores: 2
- Memory: 7.5 GiB
- Instance storage: 32 GB SSD
- Cost per hour: $0.65

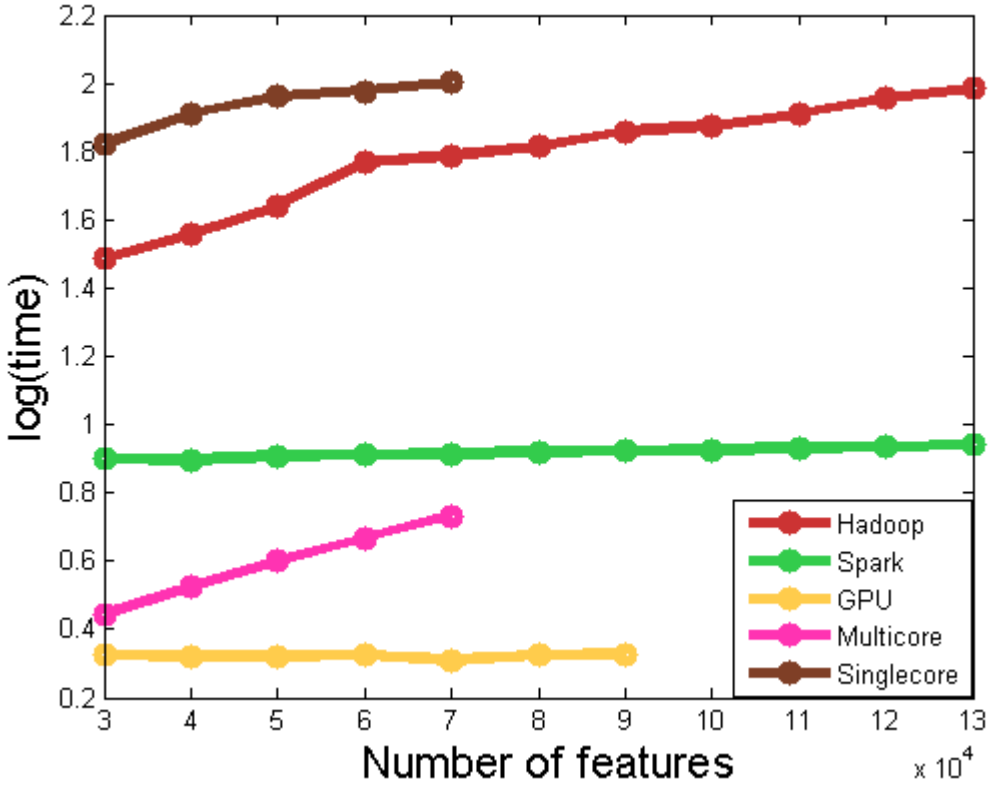


AWS Spark Instances

- Type : r3.large
- Instances used: 4
- Processor: Intel Xeon E5-2670 v2 (Ivy Bridge)
- CPU cores: 2
- Memory : 15 GiB
- Instance storage: 32 GB SSD
- Cost per hour: $0.7

