Lec11. Cloud-based Data Analytics

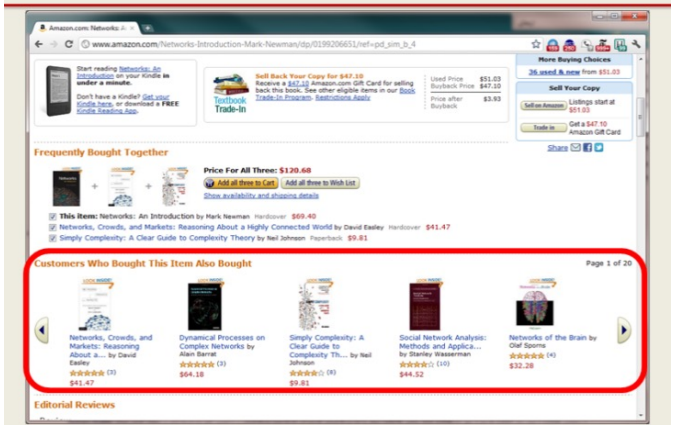
Last Week Recap: What is Recommendation?

- Which mobile phone should I buy?
- Where should I visit for my business trip?
- Whom should I follow on Twitter?
- Where should I invest my money?

- Which tour is the best for **our class**?

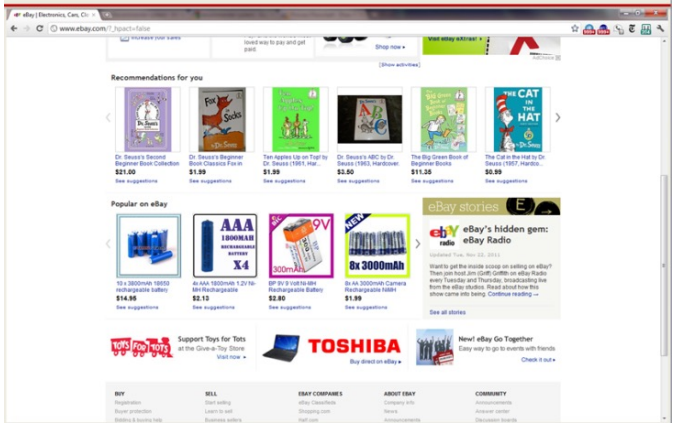
Last Week Recap: Recommender Systems

Book recommendation in Amazon

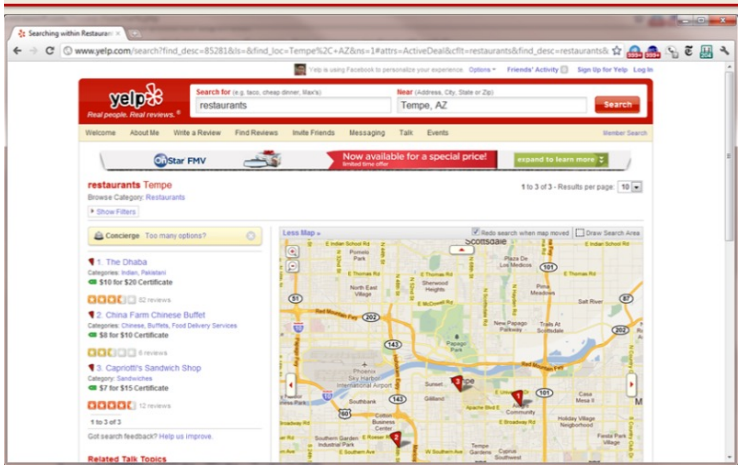


Video clip recommendation in YouTube

Product Recommendation in ebay



Restaurant Recommendation in Yelp



recommendation = personalized prediction

Last Week Recap: Types of Recommender Systems

1. Content-based recommendation:
   - Recommend **based on similarity** between user features and item features
2. Rating-based recommendation (Collaborative Filtering)
   - Recommend **based on rating** matrix
3. Hybrid recommendation
   - Use both **contents** and **ratings**
4. Clustering-based recommendation
   - Recommend **based on clusters of rating** matrix

Last Week Recap: RecSys Challenges

- **Cold-Start Problem**
  - Recommender systems use **historical data** or information provided by the user to recommend items, products, etc.
  - When user join sites, they still haven't bought any product, or they have no history.
  - It is hard to infer what they are going to like when they start on a site.
- **Data Sparsity**
  - When historical or prior information is **insufficient**.
  - Unlike the cold start problem, this is in the system as a whole and is not specific to an individual.
- **Attacks**
  - **Push Attack**: pushing ratings up by making fake users
  - **Nuke attack**: DDoS attacks, stop the whole recommendation systems
- **Explanation**
  - Recommender systems often recommend items with **no explanation** on why these items are recommended

