Data Mining - Various Topics

Joe Burdis
Fall 2024
CUNY Graduate Center

Outline

1. Recap of Week 1
2. Reviews (mostly at home / own your own)
   a. Probability and Proof Techniques: https://web.stanford.edu/class/cs246/handouts/CS246_Proof_Probability.pdf
   b. Linear Algebra: https://www.3blue1brown.com/topics/linear-algebra and https://web.stanford.edu/class/cs246/handouts/CS246_LinAlg_review.pdf
3. Distance Metrics and Similarity Measurements for Nearest Neighbors and Clustering
4. Machine Learning Experimentation and Importance of Data Partitioning
5. Python Notebook: Week02_EDA
6. Distributed File System, MapReduce, and Spark (For those not in Big Data Analytics Course)

Recap of Week 1: Python Data Science Libs

1. Data types: boolean, int, float, list, set, tuple, dict, str
2. Using “class” to build data type
3. Defining functions
4. NumPy
5. Pandas
6. Matplotlib
7. Seaborn
8. Plotly
9. Scikit-learn
10. SciPy
11. Keras/TensorFlow, PyTorch
12. Many more …

For data based apps, see streamlit and plotly-dash

Aspects of Data Mining

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	Learning Embeddings	Recommender systems
Clustering	Graph Neural Networks	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Experimentation	Duplicate document detection

Distance Metrics and Similarity Measures

Distance	Equation
Euclidean	$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
Manhattan	$d(x,y) = \sum_{i=1}^{n} \left\| x_i - y_i \right\|$
Minkowski	$d(x,y) = \left(\sum_{i=1}^{n} \left\| x_i - y_i \right\|^p \right)^{\frac{1}{p}}$
Jaccard	$J(A,B) = \frac{\|A \cap B\|}{\|A \cup B\|}$
Cosine	$1 - \frac{x \cdot y}{\|\|x\|\| \cdot \|\|y\|\|}$

Distance / Similarity using Deep Learning

Using Deep Learning for Image or Text Similarity

- Often find some embedding with a deep learning featurizer / encoding space
- Measure the distance in the encoded space (either with or without using dimensionality reduction techniques)

Images Example:

https://keras.io/examples/vision/semantic_image_clustering/

To be discussed later in the course

Distributed File System, MapReduce, and Spark

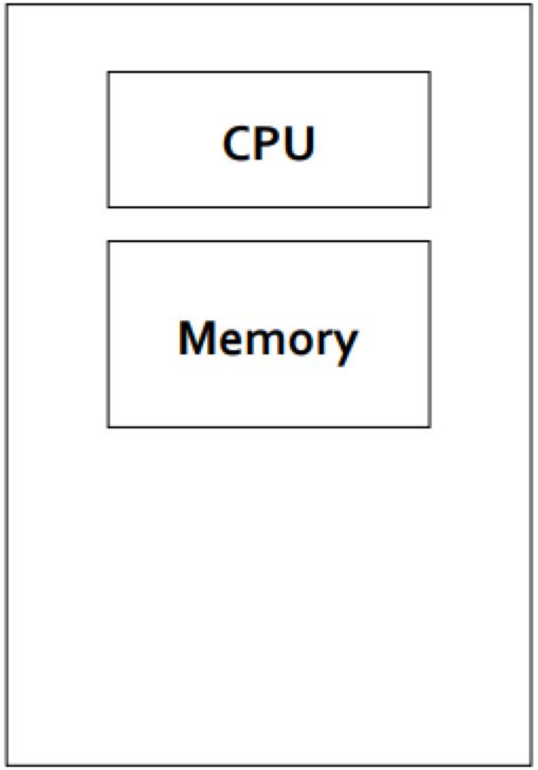
Challenges of Big Data

- **Volume**: MB, GB, Terabytes, Petabytes, …
- **Variety**: time series, images, videos, audios, 3D images, multiview data, …
- **Velocity**: batch-processing vs. real-time processing
- **Veracity**: unstructured data, missing data, data inconsistency, outliers, noise, ...

MapReduce is a computational model aiming to process massive datasets with a parallel algorithm on a computer cluster.