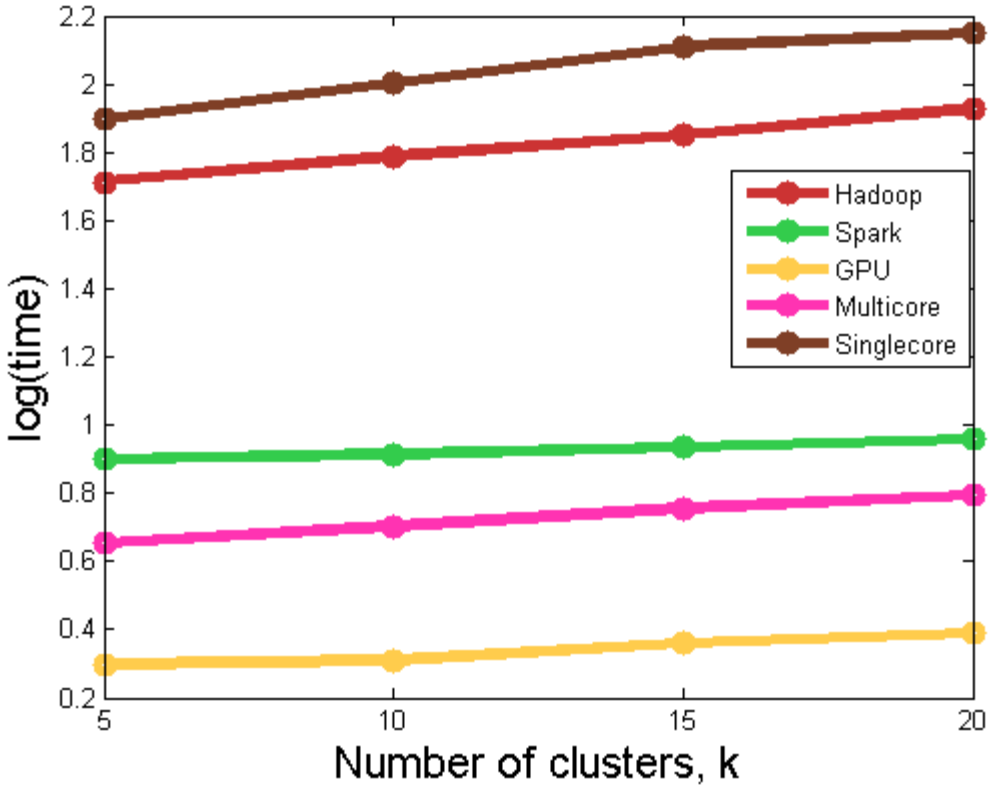
Comparison Result: K-Means

- Runtime Comparison of K-means algorithm on four big data platforms and naïve single core implementation for *k*=10 with data dimension varying from 30K to 130K.



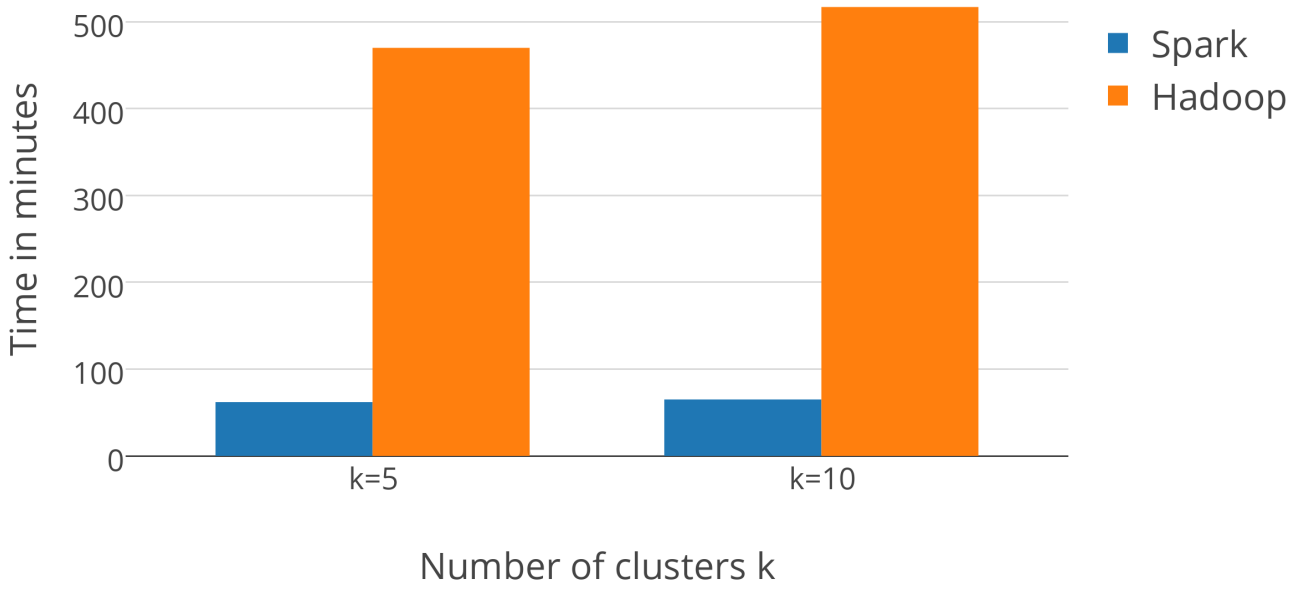
Comparison Result: K-Means

- Runtime comparison of K-means algorithm on four big data platforms and single core machine varying number of clusters *k* with data dimension 70K.



