NYU POLYTECHNIC SCHOOL OF ENGINEERING

Virtualization and Parallel Programming in Data Centers

Part II

Cloud Programming

Outline

- Hadoop
  - MapReduce
  - HDFS (Hadoop Distributed File System)
- Beyond Hadoop
  - Iterative & interactive algorithms
  - Streaming data

Motivation

- Peta-scale datasets and their processing are everywhere on the cloud:
  - In 2020, there are over 2 trillion Google searches per day.
  - In Aug 2020, the total number of people who use YouTube – 1.3 billion. 300 hours of video are uploaded to YouTube every minute. Almost 5 billion videos are watched on Youtube per day.
  - In Sept 2020, Amazon had over 2.44 billion combined desktop and mobile visits, up from 2.01 billion visits in February 2020.
  - In Oct 2020, Facebook has over 2.74 billion monthly active users. Its Ad Revenues for Q3: $21.221 billion up 22 percent year-over-year.

- Big Data makes simple programming difficult.

Goals

- The key of cloud programming is to write reliable, scalable, and distributed applications
- Readily available on commodity devices
- We want to leverage large quantity of computing resources
- We want to extensively leverage parallelization

Serial vs Parallel Programming

- Serial:
  - All tasks are processed in one processor
  - Instructions are executed one after another
- Parallel:
  - Tasks are processed by multiple processors
  - Instructions are executed simultaneously
- Observation:
  - Not all functions can be parallelized, but many can
  - The focus of cloud computing is not on how to make everything parallelized, but on how to facilitate the implementation of parallel processing for appropriate tasks.

Example

- We want to count the occurrence of word “apple”, “banana” and “orange” in 10,000 files.

- Serial programming:
  - Open each file, scan through the file to count the number and return the result

- The concept of serial programming is very simple and its implementation is also very simple.

Example (cont.)

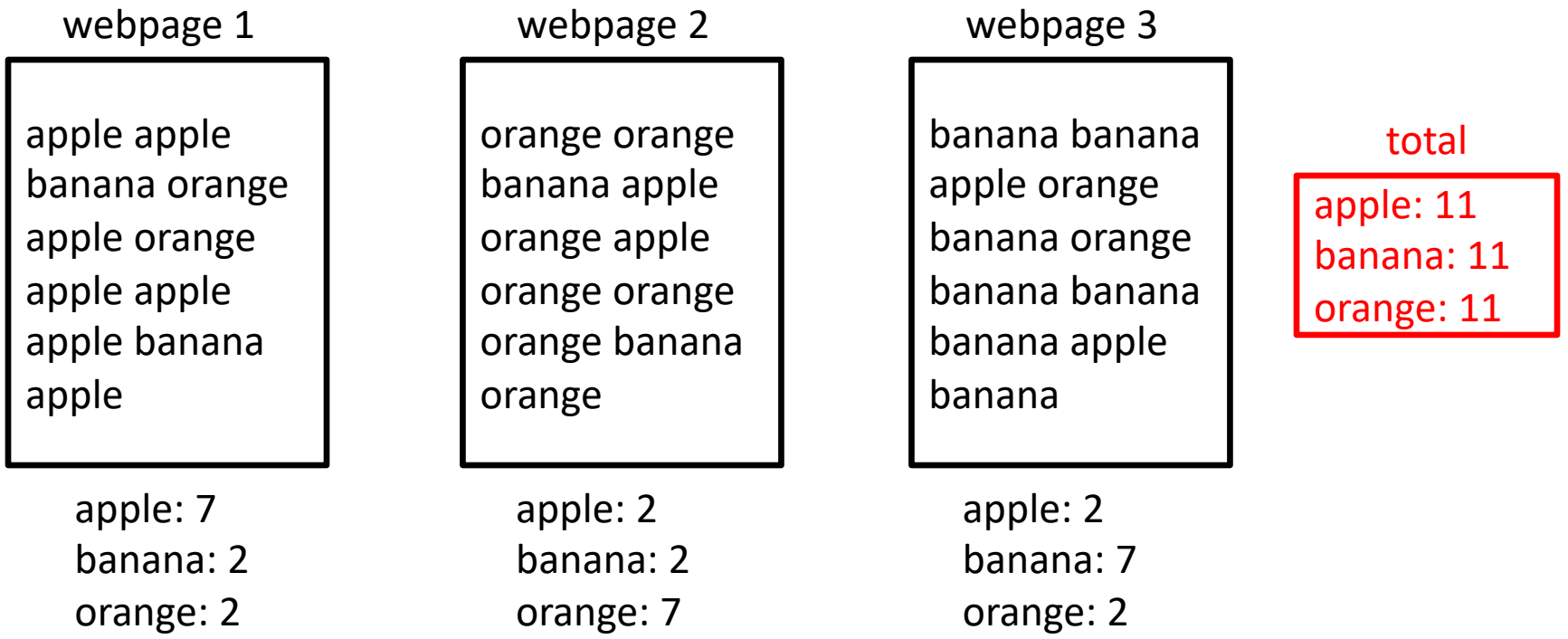
- We want to count the occurrence of the word “apple”, “banana” and “orange” in 10,000 files.