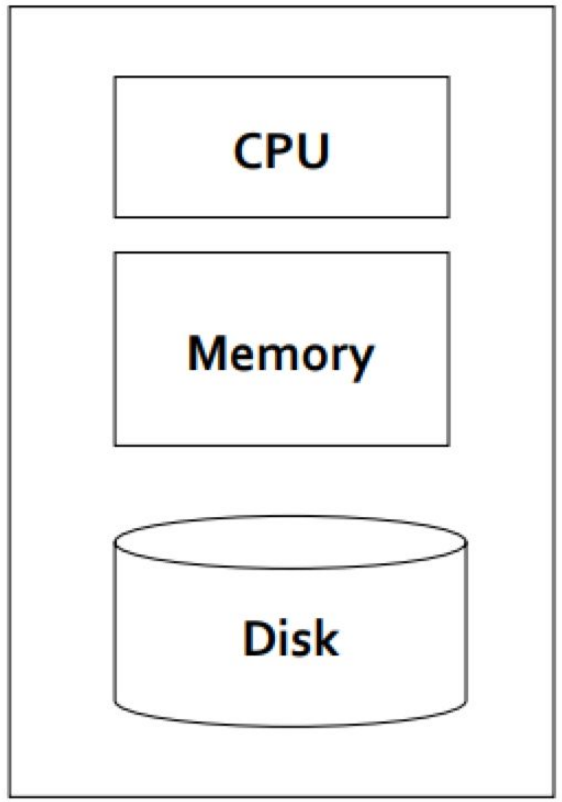
Reference: Mining Massive Datasets, Chapter 2, http://www.mmds.org

Single Node Architecture



Machine Learning, Statistics

Single Node Architecture



Machine Learning, Statistics

"Classical" Data Mining

Challenge of Single-Node Computing

Consider the following scenario:

- Google needs to crawl 130 trillion (130 * 10^{12}) webpages.
- The average size of a webpage is 3 MB.
- Total data size: 390,000,000 TB
- Bottleneck: the data bandwidth between disk and CPU: consider SSD with 5 GB/s R/W speeds
- Time to read all data once: about 78 * 109 seconds (about 2,500 years)

Solution:

- Use multiple disks and CPUs
- Process the data in parallel

Cluster Architecture

- racks consisting of 16 - 64 commodity Linux nodes

- nodes are connected by a gigabit switch (1 Gbps between any pair of nodes in a rack)

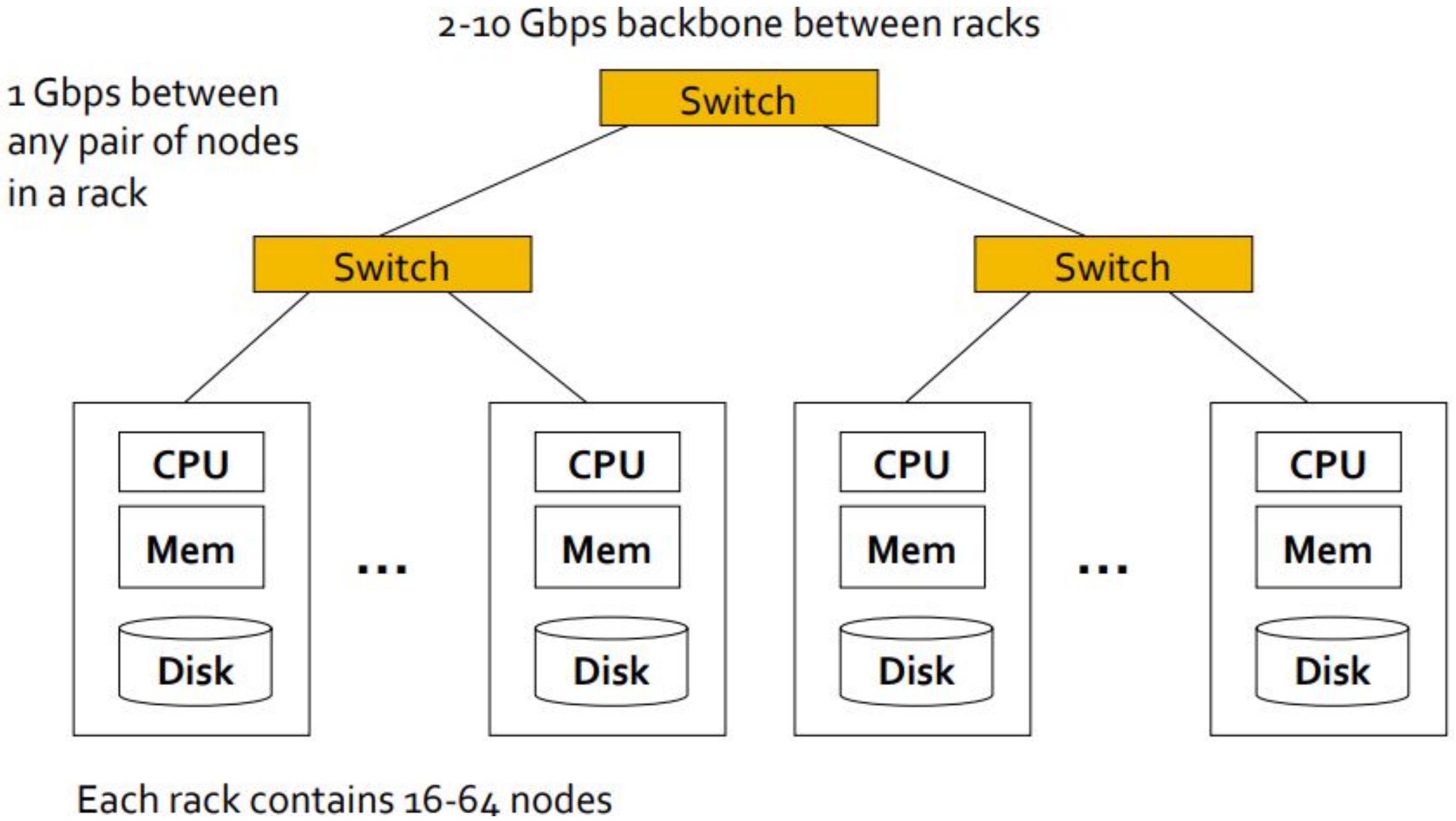
- multiple racks connected by backbone switches (2 - 10 Gbps, sometimes up to 100 Gbps)