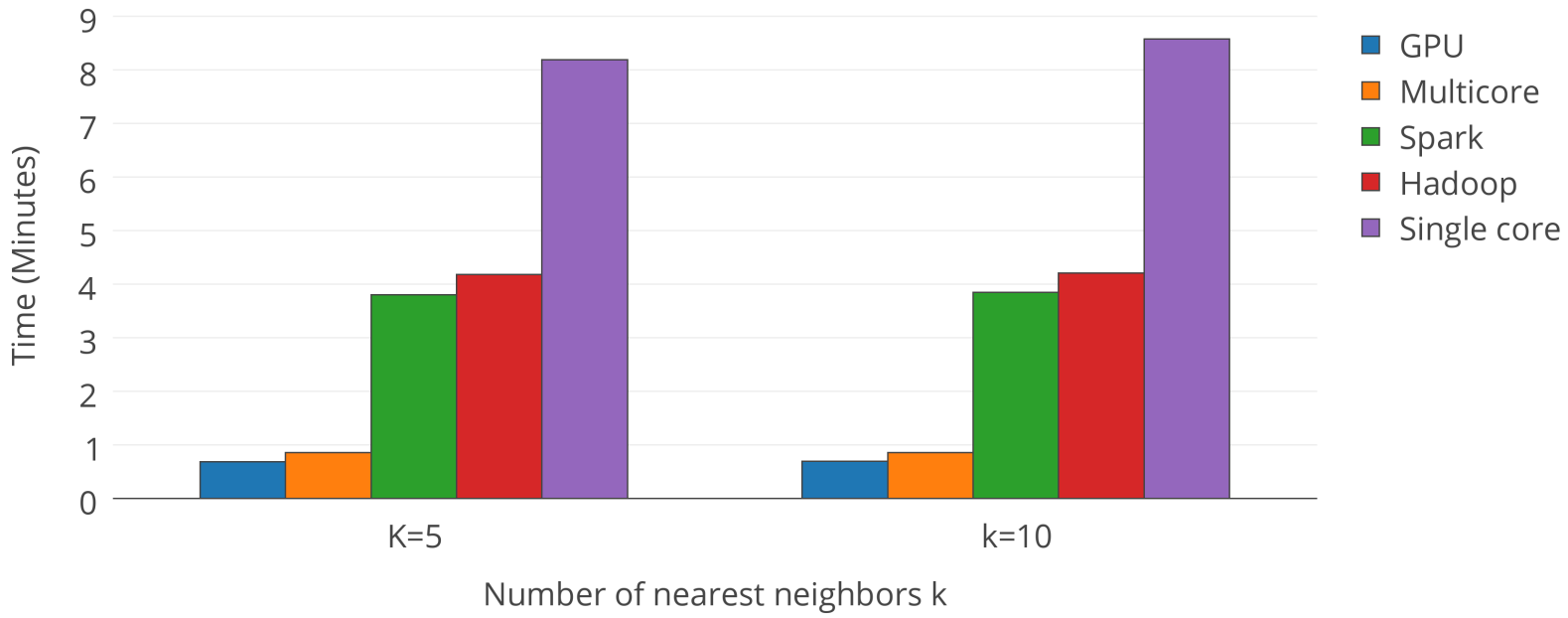
Comparison Result: Scalability

- Scalability test of the platforms with large data (1.3 million features, 20K rows) having size 50.4 Gigabytes. Only Hadoop and Spark were able to process such big data.



Comparison Result: K-NN

- Runtime comparison of K-NN algorithm on big data platforms.