- Parallel programming with 100 computers
  - Give each computer a job to count 100 files
  - Each computer opens each file assigned to it
    - Scan through the file, count the number, and return the result

- The concept of parallel programming is very simple, but how about the implementation?

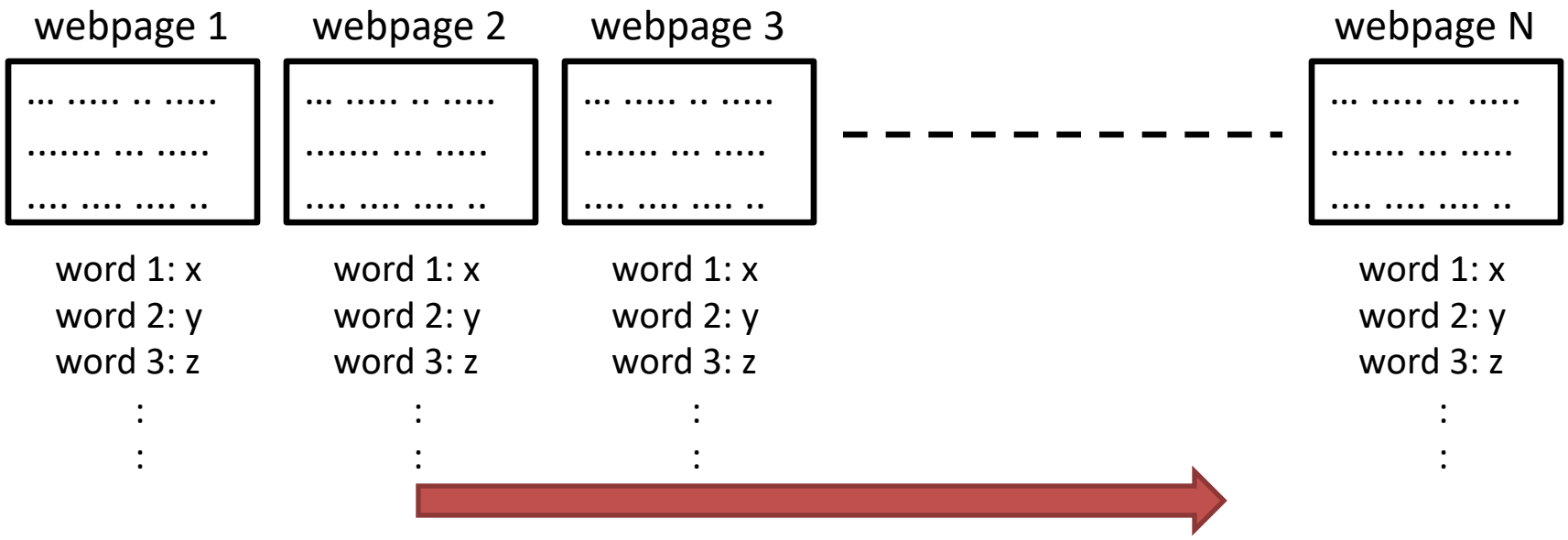
Example: Word Count

- Google needs to count word occurrence for indexing



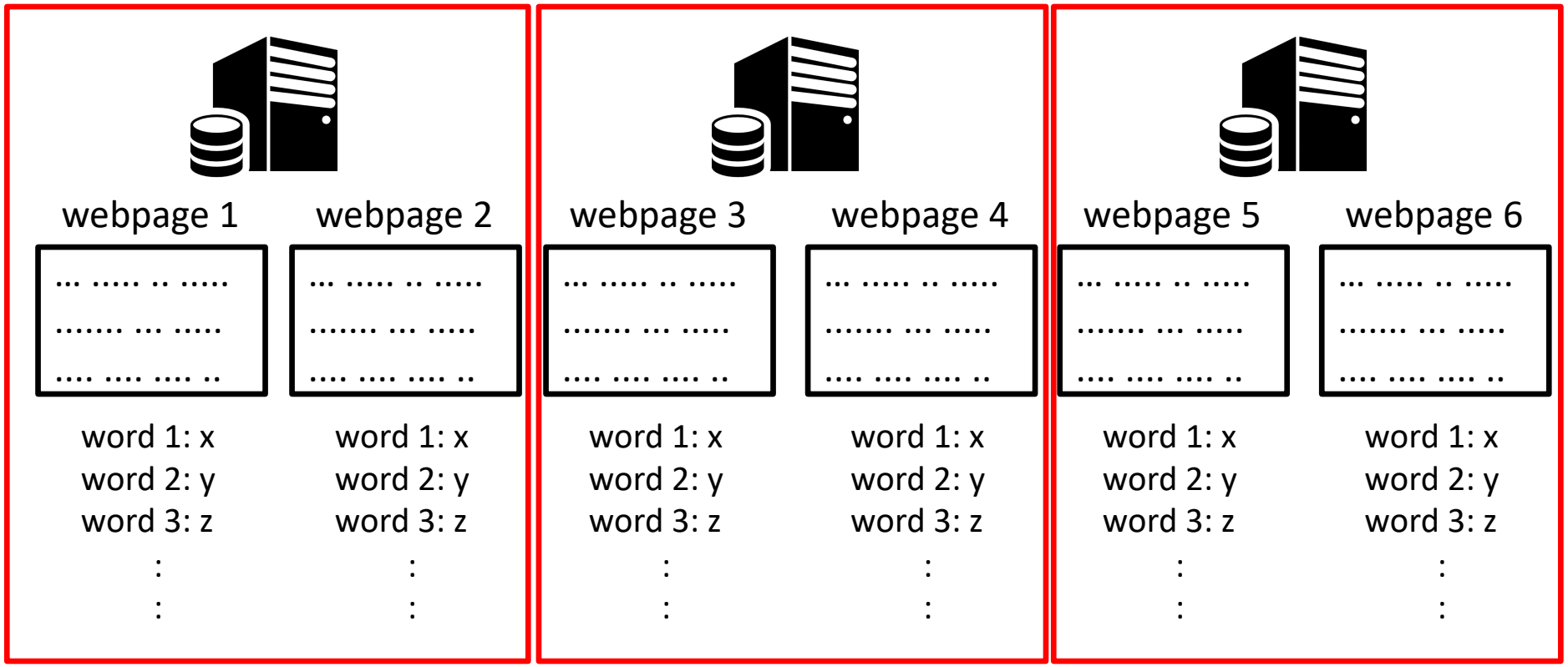
Example: Word Count (cont.)

- traditional sequential process is slow



Example: Word Count (cont.)