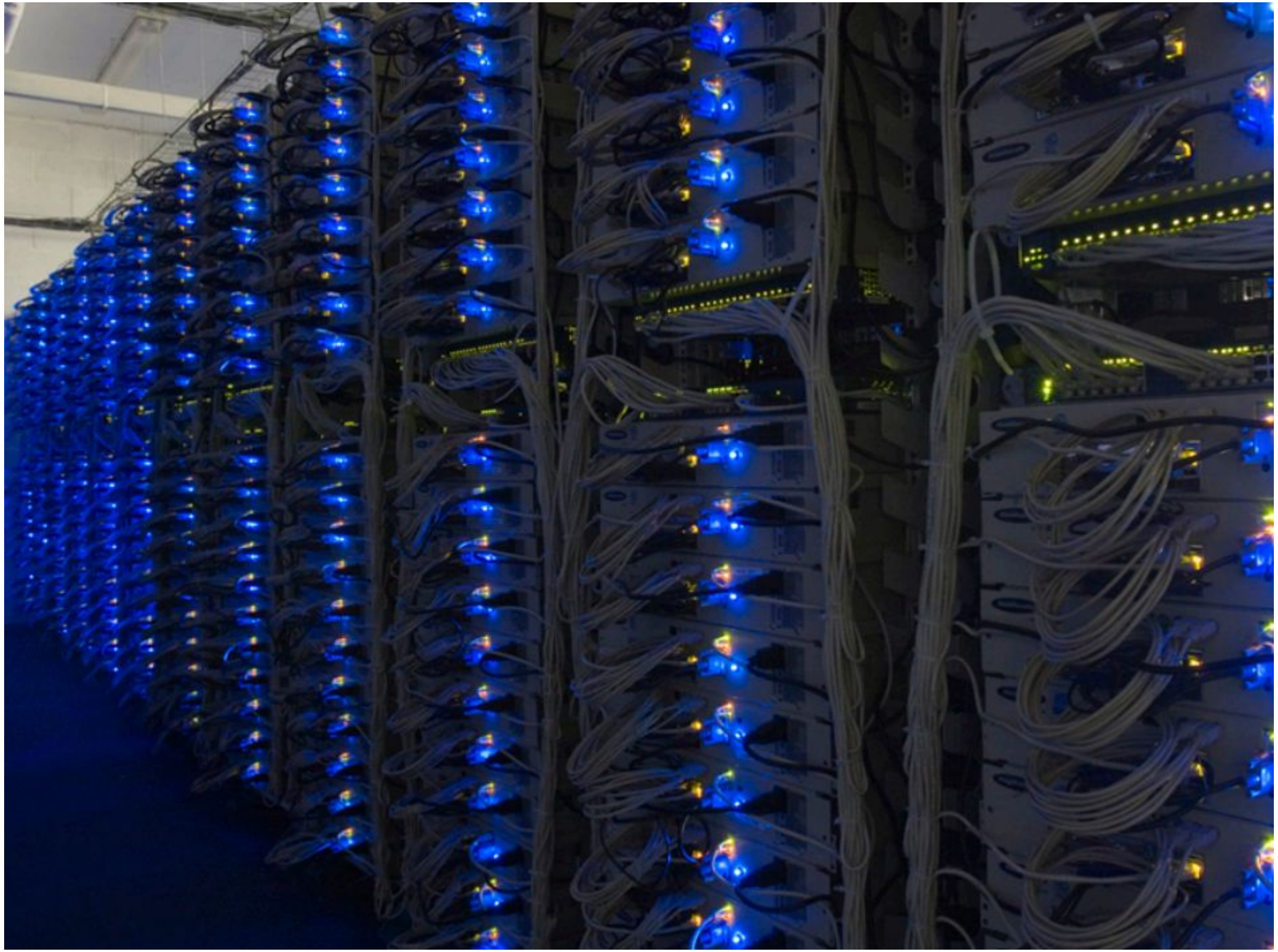
In 2016, it is estimated that Google has 2.5 million nodes (Wikipedia).