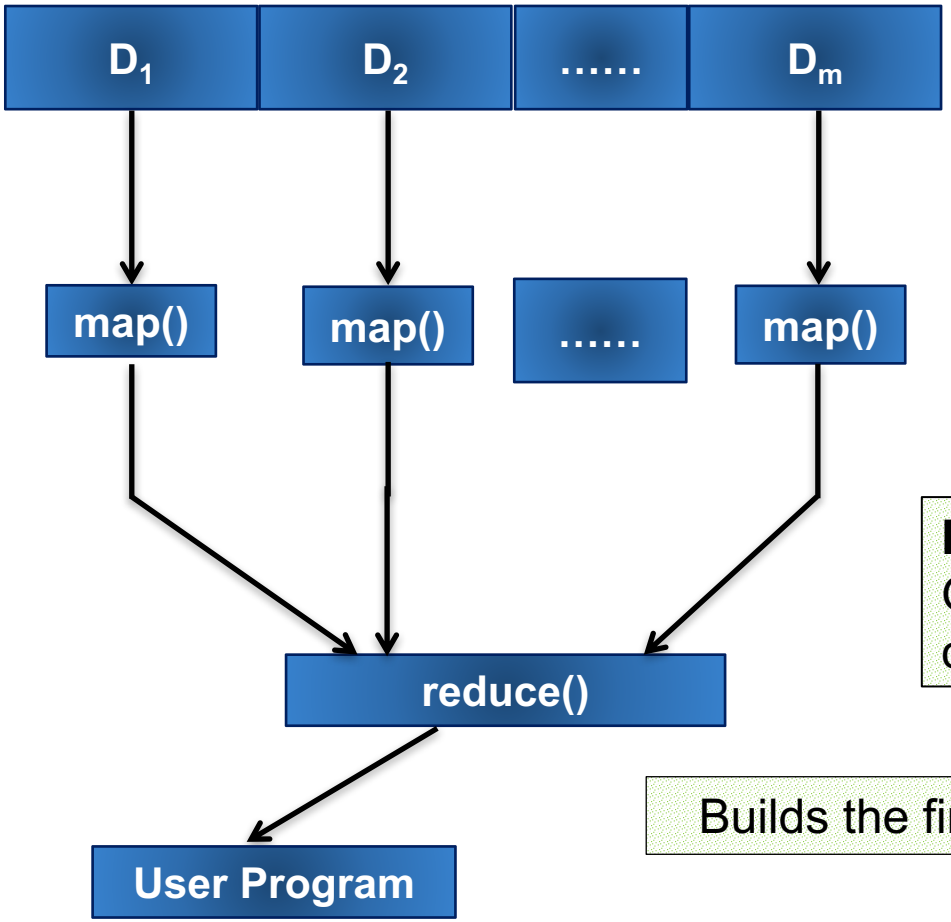


- Vertical scaling platforms perform better than horizontal scaling platforms. Single core implementation takes the highest amount of time as expected.

Comparison Summary

- We observe vertical scaling platforms such as GPU and MultiCore CPU are faster than horizontal scaling platforms such as Hadoop and Spark.
- We also observed Horizontal scaling methods are more scalable. For example vertical scaling platform GPU cannot scale with a data bigger than 90K features and MultiCore CPU cannot handle more than 70K features. Hadoop and Spark was able to process a data with 1.3 million features.