Course structure

W1. Data Processing with Python

W2. Data Exploration with Python

W3. Data Modeling with Pytyhon

W4. Data Analytics for Timeseries

Holiday

W5-6-7. Data Analytics for Texts

W8. Data Analytics for Images

W9. Data Analytics for Graphs

W10-11. **Data Analytics for Other Data**

W12. Revision

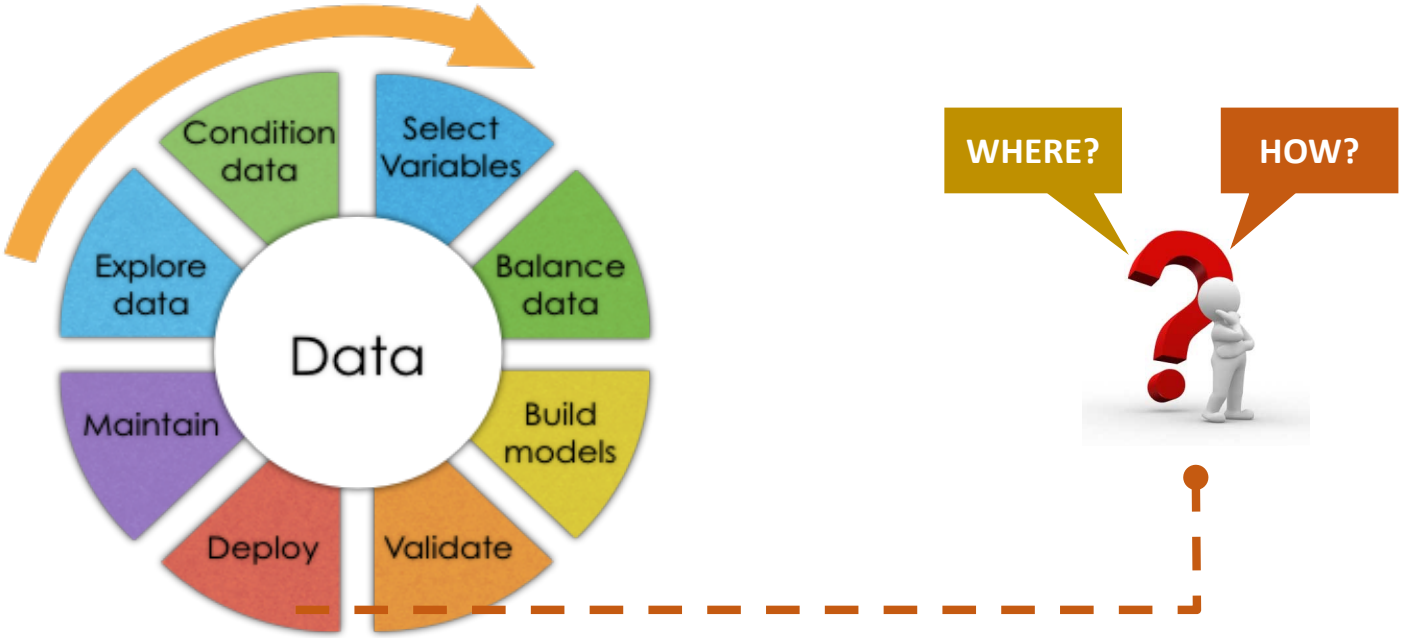
Cloud-based Data Analytics

I. **Cloud Computing: Concepts**
II. Cloud-based Data Analytics with Hadoop
III. Could-based Data Analytics with Spark

1. Introduction

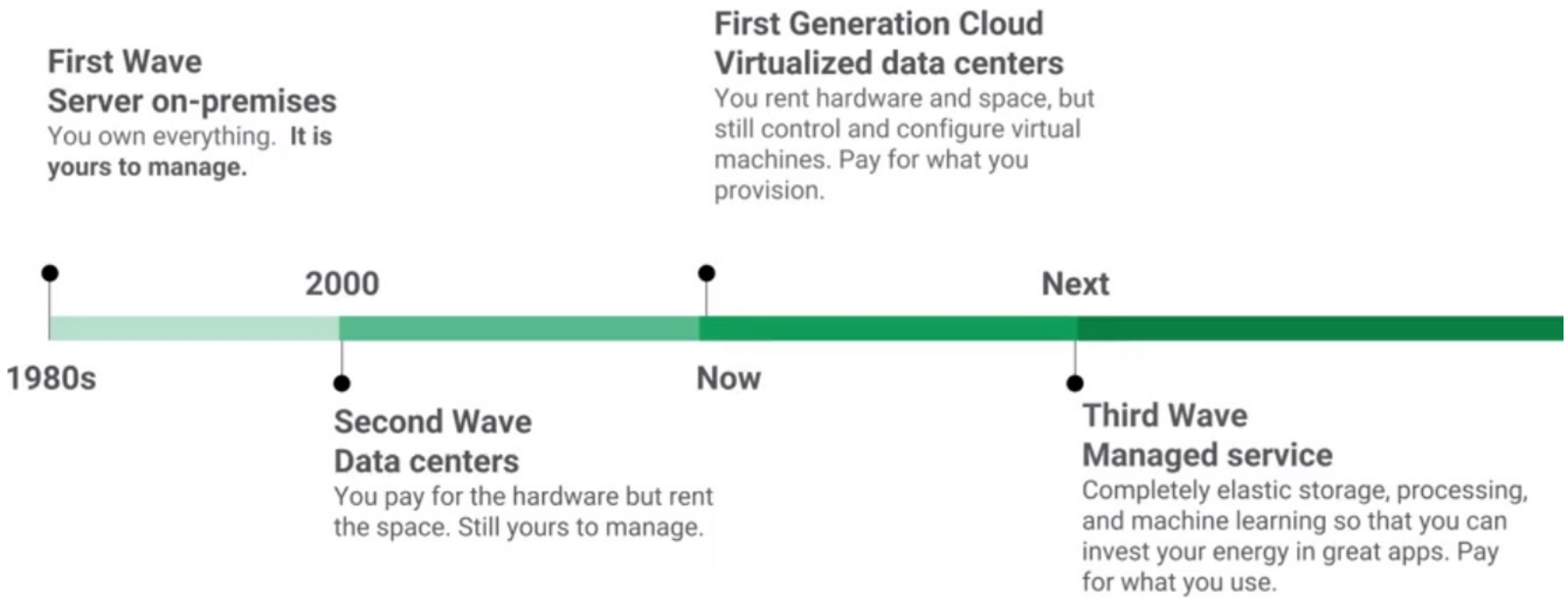
- Cloud computing is **Internet-based computing**, whereby **shared resources**, software, and information are provided to computers and other devices **on demand**, like the electricity grid.
- Cloud computing is a style of computing in which **dynamically scalable** and often virtualized resources are provided as a service over the Internet.
- Cloud computing is a general term for anything that involves **delivering hosted services over the Internet.**

Data Lifecycle

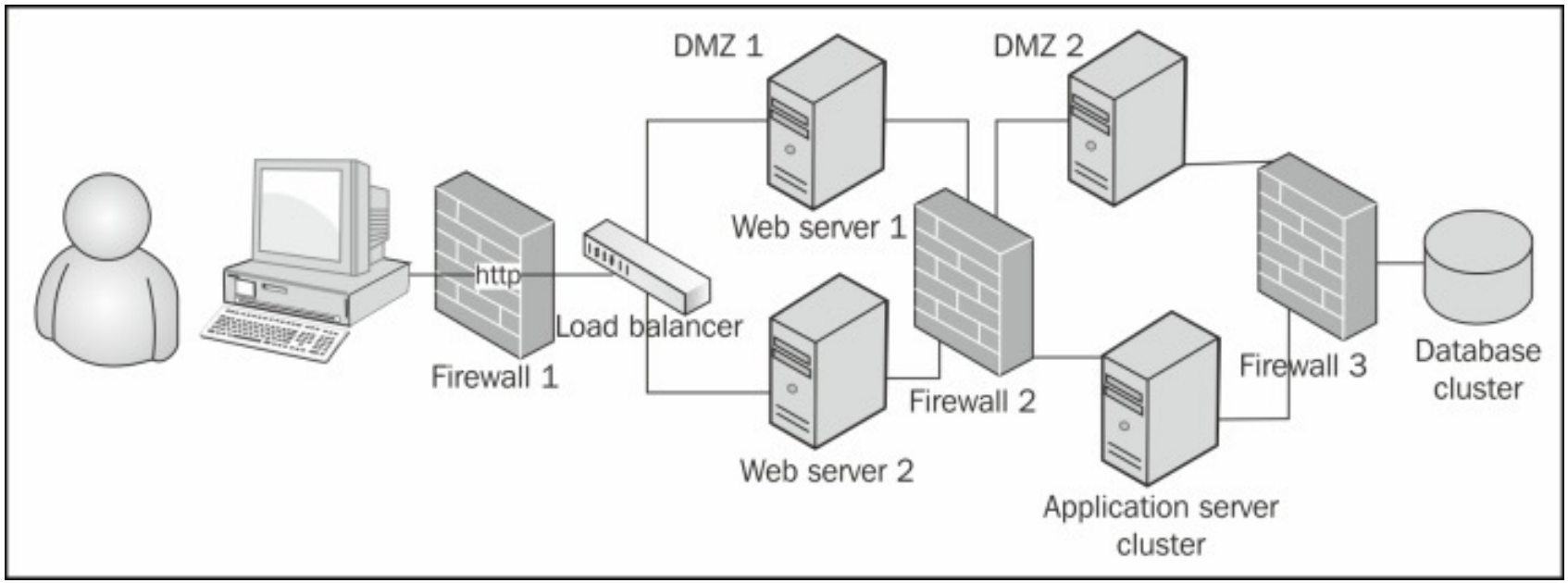


http://www.evolved-analytics.com/sites/default/files/modelLifeCycleSmall.png

Timeline


First Wave
Server on-premises
You own everything. It is yours to manage.
First Generation Cloud
Virtualized data centers
You rent hardware and space, but still control and configure virtual machines. Pay for what you provision.
1980s
2000
Now
Next
Second Wave
Data centers
You pay for the hardware but rent the space. Still yours to manage.
Third Wave
Managed service
Completely elastic storage, processing, and machine learning so that you can invest your energy in great apps. Pay for what you use.

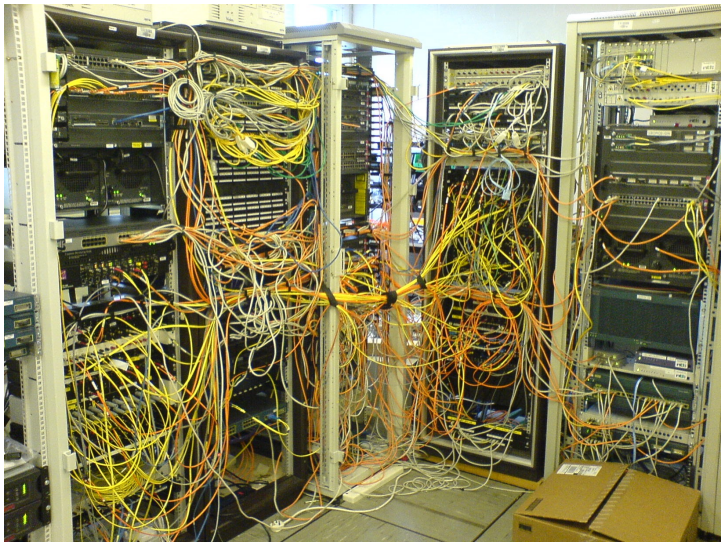
On-Premises Server



https://www.oreilly.com/library/view/java-ee-7/9781782176428/graphics/6428EN_09_05.jpg

On-Premises Disadvantages

- Requires IT resources to **set up and maintain** hardware and software.
  → Not suitable for small/medium companies.
- Requires hardware and infrastructure **capital** investment.
- **Limited scalability** with growth of organization.
- Need for **space** to build/store hardware.
- More prone to **data loss** in **emergency** or **disaster** situations.



Cloud

- **Shared pools** of configurable computer system resources and higher-level services.
- Rely on **sharing of resources** to achieve coherence and economies of scale, similar to a public utility.