- Google has lots, lots, lots of machines
  - can each machine handle part of the data?



The Key Challenge

- For parallel programming, the core algorithm might be very simple.

- However, the overhead is considerable. In the previous example:
  - How to manage the computers?
  - What if the number of computers changes?
  - How to communicate among the computers?
  - How to distribute the files?
  - How to ensure reliability? What if a computer is down?

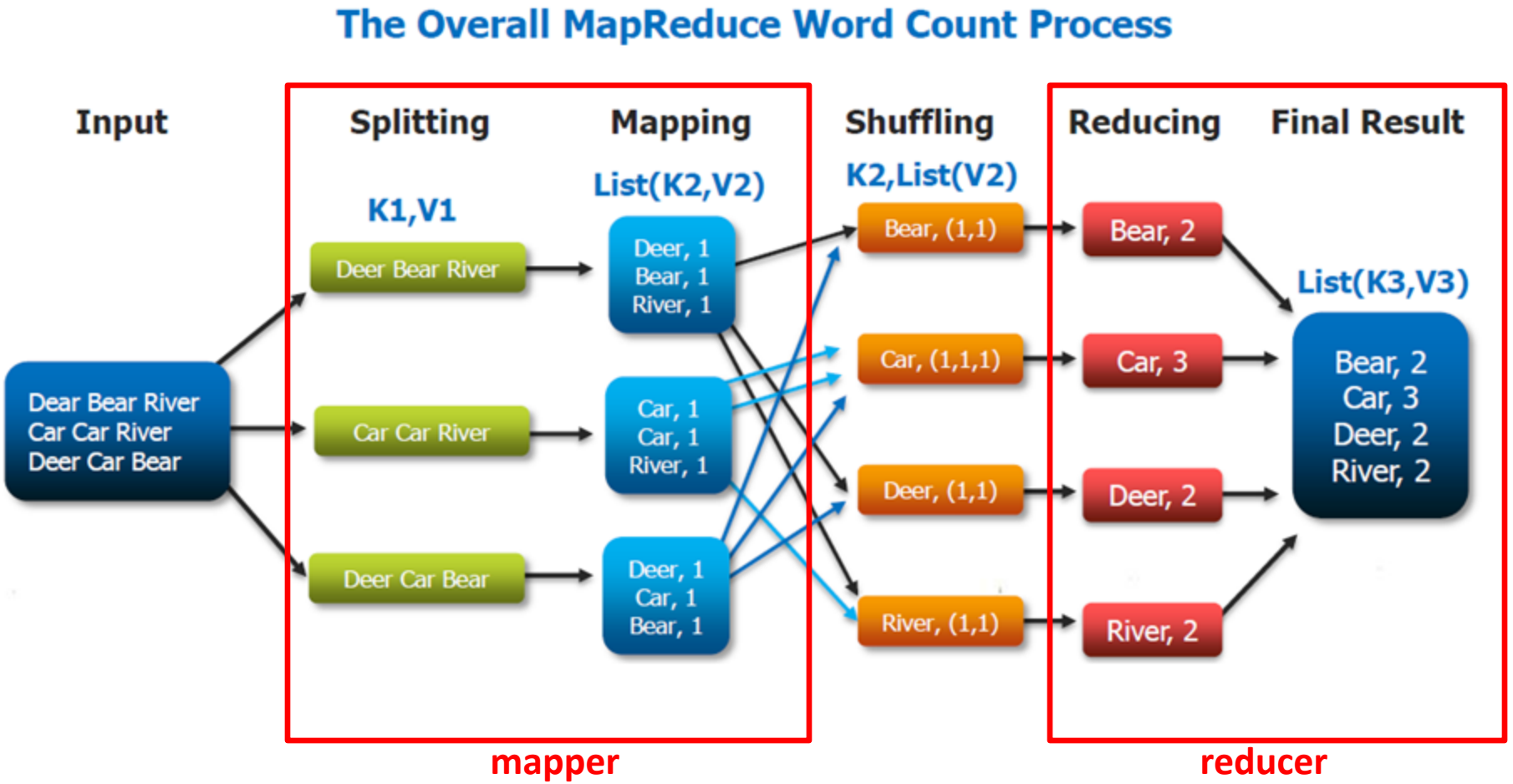
Key Idea of MapReduce

- We do not burden the programmers with the overhead. Instead, let them focus on the core algorithm designs.
- The overhead is common among most (if not all) of the programs. So why not create a platform/framework to take care of such processing?
- MapReduce is a programming model proposed by Google for processing large data sets.