- We find that GPU outperforms MultiCore CPU by spawning very high number of threads.
- We observe that, for both iterative and non-iterative scenarios, Spark yields better timing than Hadoop. We see that for iterative scenarios (for e.g. K-means), the slowness of Hadoop is due to unavoidable file I/O operations in each iteration.

Map-Reduce Workflow for Classification



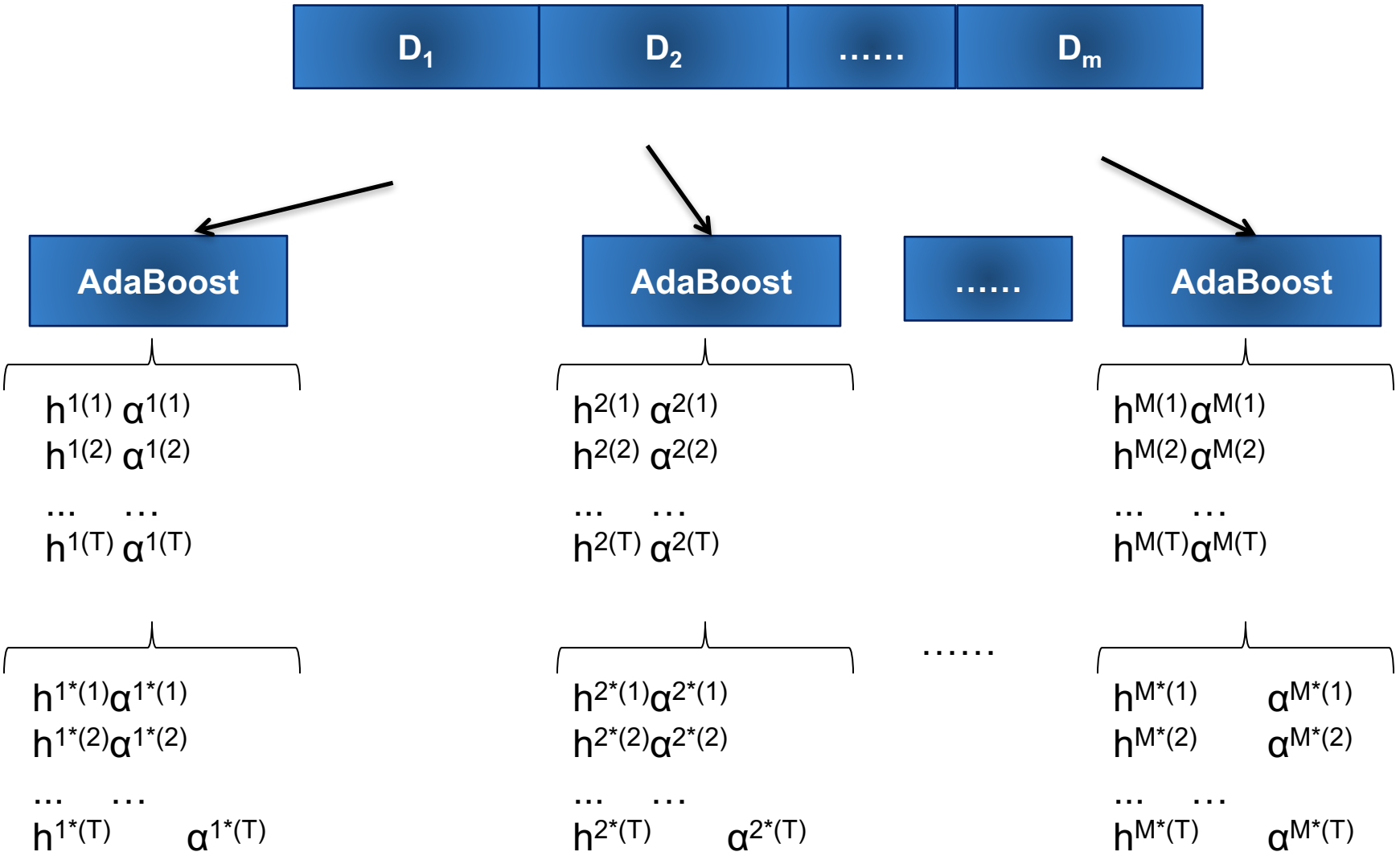
Map task –
Runs AdaBoost on its data and return the sorted list of weak classifiers