Source: https://en.wikipedia.org/wiki/Cloud_computing

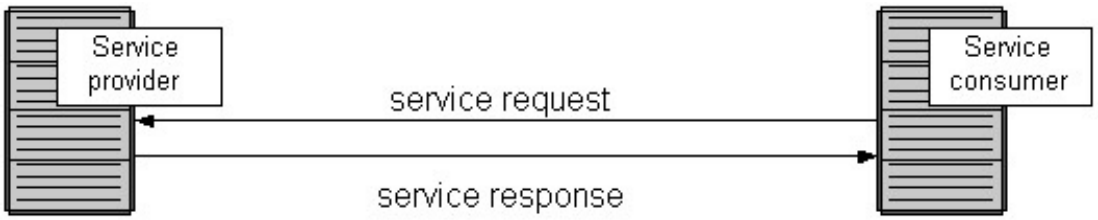
Core Ideas

- Perspective from user :
  - Users do not care about how the works are done
    - ✓ Instead, they only concern about what they can get
  - Users do not care about what the provider actually did
    - ✓ Instead, they only concern about their quality of service
  - Users do not want to own the physical infrastructure
    - ✓ Instead, they only want to pay as many as they used

- What does user really care ?
  - They only care about their “Service”

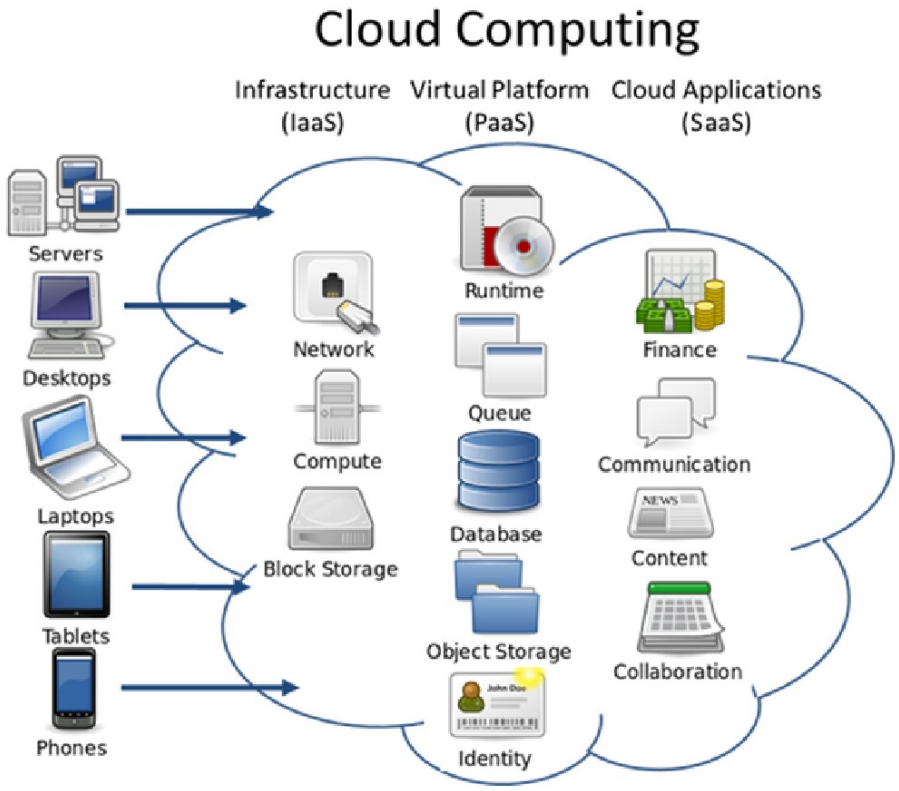
2. Characteristics

1. On demand self-services: use web services via network
2. Rapid scalability: scale up/down wrt demands
3. Multi-tenancy: Resource Pooling for sharing applications/ infrastructure
4. Pay-as-you-go Service
5. Fault-Tolerance: continue operating properly in the event of the failure of some of its components.
6. Security & Privacy
7. Portability: access services using any devices, anywhere



3. Service Models

- IaaS
- PaaS
- Saas
- …



Cloud Infrastructure as a Service (IaaS)

- The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources.
- The consumer is able to deploy and run arbitrary software, which can include operating systems and applications.
- The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).
- *Examples: Amazon EC2, GoGrid, iland, Rackspace Cloud Servers, ReliaCloud.*

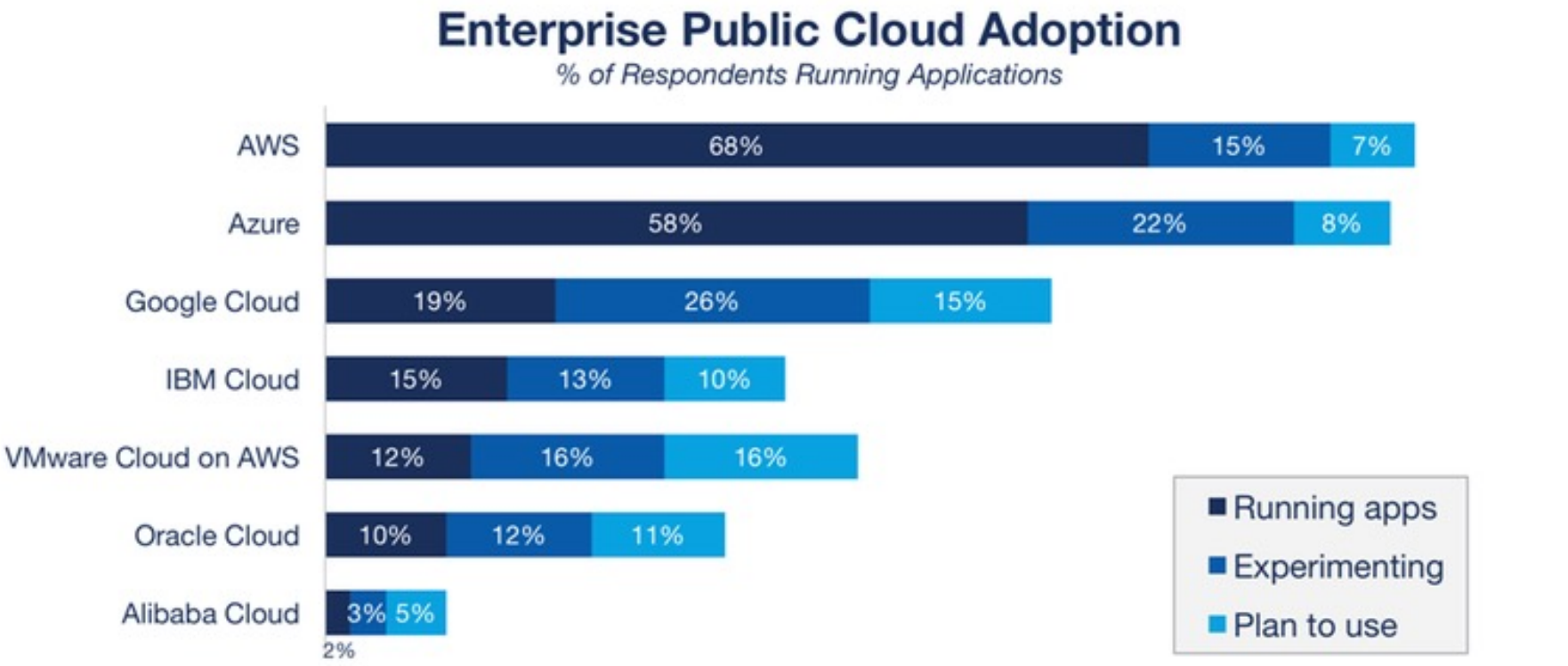
Cloud Platform as a Service (PaaS)

- The capability provided to the consumer is to deploy onto the cloud infrastructure *consumer-created* or *acquired applications* created using *programming languages and tools* supported by the provider.
- The consumer does not manage or control the underlying cloud infrastructure.
- Consumer has control over the deployed applications and possibly application hosting environment configurations.
- *Examples: Windows Azure, Google App.*

Cloud Software as a Service (SaaS)

- The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure.
- The applications are accessible from various client devices such as a web browser (e.g., web-based email).
- The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage,…
- *Examples: Caspio, Google Apps, Salesforce, Nivio, Learn.com.*

Market Share



Source: RightScale 2018 State of the Cloud Report

Cloud-based Data Analytics

I. Cloud Computing: Concepts

II. Cloud-based Data Analytics with Hadoop

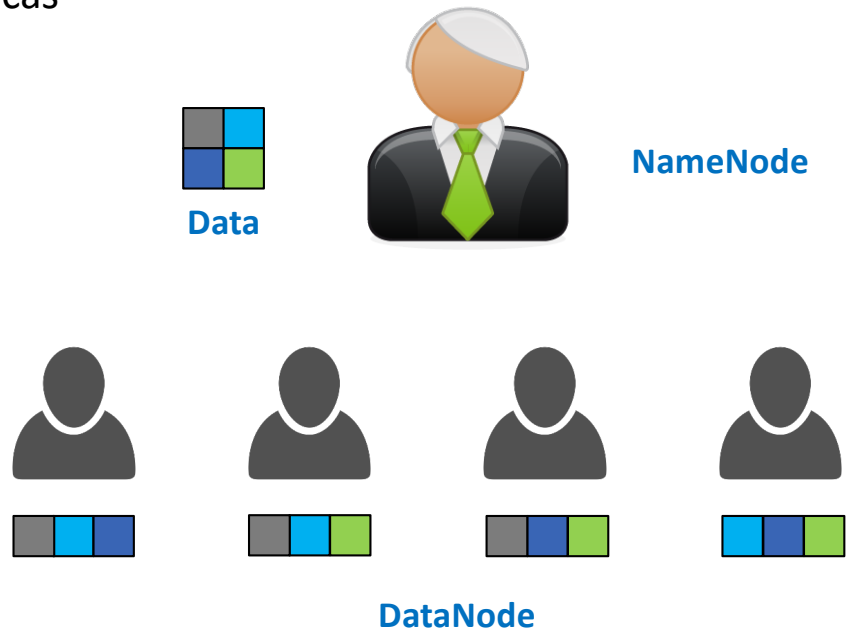
III. Could-based Data Analytics with Spark

What is Hadoop?