- MapReduce libraries are provided to take care of such overhead jobs.

MapReduce

- Map
  - Assign jobs to many workers
  - Each worker takes the input and generates an intermediate result
- Reduce
  - The intermediate results are aggregated together to generate the final result

Example


The Overall MapReduce Word Count Process
Input
Splitting
Mapping
Shuffling
Reducing
Final Result
K1,V1
List(K2,V2)
K2,List(V2)
List(K3,V3)
Dear Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, (1,1)
Car, (1,1,1)
Deer, (1,1)
River, (1,1)
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2
mapper
reducer

(key, value) pairs

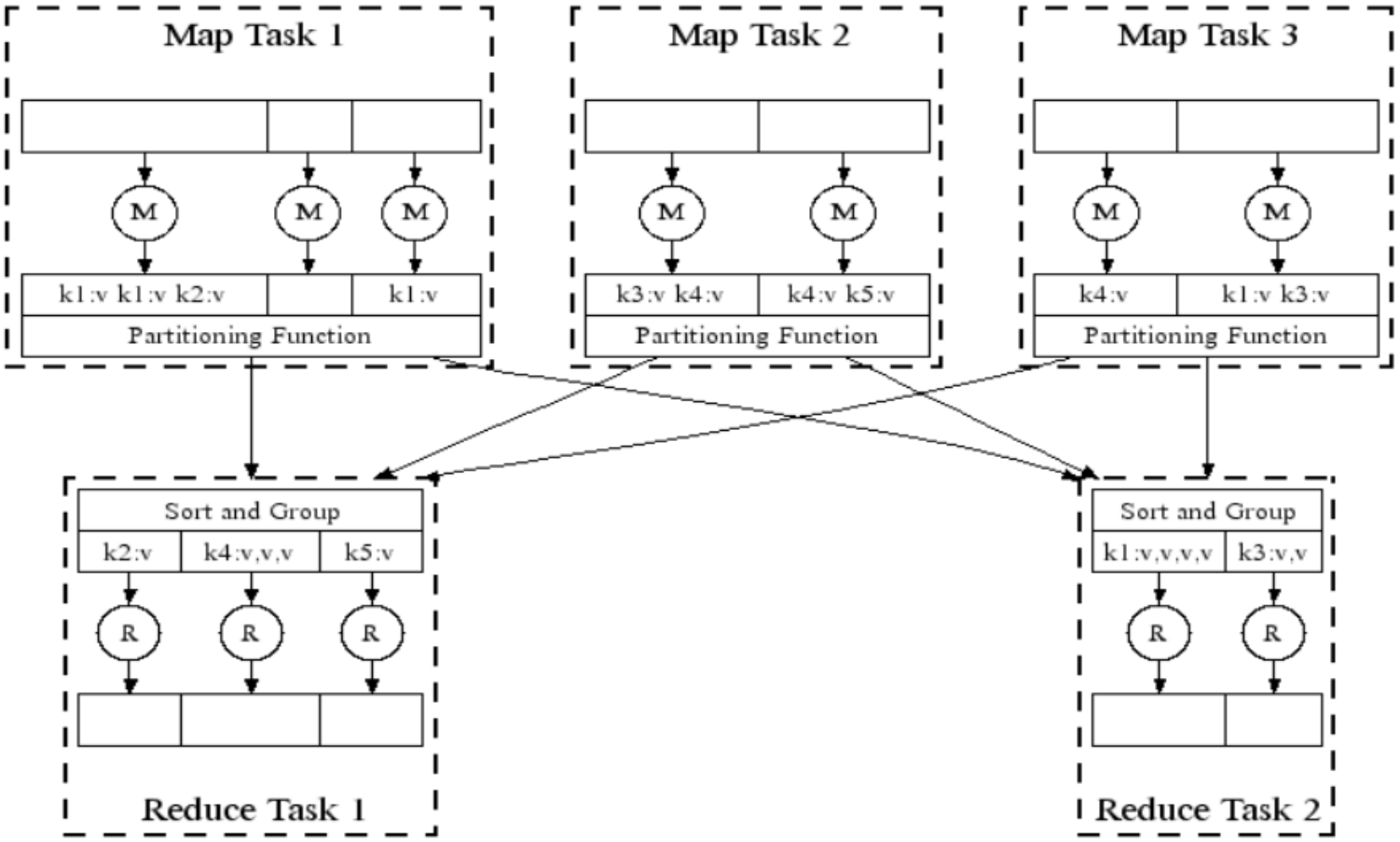
- Input to map()
  - (k1, v1)
- Output of map()
  - list(k2, v2)
- Input to reduce()
  - (k2, list(v2))
- Output of reduce()
  - list(v2)