Cluster Computing Challenges

- A single server can stay up for 3 year (1000 days)
    - Node failures: 2.5 million in cluster => 2500 failures per day.
    - How to store data persistently and keep it available if nodes can fail?
    - How to deal with node failures during a long-running computation?
- Network bottleneck
    - Moving 10TB via 1Gbps bandwidth takes approximately 1 day.
- Computations need to be parallelized to fully utilize the cluster infrastructure.

Cluster Architecture


2-10 Gbps backbone between racks
1 Gbps between any pair of nodes in a rack
Switch
Switch
Switch
CPU
Mem
Disk
...
CPU
Mem
Disk
CPU
Mem
Disk
...
CPU
Mem
Disk
Each rack contains 16-64 nodes



Quick Comments on Map-Reduce

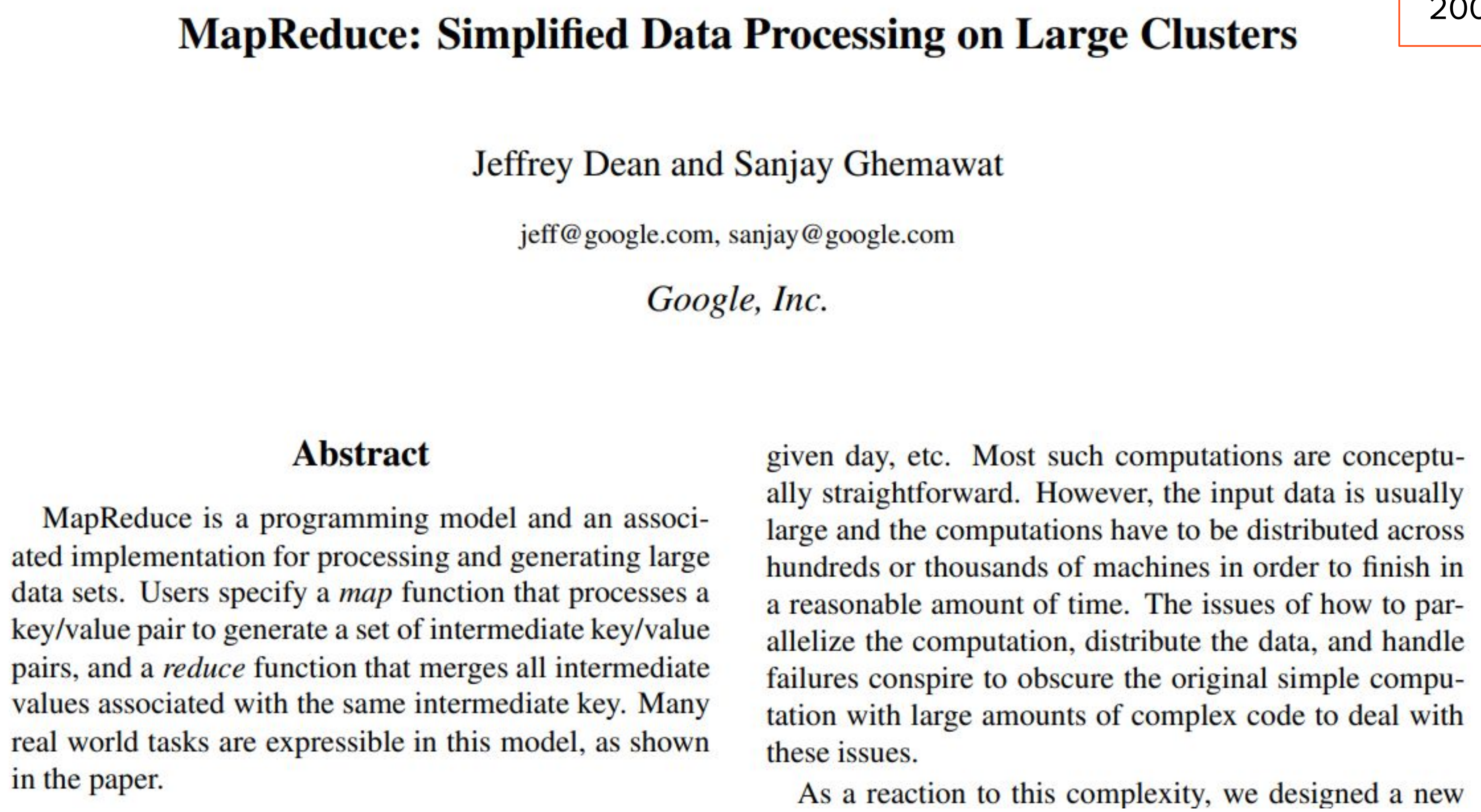
Map-Reduce addresses the challenges of cluster computing