- ✓ Data storage layer
  - Local file systems are integrated to serve as one.
    - Hadoop Distributed File System (HDFS)
- ✓ Data processing layer
  - Tasks are divided and distributed to each node and processed locally
    - MapReduce Programming Model

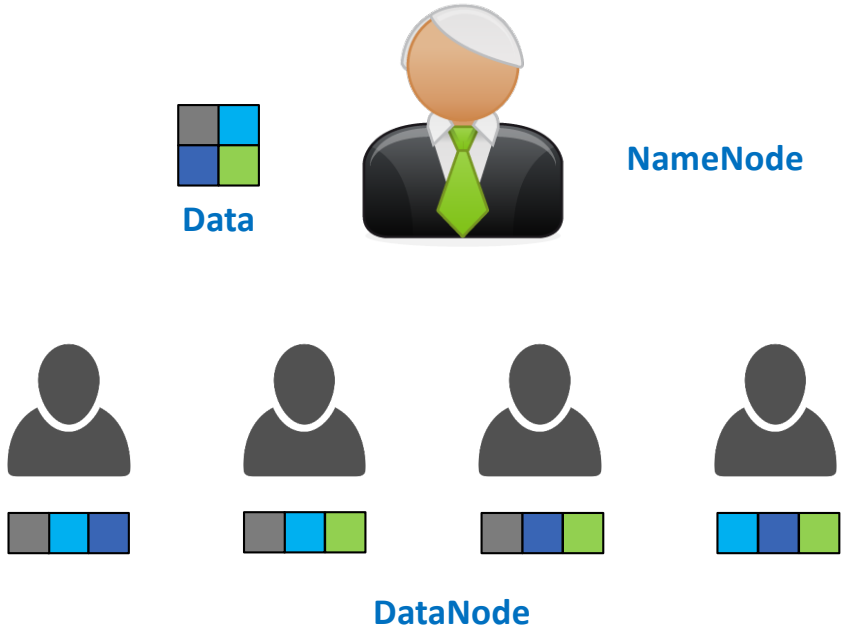
Hadoop: Storage model - Distributed File System (HDFS)

- Master(NameNode)
  - ✓ Cut the input data into blocks
  - ✓ Decide the destination of the blocks and replicas
  - ✓ Store the metadata of DataNodes
  - ✓ Monitor the DataNodes
- Slave(DataNode)
  - ✓ Store the data blocks
  - ✓ Provide access



Hadoop: Storage model - Distributed File System (HDFS)

- Key/value Pair
  - ✓ Data structure: (*key, value*)
  - ✓ Both key and value can be any format: String, integer or even object.
    - A card → (spade, four), A video → (video_name, video)
  - ✓ The design of the k/v pair is flexible
    - A document →
      - (docID, content)
      - (word1, count), (word2, count),…

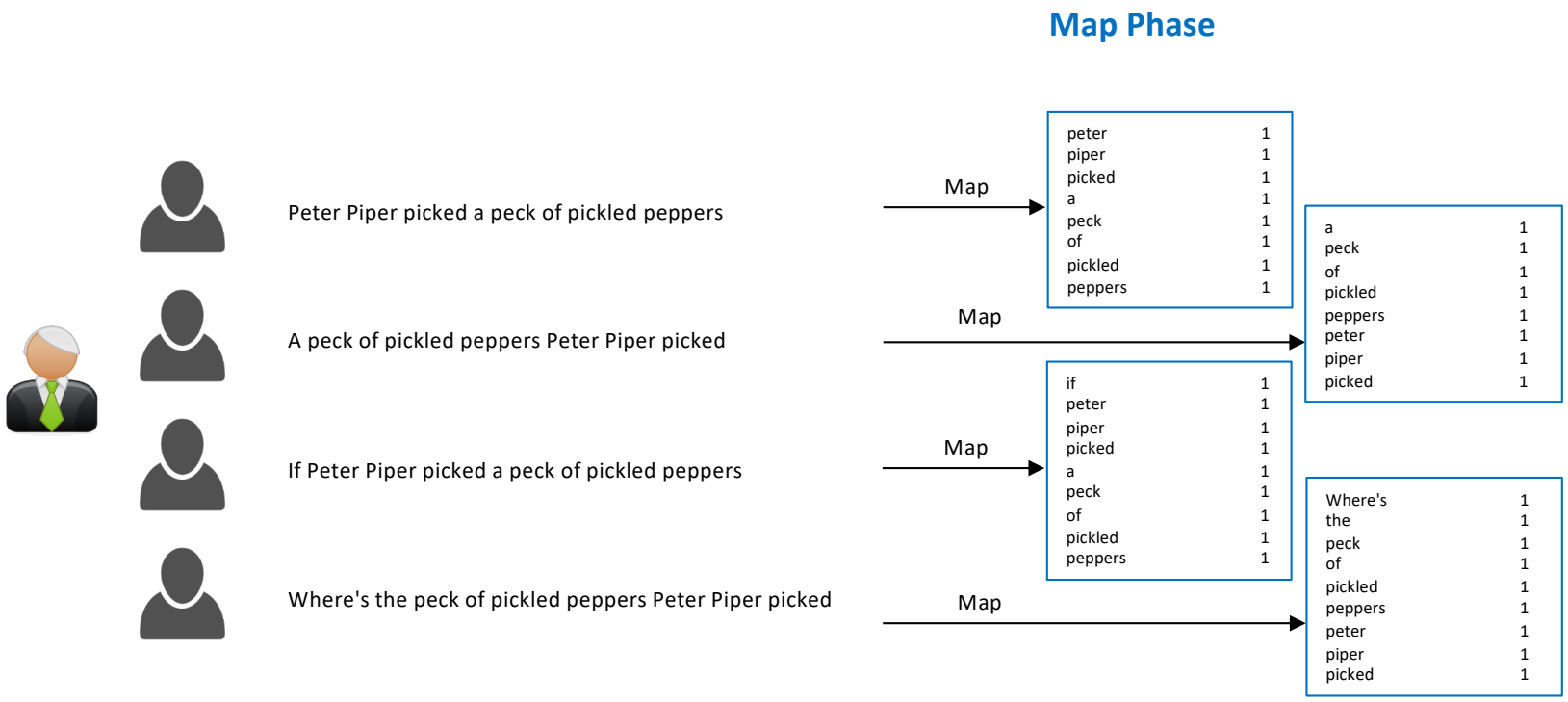


MapReduce Programming Model

- MapReduce Model
  - ✓ Map Phase: Programmable
    - Input: a set of files
    - Output: list of (key, value)
  - ✓ Shuffle Phase
    - Send k/v pairs of the same key to the same server
  - ✓ Reduce Phase: Programmable
    - Input: (key, list of {value})
    - Output: (key, value) or output files