- Example: counting the number of occurrence of each word in many documents

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Execution Overview



- Handles failures automatically, e.g., restarts tasks if a node fails; runs multiples copies of the same task to avoid a slow task slowing down the whole job.

What about storage?

- MapReduce is just a programming model
- How should we store the massive data and distribute the data to different machines? (Mappers and Reducers)

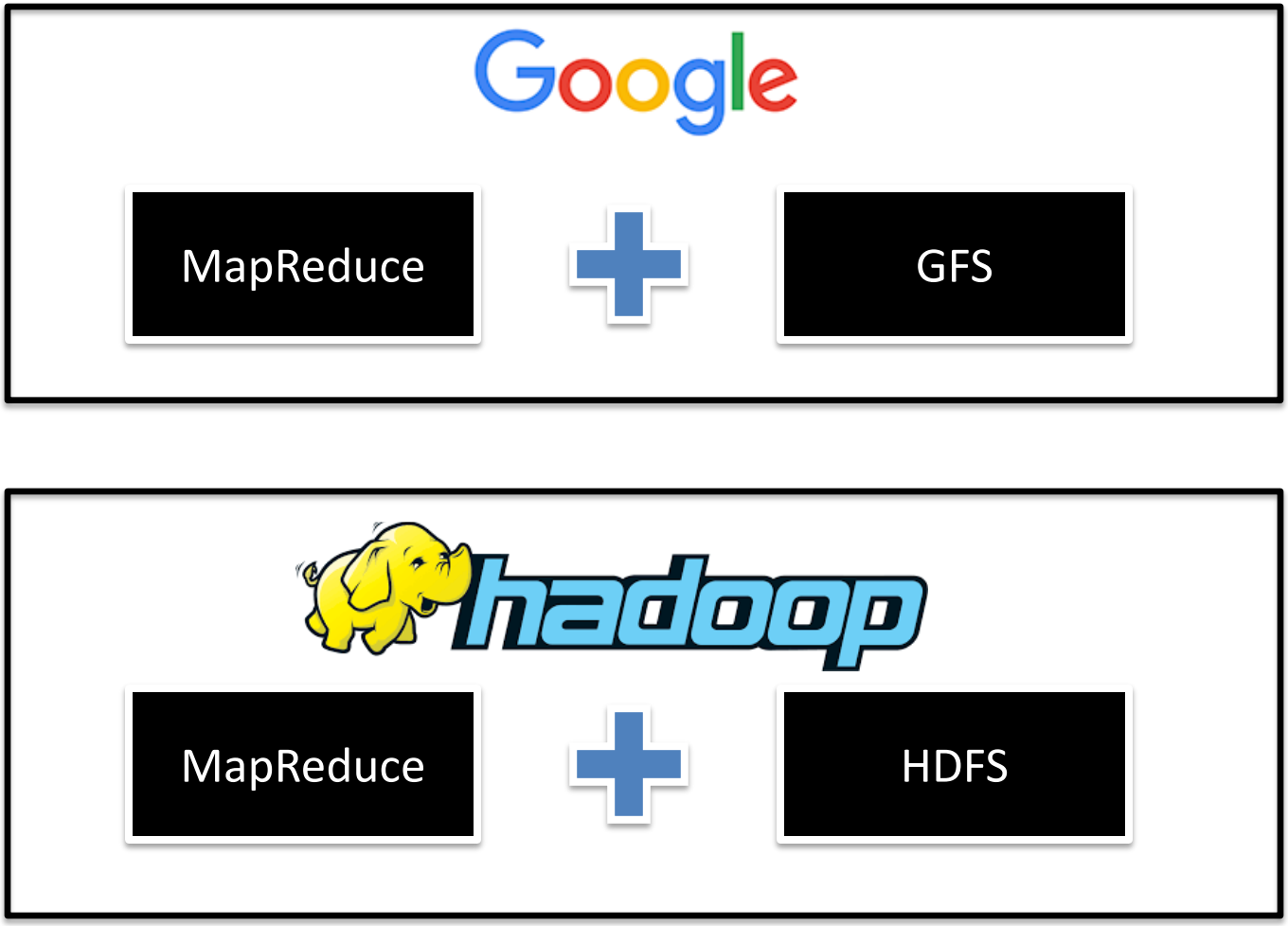
- Google's MapReduce operations run on a special file system called Google File System (GFS) that is highly optimized for this purpose.