- Store data redundantly on multiple nodes
- Move computation close to data
- Simple programming model

Typical usage pattern

- Huge files (100GB+)
- Data is rarely updated in place
- Reads and appends are common

2004



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

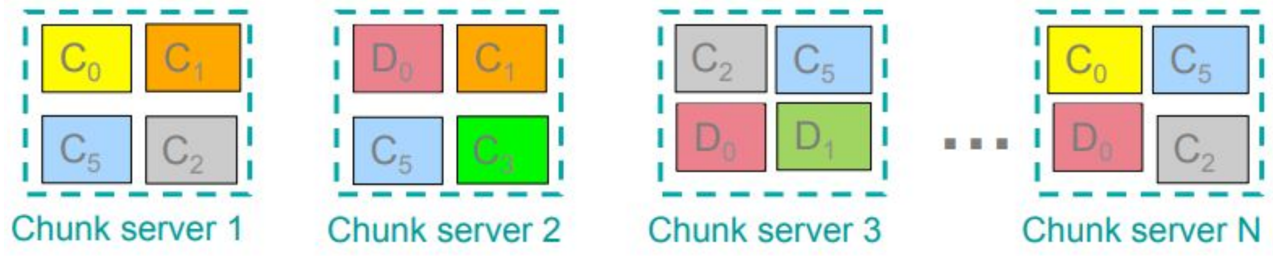
MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new

Redundant Storage Infrastructure

- Distributed File System
  - Google GFS
  - Hadoop HDFS
- Data kept in "chunks" (16 - 64 MB) spread across machines
- Each chunk replicated on different machines (2x or 3x)



Bring computation directly to the data!

Chunk servers also serve as compute servers

Distributed File System

Master node

- In Hadoop it is called Name Node
- Stores metadata about where files are stored
- Might be replicated
- Takes care of coordination: idle, in-progress, completed, failed

Client library for file access

- Talks to master node to find chunk servers
- Connects directly to chunks to access data

Computational Model

Task:

- We have a huge text document
- Count the number of times each distinct word appears in the file
- Sample applications
    - Analyze web server logs to find popular URLs
    - Term statistics for search

Case 1:

- File too large for memory, but all <word, count> pairs fit in memory
- Solution: Hashtable indexed by word; for each word, the count is stored (single scan though the file allows to fill the counts)