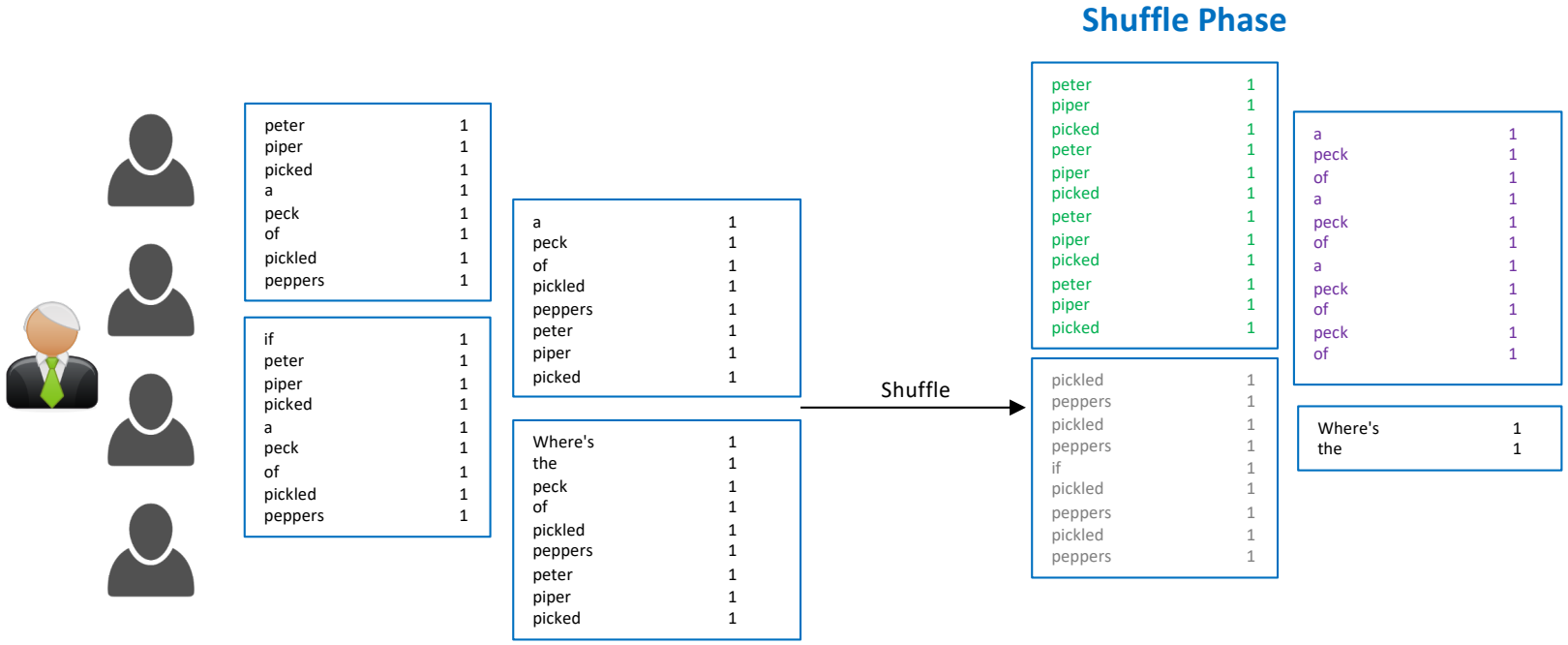
MapReduce Programming Model

- Word Count
  - ✓ *Peter Piper picked a peck of pickled peppers,*
    A peck of pickled peppers Peter Piper picked,
    If Peter Piper picked a peck of pickled peppers,
    Where's the peck of pickled peppers Peter Piper picked?



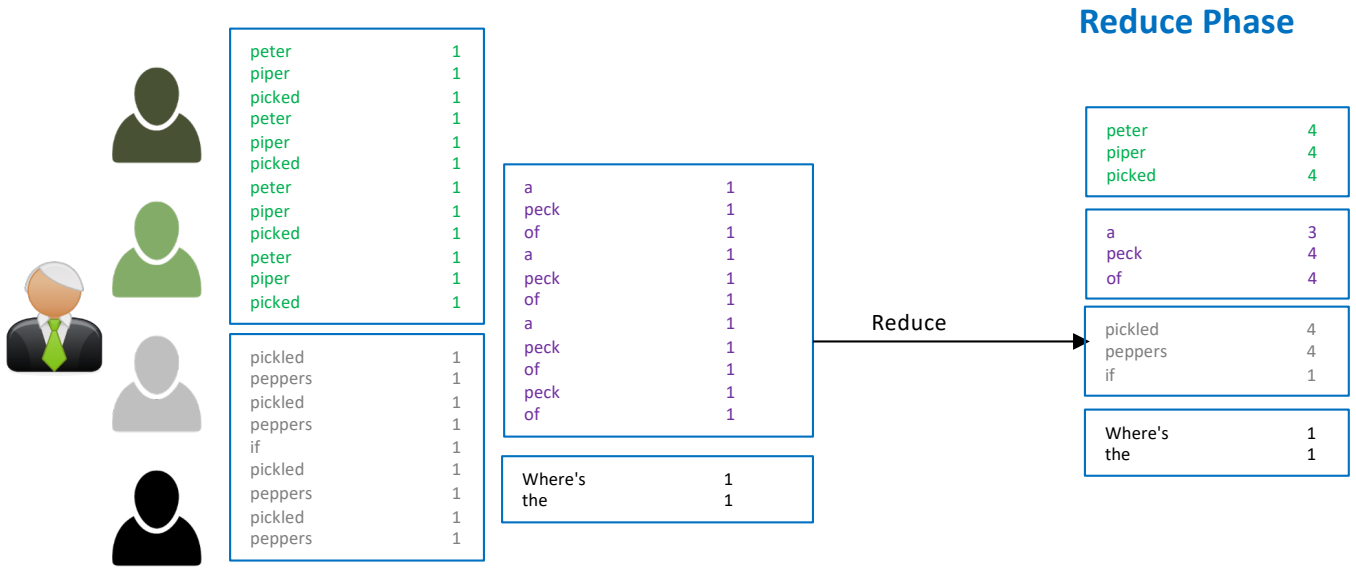
MapReduce Programming Model

- Word Count
  - ✓ *Peter Piper picked a peck of pickled peppers,*
    A peck of pickled peppers Peter Piper picked,
    If Peter Piper picked a peck of pickled peppers,
    Where's the peck of pickled peppers Peter Piper picked?



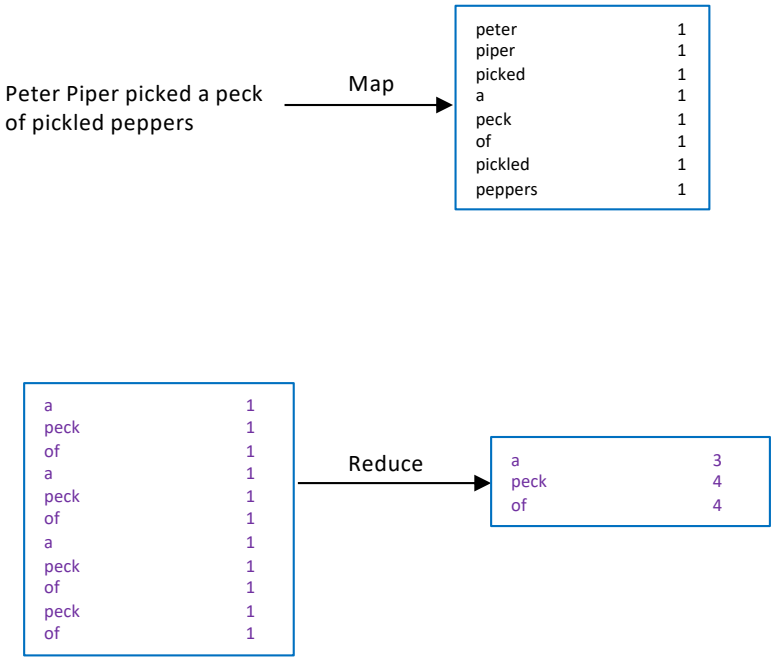
MapReduce Programming Model

- Word Count
  - ✓ *Peter Piper picked a peck of pickled peppers,*
    A peck of pickled peppers Peter Piper picked,
    If Peter Piper picked a peck of pickled peppers,
    Where's the peck of pickled peppers Peter Piper picked?



MapReduce Programming Model

- Key-Value Pair
  - ✓ (peter,1), (piper,1)
- Map Phase
  - ✓ Input: documents
  - ✓ Output: list of (word, count)
- Shuffle Phase
- Reduce Phase
  - ✓ Input: (word, list of {count})
  - ✓ Output: list of (word, total count)
- MapReduce model is popular
  - ✓ Simple
  - ✓ Versatile
  - ✓ Scalable