Reduce task –
Generates the merged classifier and find its weight

Builds the final classifier from Reducer output

Indranil Palit and Chandan K. Reddy, "**Scalable and Parallel Boosting with MapReduce**", *IEEE Transactions on Knowledge and Data Engineering (TKDE),* Vol.24, No.10, pp.1904-1916, October 2012.

AdaBoost.PL Algorithm



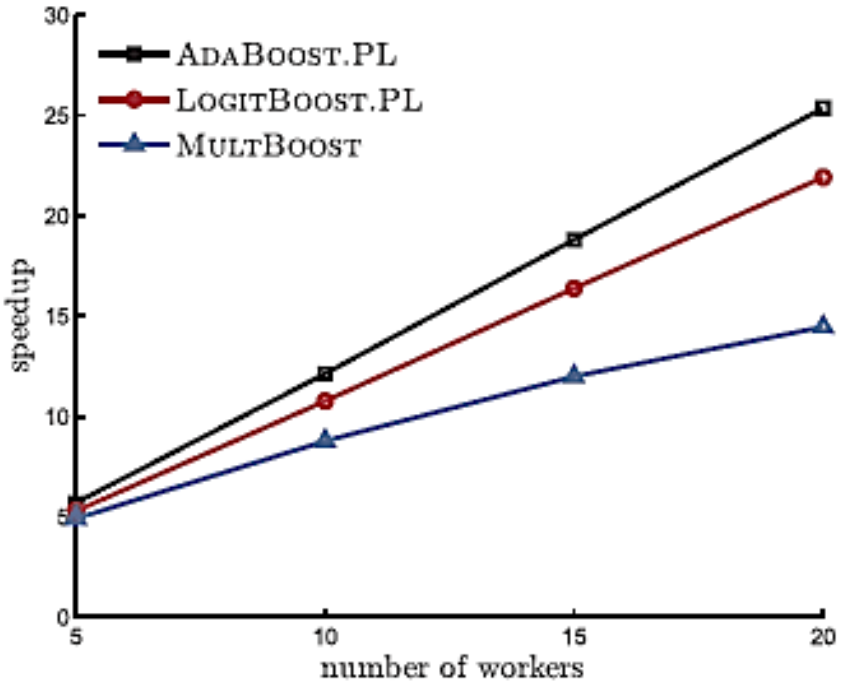
Primary Advantages of Parallelization

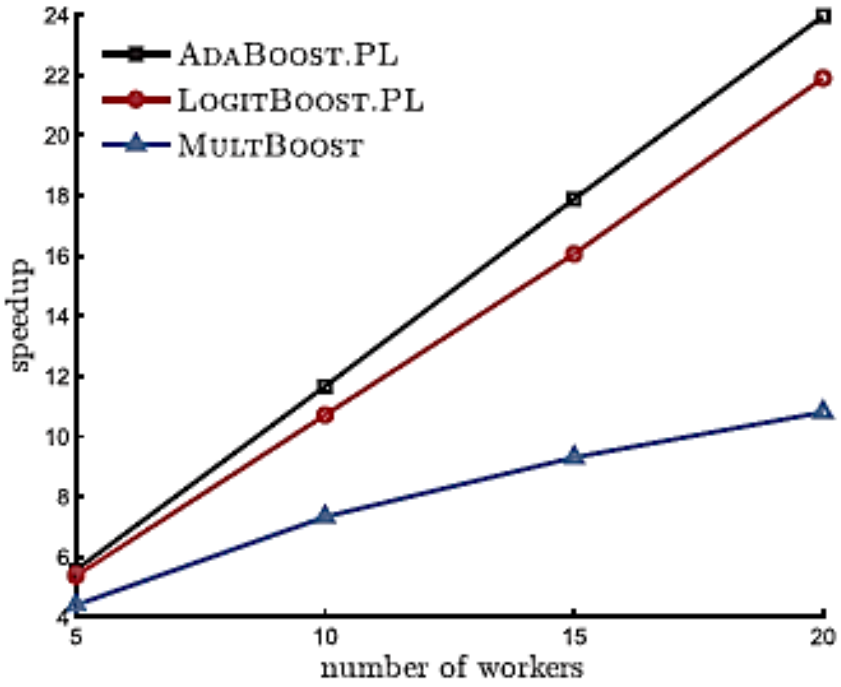
- Scalability: Less resource intensive than serial version, since we are distributing the workload across different machines rather than learning on the entire dataset on a single machine.

- Speed: Runs significantly faster than serial version, since we are simultaneously learning from N data subsets rather than learning a single large data set.

- Privacy Preserving: Preserves privacy by learning each data subset in a local node without sharing any of its data outside the node.

Speedup Performance

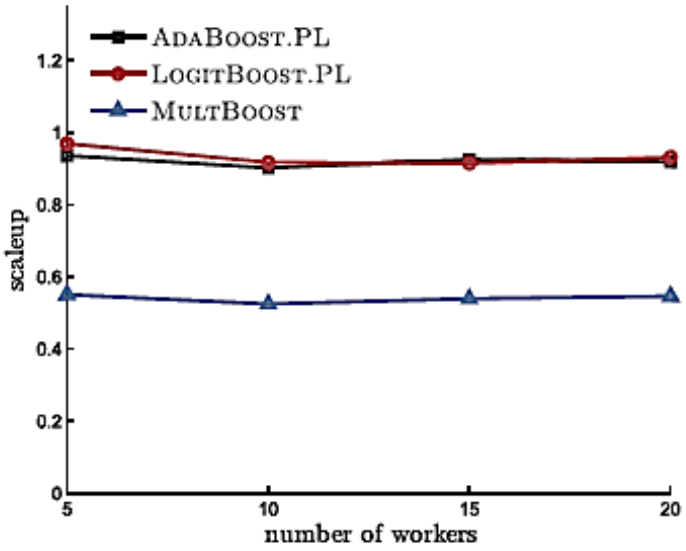


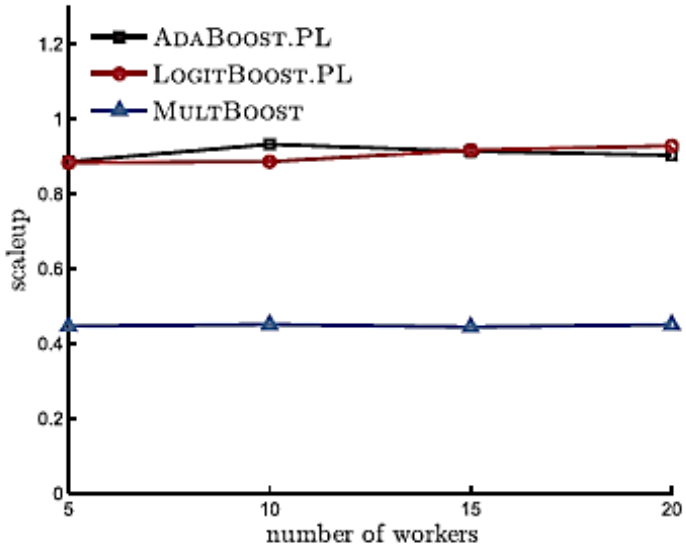


Speedup is defined as the ratio of the execution time on a single processor to the execution time for an identical data set on p processors.