Computational Model

Case 2: <word, count> pairs cannot all fit in memory

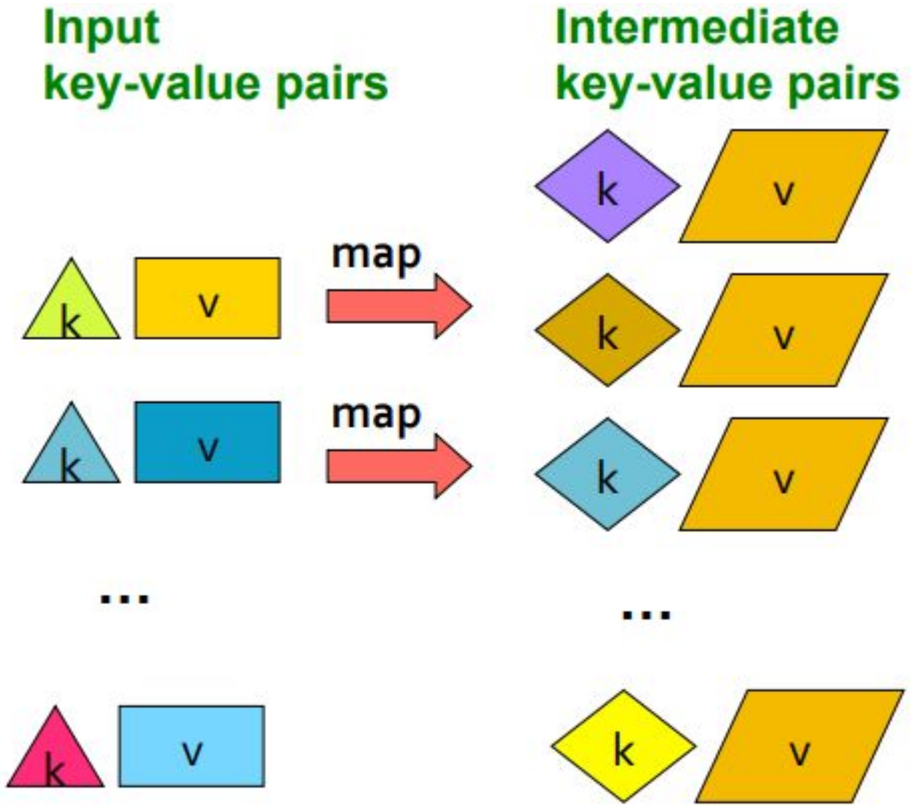
- Create a stream of words from documents
- Map each word to a pair <word, 1>
- Group the pairs by word
    - for now you can sort-of think of it as <word,k> on a single node
- Reduce all pairs referring to the same word to a single count across nodes
    - <word , n> showing that the word occurred n times

- Solution to Case 2 can be naturally made parallelizable
- Each task can be taken by a separate node
- No random accesses to disks are needed