Cloud-based Data Analytics

I. Cloud Computing: Concepts

II. Cloud-based Data Analytics with Hadoop

III. Could-based Data Analytics with Spark

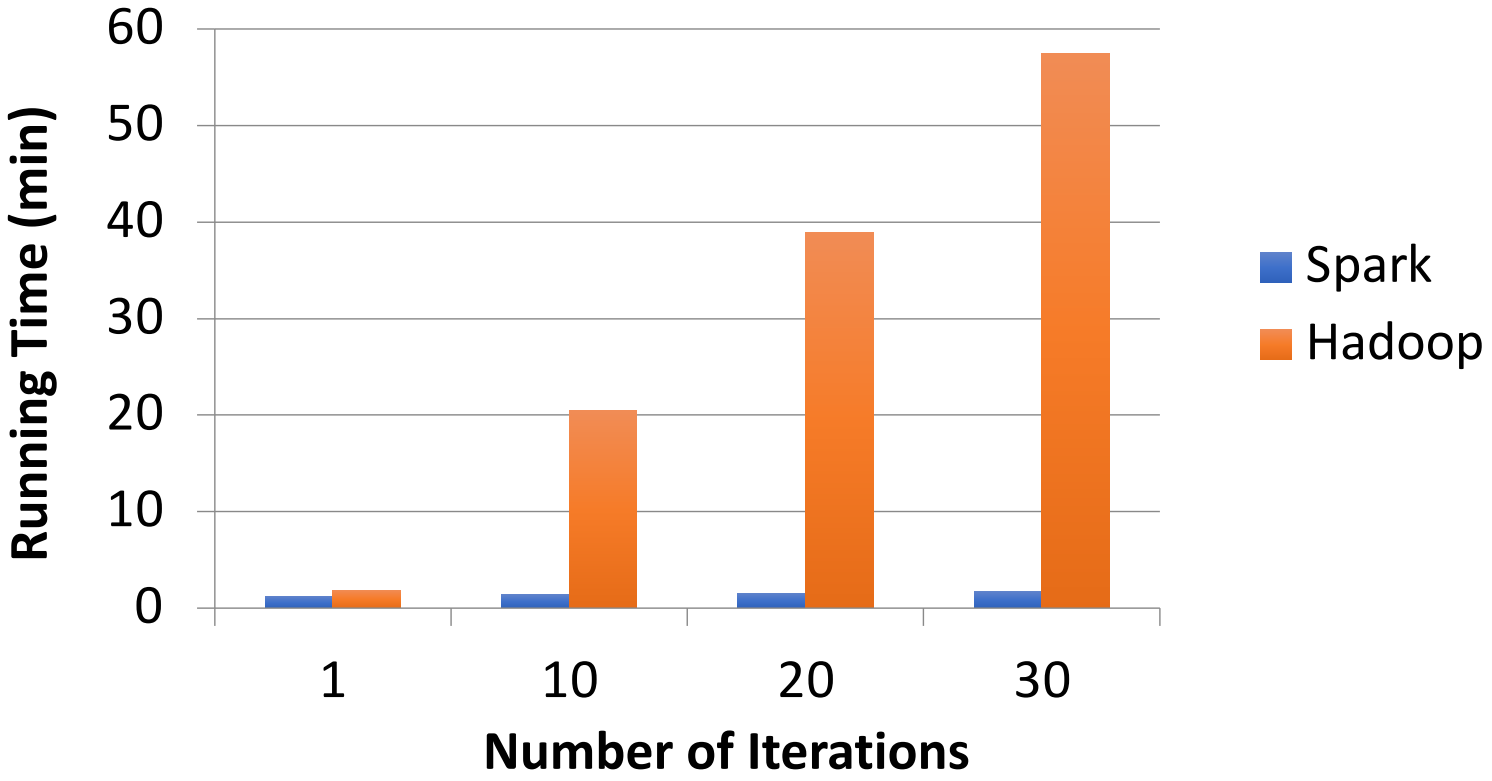
Spark: In-Memory Distributed Data Analysis

Spark Features

- Fast version of Hadoop
  - ✓ **In-memory system:** significantly reduce the disk write
  - ✓ Performs better when the memory size is large
- Data Storage
  - ✓ Support HDFS
  - ✓ Other distributed/cloud file system (HBase, Amazon S3, etc.)
- Data Processing
  - ✓ More operations
  - ✓ Pipeline mode based on RDDs(Resilient Distributed Datasets)
- More language supported
  - ✓ Scala
  - ✓ Python
  - ✓ Java
  - ✓ R

Spark Performance

Logistic Regression:

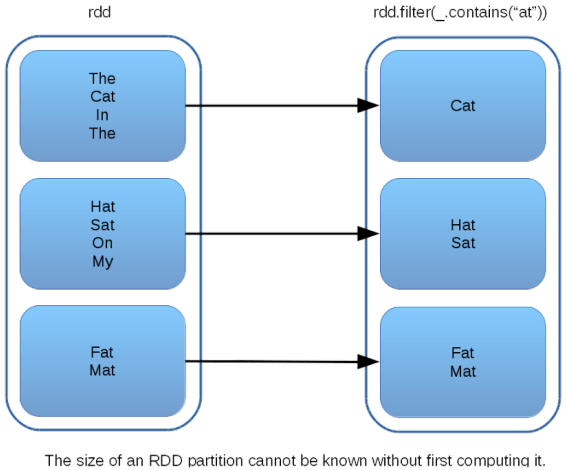


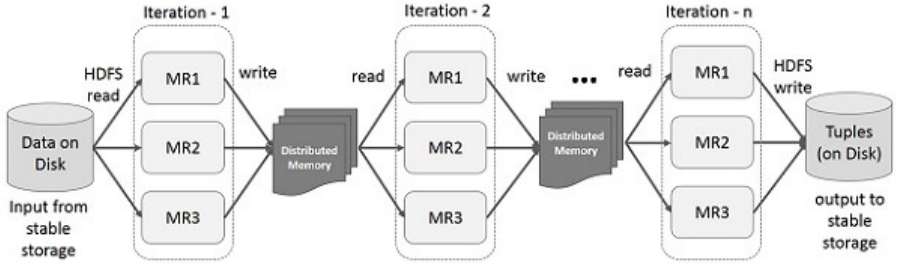
Spark Performance

Iterative algorithms:

Resilient Distributed Datasets (RDD)

- Fundamental Data Structure in Spark
  - ✓ Operation
  - ✓ Data partitions
- Operation
  - ✓ Transformations
    - map(), flatMap(), filter(), groupByKey()
  - ✓ Actions
    - count(), collect(), reduce(), take()
- Lazy Execution
  - ✓ Transformations will not run until actions are called
  - ✓ High efficiency
- Fault Tolerance
  - ✓ Rerun the RDDs
- Performance Advantage
  - ✓ Complex tasks
    - Iterative tasks