Scaleup Performance

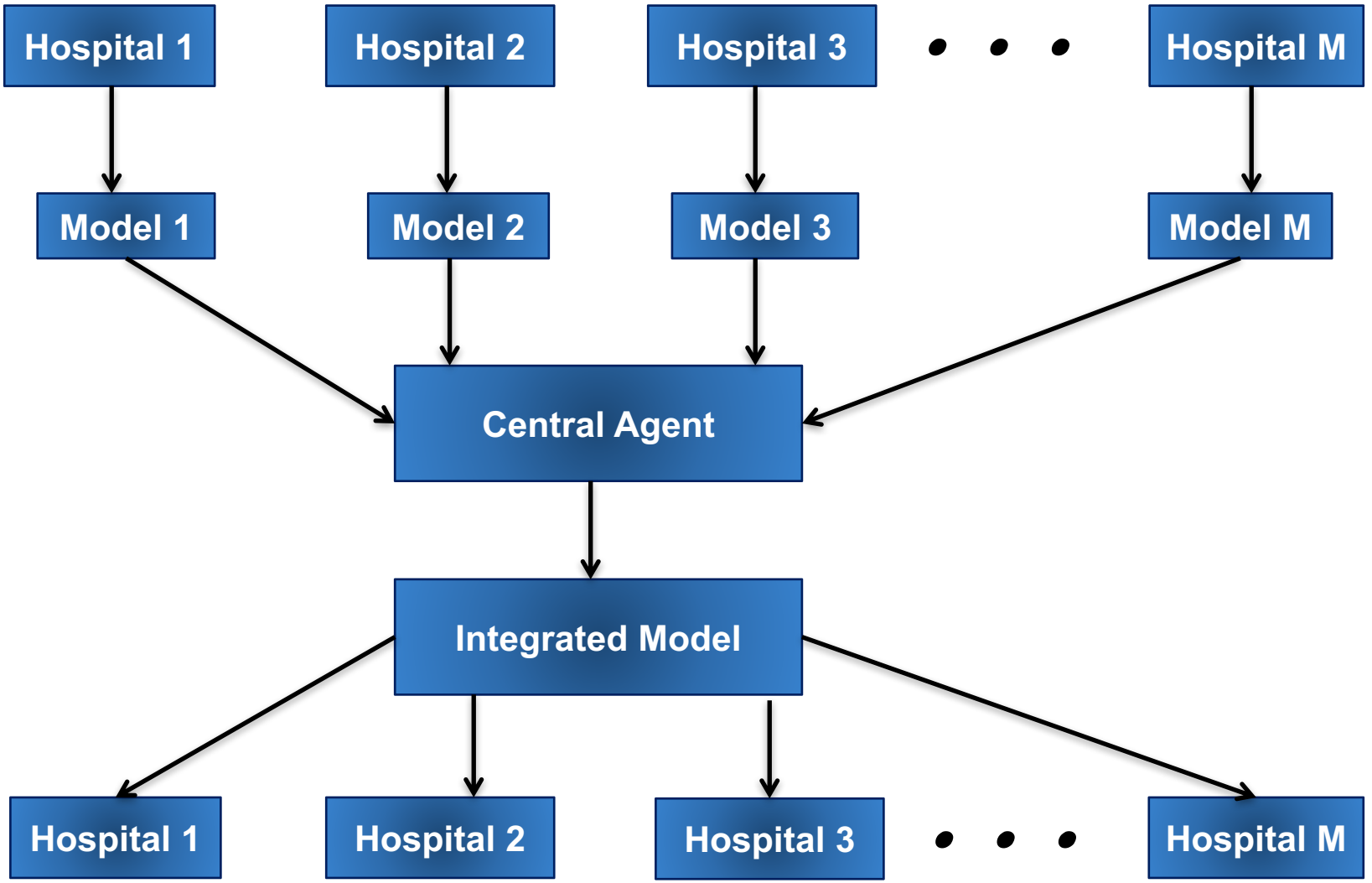
- Scaleup is the ratio of the time taken on a single processor by the problem to the time taken on p processors when the problem size is scaled by p.