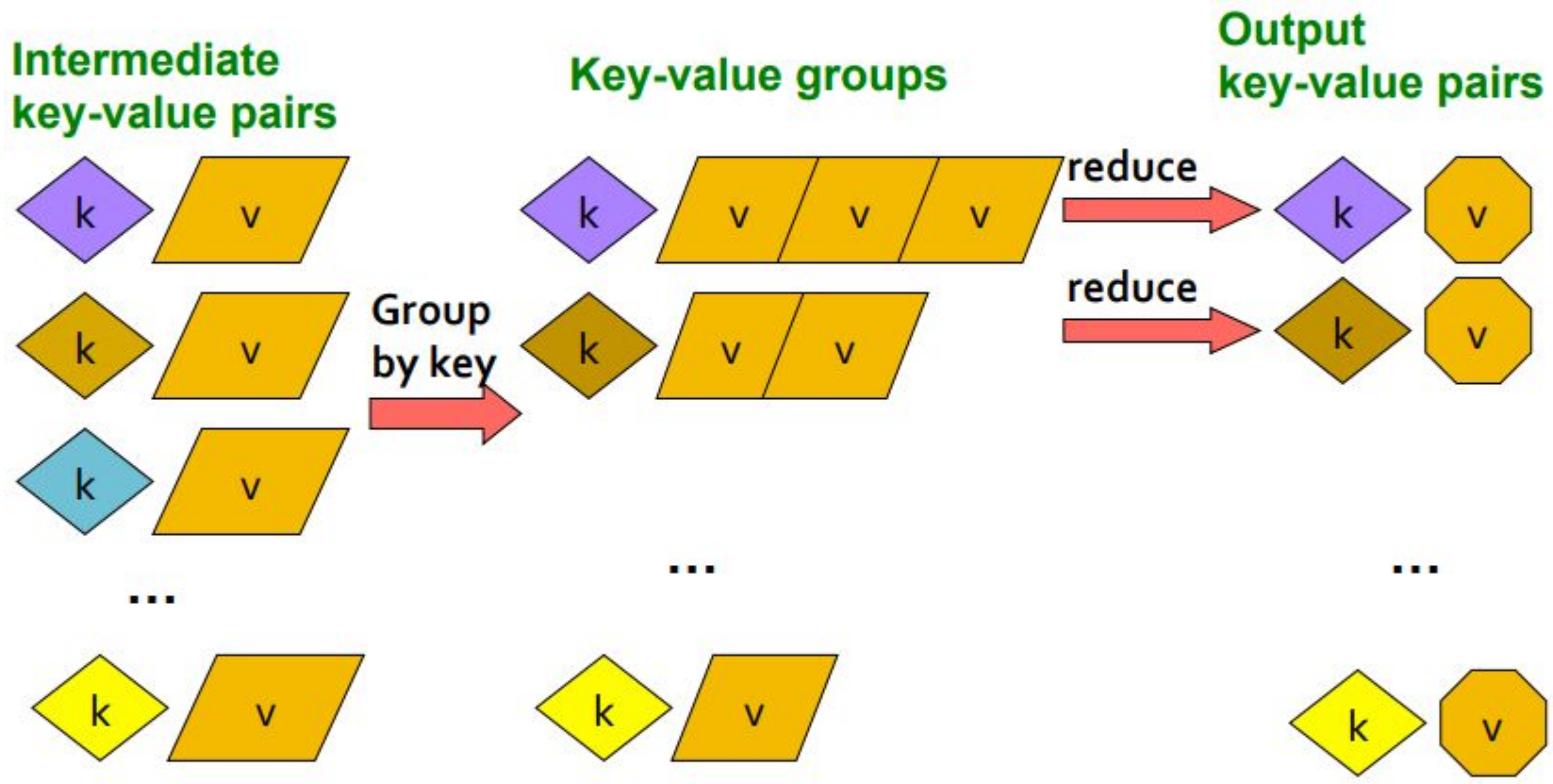
MapReduce: Overview

- Map
  - Scan input files to create a record stream
  - Extract something you care about from each record (key-value pairs)
- Group by key
  - Sort and shuffle
- Reduce
  - Aggregate, summarize, filter, or transform
  - Write the results

The Map Step



The Reduce Step



What is Spark?