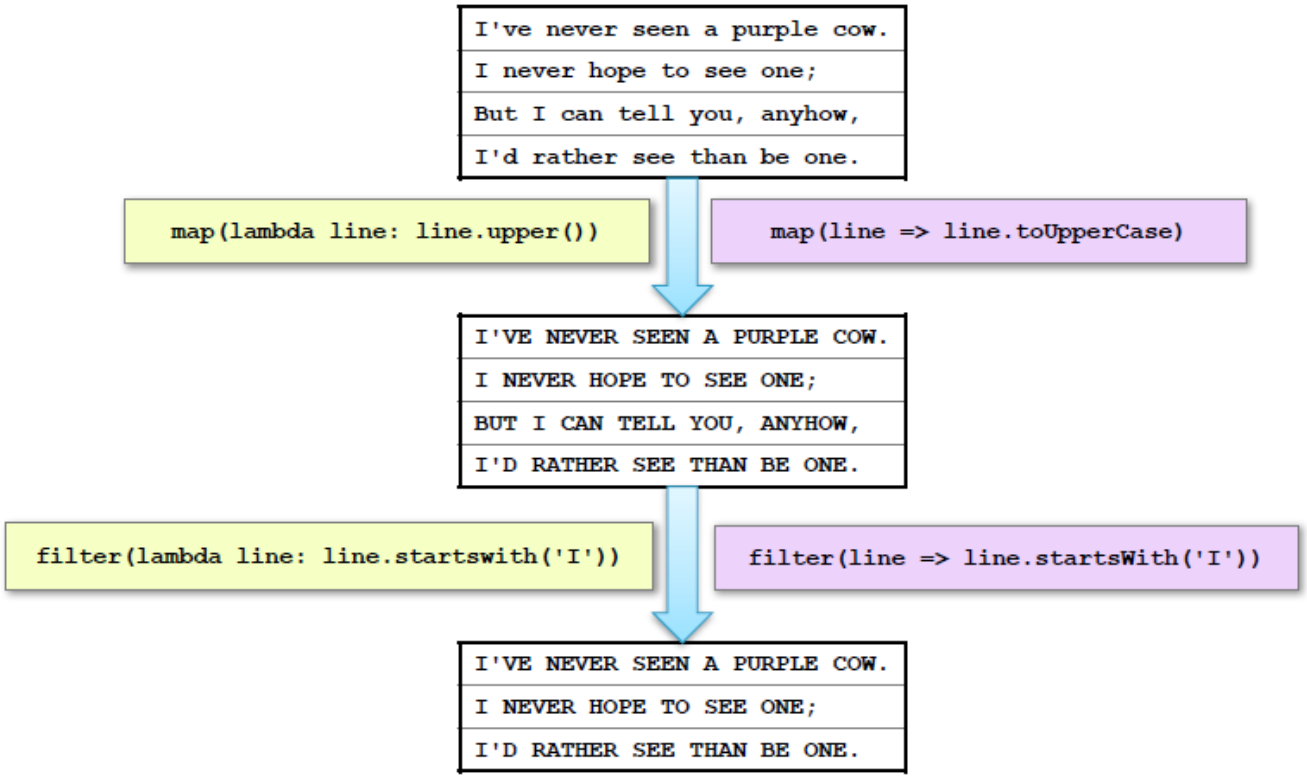




RDD Actions

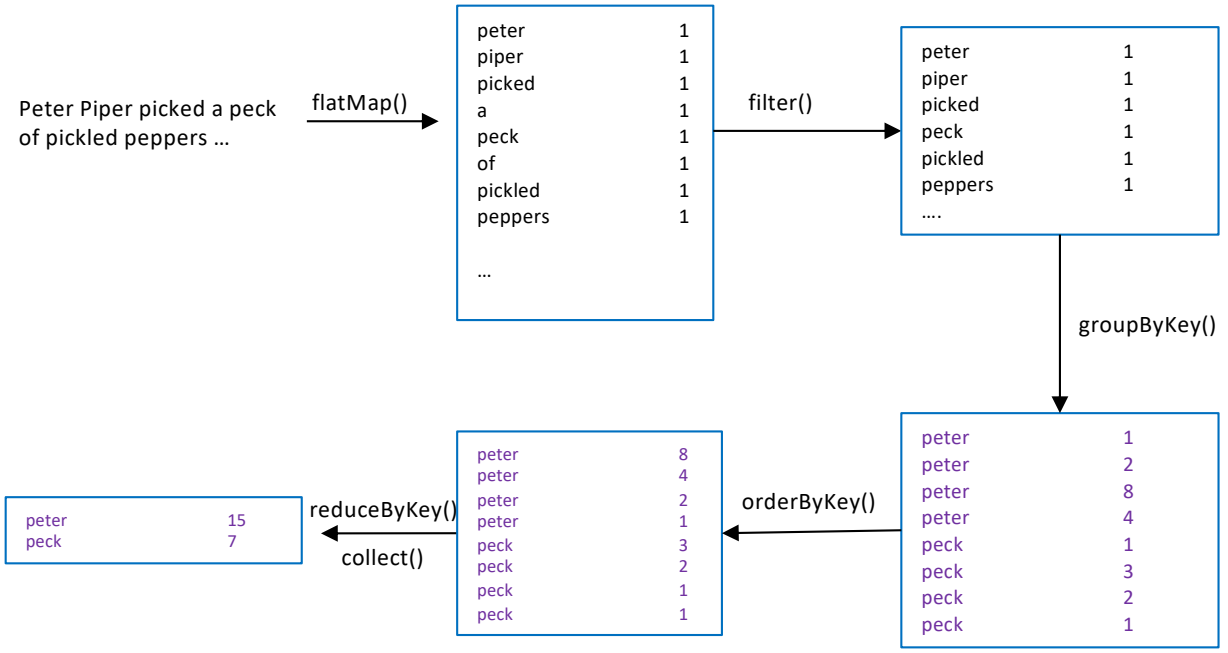
- Apply transformation chains on RDDs, eventually performing some additional operations (e.g., counting).
- Some actions only store data to an external data source (e.g. HDFS), others fetch data from the RDD (and its transformation chain) upon which the action is applied, and convey it to the driver.
- Some common actions
  - ✓ count() – return the number of elements
  - ✓ take(*n*) – return an array of the first *n* elements
  - ✓ collect()– return an array of all elements
  - ✓ saveAsTextFile(*file*) – save to text file(s)

Example: *Map* and *Filter* Transformations



Resilient Distributed Datasets (RDD)

- Word Count
  - ✓ Easy to use



Spark Examples

Spark Example 1. PageRank

- Give pages ranks (scores) based on links to them
  - ✓ Links from many pages ➔ high rank
  - ✓ Link from a high-rank page ➔ high rank

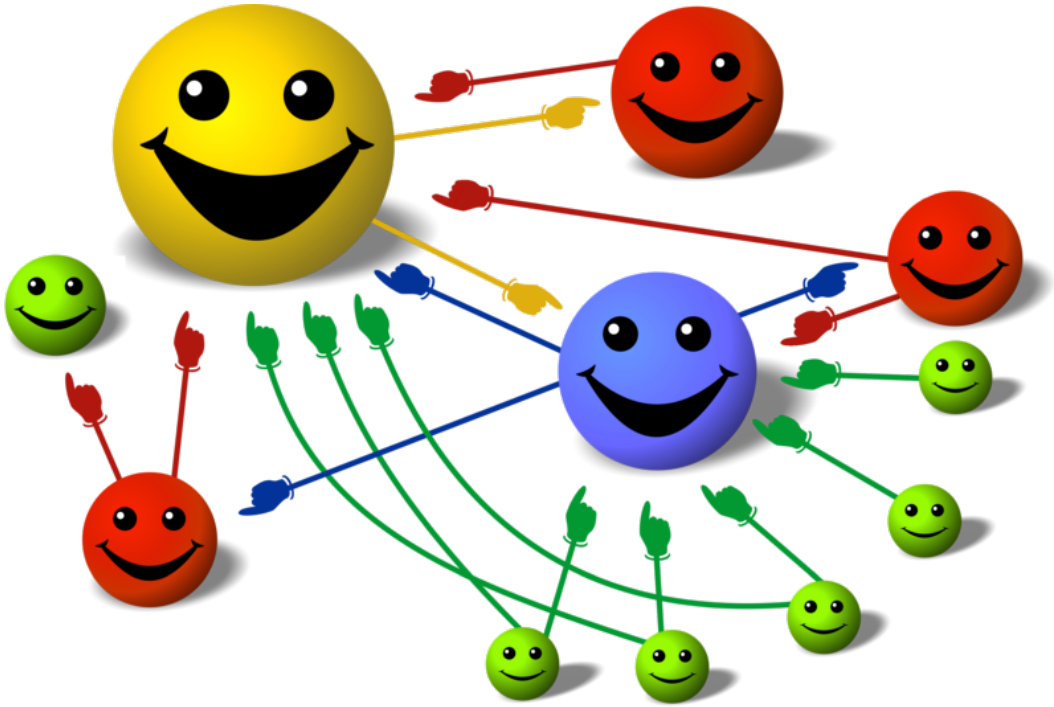
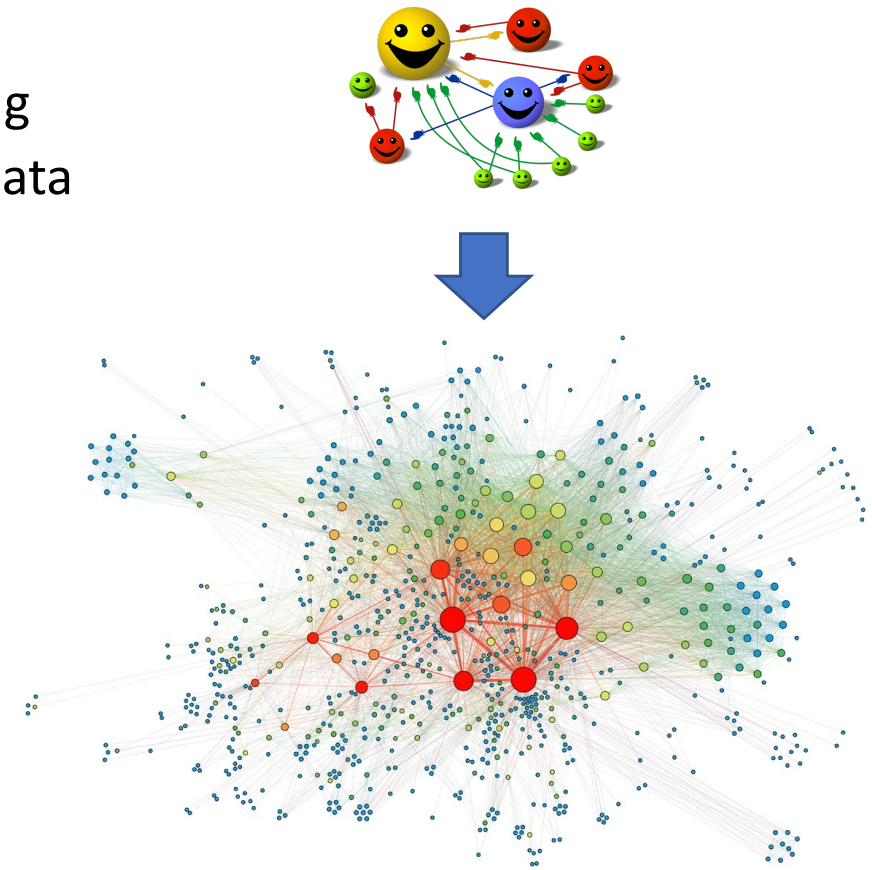


Image: en.wikipedia.org/wiki/File:PageRank-hi-res-2.png

Problem

- Good example of a more complex algorithm
  - ✓ Multiple stages of map & reduce
- Benefits from Spark's in-memory caching
  - ✓ Multiple iterations over the same data