- Study scaleup behavior by keeping the problem size per processor fixed while increasing the number of available processors.
- For a fixed data set, speedup captures the decrease in runtime when we increase the number of available cores. Scaleup is designed to capture the scalability performance of the parallel algorithm to handle large data sets when more cores are available.

Distributed Privacy Preserving Model



Yan Li, Changxin Bai, and Chandan K. Reddy, "**A Distributed Ensemble Approach for Mining Healthcare Data under Privacy Constraints**", *Information Sciences*, Vol.330, pp.245-259, February 2016.

Linear Regression using Least Squares

Develop a general and exact technique for parallel programming of a large class of ML algorithms for multicore processors

Model: $y = \theta^T x$

Goal: $\theta^* = \min_\theta \sum_{i=1}^{m} \left(\theta^T x_i - y_i\right)^2$

Solution: Given m examples: (x_1, y_1), (x_2, y_2), …, (x_m, y_m), Matrix X will have with x_1, …, x_m as rows, and row vector $Y=(y_1, y_2, \ldots y_m)$. Then the solution is

$$\theta^* = \left(X^T X\right)^{-1} X^T \vec{y}$$

Parallel computation:

- $A = X^T X$ $\quad A = \sum_{i=1}^{m} \left(x_i x_i^T\right)$
- $b = X^T \vec{y}$ $\quad b = \sum_{i=1}^{m} \left(x_i y_i\right)$

← Cut to m/num_processor pieces

Complexity with MapReduce

	single	multi
LWLR	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2 \log(P))$
LR	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2 \log(P))$
NB	$O(mn + nc)$	$O(\frac{mn}{P} + nc \log(P))$
NN	$O(mn + nc)$	$O(\frac{mn}{P} + nc \log(P))$
GDA	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2 \log(P))$
PCA	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2 \log(P))$
ICA	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2 \log(P))$
k-means	$O(mnc)$	$O(\frac{mnc}{P} + mn \log(P))$
EM	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2 \log(P))$
SVM	$O(m^2 n)$	$O(\frac{m^2 n}{P} + n \log(P))$

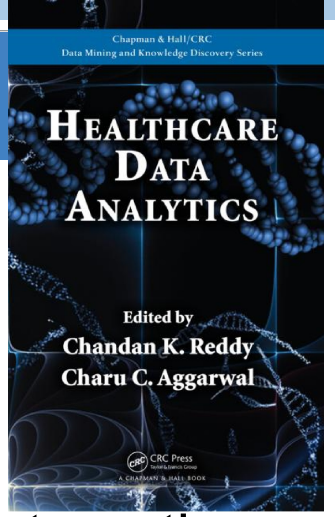
Chu, Cheng, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. "**Map-reduce for machine learning on multicore**", *Advances in neural information processing systems* 19: 281, 2007.

Conclusion

- Big Data is not about handling a particular challenge, rather it is a field in itself.
- Big data can provide potentially revolutionary solutions to the problems where there are no answers yet.
- It can directly impact various disciplines especially in the way the data is currently being handled in those disciplines.

- Different platforms have different strengths and the choice of platforms can play a critical role in the eventual success of the application and/or algorithm used.
- Algorithms for Big Data Analytics are still at their infancy.

Healthcare Data Analytics



Covers recent advances in Healthcare analytics

Survey Chapters from prominent researchers

- Electronic Health Records
- Biomedical Image Analysis
- Sensor Data
- Biomedical Signal Analysis
- Genomic Data
- Clinical Text Mining
- Biomedical Literature
- Social Media Analytics
- Clinical Prediction Models
- Temporal Pattern Mining
- Visual Analytics
- Clinico-Genomic Data Integration
- Information retrieval
- Privacy-Preserving Data Sharing
- Pervasive Healthcare
- Fraud Detection
- Pharmaceutical Data Analysis
- Clinical Decision Support Systems
- Computer Aided Imaging Systems
- Mobile Imaging

KDD 2013 Tutorial on Big Data Analytics on Healthcare. Slides available at http://dmkd.cs.wayne.edu/TUTORIAL/Healthcare/. Sample chapters on Clinical Prediction, Decision Support, and EHR are available at http://www.cs.wayne.edu/~reddy/

Acknowledgements

Funding Agencies

- National Science Foundation
- National Institutes of Health
- Department of Transportation
- Blue Cross Blue Shield of Michigan

Graduate Students

- Dilpreet Singh
- Rajiur Rahman
- Vineeth Rakesh

Thank You

Questions and Comments

Feel free to email questions or suggestions to

reddy@cs.wayne.edu

http://www.cs.wayne.edu/~reddy/