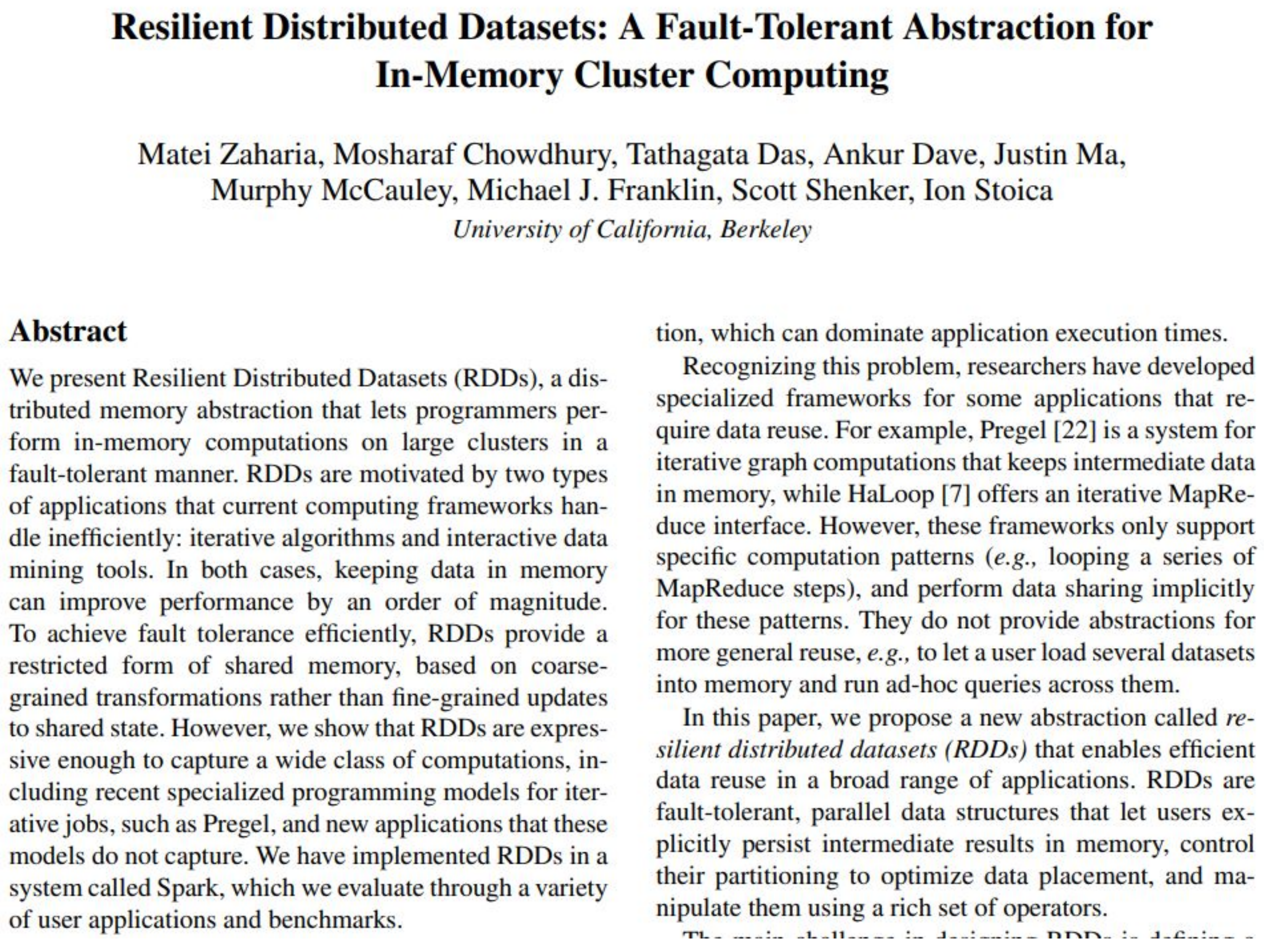
Apache Spark is a fast, in-memory data processing framework which allows data workers to efficiently execute tasks that require fast iterative access to datasets.

- Apache Spark is an open-source cluster-computing framework for real-time processing developed by the Apache Software Foundation.

- Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance.

- It was built on top of Hadoop MapReduce and it extends the MapReduce model to efficiently use more types of computations.

2011



Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma,
Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

Abstract

We present Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory, based on coarse-grained transformations rather than fine-grained updates to shared state. However, we show that RDDs are expressive enough to capture a wide class of computations, including recent specialized programming models for iterative jobs, such as Pregel, and new applications that these models do not capture. We have implemented RDDs in a system called Spark, which we evaluate through a variety of user applications and benchmarks.

tion, which can dominate application execution times.

Recognizing this problem, researchers have developed specialized frameworks for some applications that require data reuse. For example, Pregel [22] is a system for iterative graph computations that keeps intermediate data in memory, while HaLoop [7] offers an iterative MapReduce interface. However, these frameworks only support specific computation patterns (*e.g.,* looping a series of MapReduce steps), and perform data sharing implicitly for these patterns. They do not provide abstractions for more general reuse, *e.g.,* to let a user load several datasets into memory and run ad-hoc queries across them.

In this paper, we propose a new abstraction called *resilient distributed datasets (RDDs)* that enables efficient data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators.

Why Spark?

- It is more than 100x faster than other computational frameworks.

- It can be programmed in Scala, Java, Python, and R.

- It has powerful machine learning libraries for complex data analysis.

PySpark Installation

PySpark is a Python API for Spark that lets you harness the simplicity of Python and the power of Apache Spark in order to tame Big Data.

- Install **JDK SE 8**

- Add the path to JDK's bin folder to PATH

- Execute “pip install pyspark” or “conda install -c conda-forge pyspark”

This installation only contains the local mode of pyspark, which will be enough of our example. A full installation of Spark can be downloaded here.

Please move to this colab notebook for a PySpark tutorial.