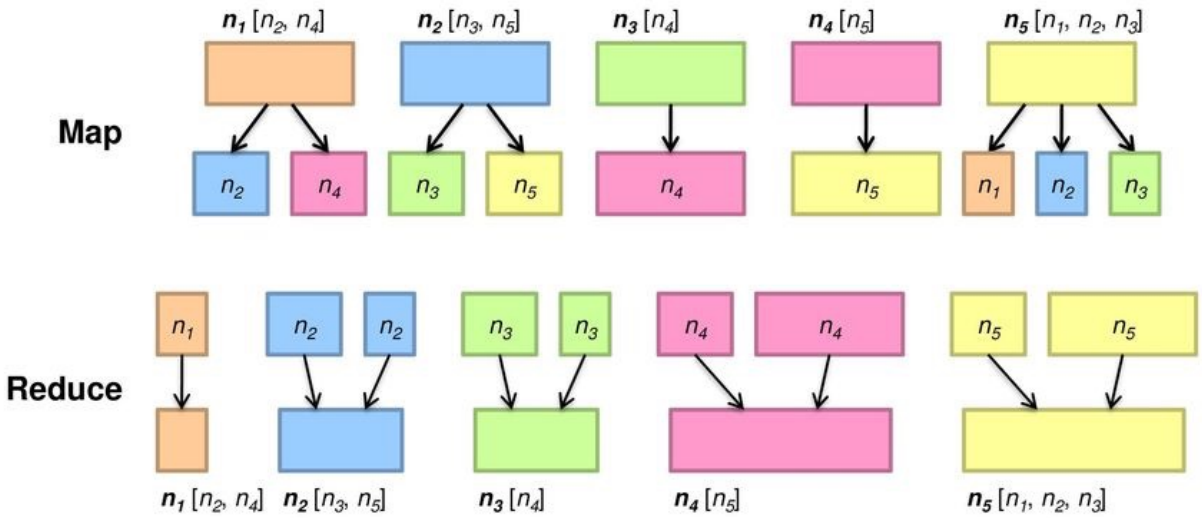


PageRank Implementation

```
# Parameter M adjacency matrix where M_i,j represents the link from 'j' to 'i', such that for all 'j'
# sum(i, M_i,j) = 1
# Parameter d damping factor (default value 0.85)
# Parameter eps quadratic error for v (default value 1.0e-8)
# Return v, a vector of ranks such that v_i is the i-th rank from [0, 1]

import numpy as np

def pagerank(M, eps=1.0e-8, d=0.85):
    N = M.shape[1]
    v = np.random.rand(N, 1)
    v = v / np.linalg.norm(v, 1)
    last_v = np.ones((N, 1), dtype=np.float32) * 100
    M_hat = (d * M) + (((1 - d) / N) * np.ones((N, N), dtype=np.float32))

    while np.linalg.norm(v - last_v, 2) > eps:
        last_v = v
        v = np.matmul(M_hat, v)
    return v

M = np.array([[0, 0, 0, 0, 1],
              [0.5, 0, 0, 0, 0],
              [0.5, 0, 0, 0, 0],
              [0, 1, 0.5, 0, 0],
              [0, 0, 0.5, 1, 0]])
v = pagerank(M, 0.001, 0.85)
```

https://en.wikipedia.org/wiki/PageRank

PageRank in MapReduce



PageRank MapReduce Pseudocode

```
map( key: [url, pagerank], value: outlink_list )
    for each outlink in outlink_list
        emit( key: outlink, value: pagerank/size(outlink_list) )

    emit( key: url, value: outlink_list )

reducer( key: url, value: list_pr_or_urls )
    outlink_list = []
    pagerank = 0

    for each pr_or_urls in list_pr_or_urls
        if is_list( pr_or_urls )
            outlink_list = pr_or_urls
        else
            pagerank += pr_or_urls

    pagerank = 1 - DAMPING_FACTOR + ( DAMPING_FACTOR * pagerank )

    emit( key: [url, pagerank], value: outlink_list )
```

PageRank with Spark

Initialize

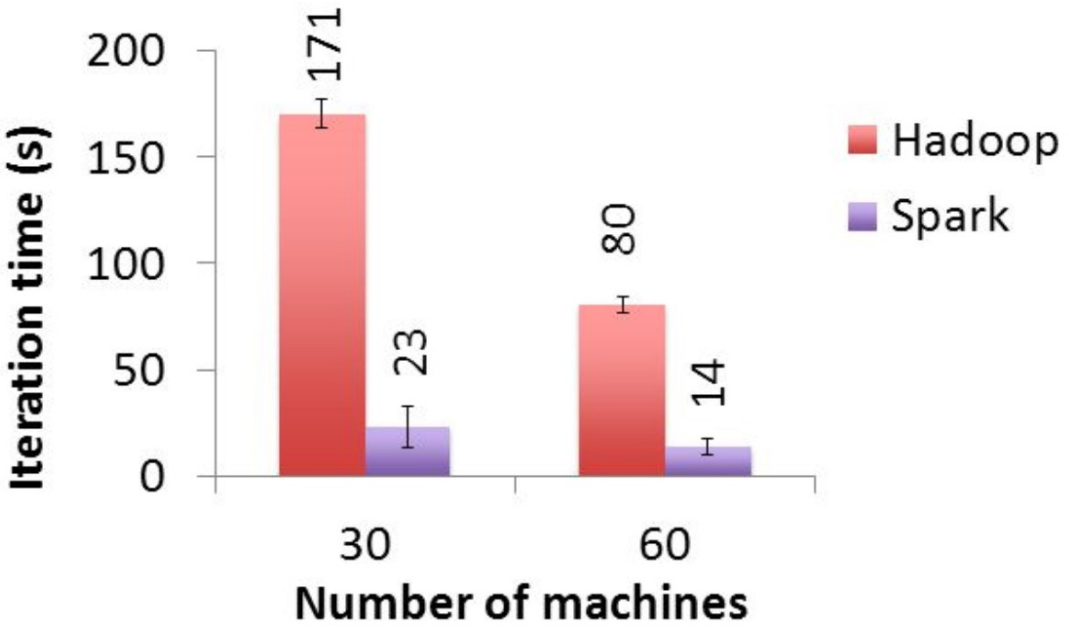
```
# Initialize the spark context.
spark = SparkSession\
    .builder\
    .appName("PythonPageRank")\
    .getOrCreate()

# Loads in input file. It should be in format of:
#     URL         neighbor URL
#     URL         neighbor URL
#     URL         neighbor URL
#     ...
lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])

# Loads all URLs from input file and initialize their neighbors.
links = lines.map(lambda urls: parseNeighbors(urls)).distinct().groupByKey().cache()

# Loads all URLs with other URL(s) link to from input file and initialize ranks of them to one.
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))
```

Calculate

```
# Calculates and updates URL ranks continuously using PageRank algorithm.
for iteration in range(int(sys.argv[2])):
    # Calculates URL contributions to the rank of other URLs.
    contribs = links.join(ranks).flatMap(
        lambda url_urls_rank: computeContribs(url_urls_rank[1][0], url_urls_rank[1][1]))

    # Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(add).mapValues(lambda rank: rank * 0.85 + 0.15)

# Collects all URL ranks and dump them to console.
for (link, rank) in ranks.collect():
    print("%s has rank: %s." % (link, rank))

spark.stop()
```

https://github.com/apache/spark/blob/master/examples/src/main/python/pagerank.py

PageRank with Spark



Spark Example 2. Word Count

- **Single Thread Computation:**

```
HashMap<String,Long> wordCount = new HashMap<>();
...
fileLines.forEach(line -> {
      String[] words = line.split(" ");
      for (String word : words) {
            if (!wordCount.containsKey(word)) {
                  wordCount.put(word, 0);
            }
            long count = wordCount.get(word);
            wordCount.put(word, count+1);
      }
});
```

Name <K>	Count <V>
Paul	100
Paula	95
Phoenix	110
Jessica	108
Jennifer	86
John	75
Jane	112
.....	

Word Count

- **Spark (Distributed in-memory computation):**

```
val textFileRDD = sc.textFile("../file.txt")

val countsRDD = textFileRDD
      .flatMap(line => line.split(" "))
      .map(word => (word, 1))
      .reduceByKey((count1, count2) => count1 + count2)
```

Summary

I. Cloud Computing: Concepts
   - Data Lifecycle
   - Cloud
   - Service Models

II. Cloud-based Data Analytics with Hadoop
   - HDFS
   - MapReduce Programming Model

III. Could-based Data Analytics with Spark
   - In-memory version of Hadoop
   - But better, with more features and more programming languages supported

References

[1] https://hadoop.apache.org/

[2] https://www.netapp.com/knowledge-center/what-is-cloud-analytics/

[3] https://spark.apache.org/