- GFS is not open source.

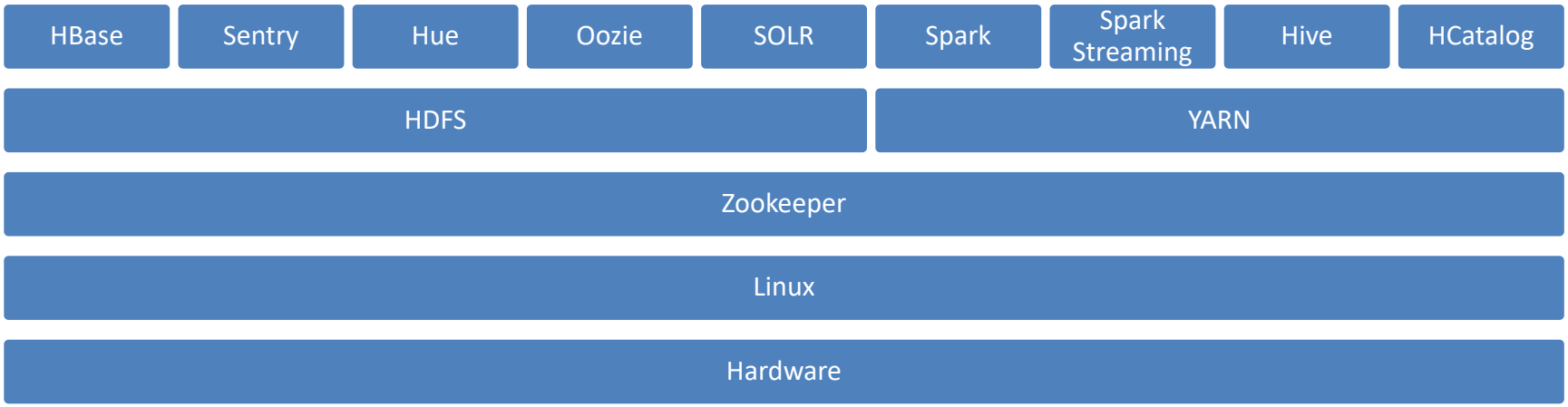
Hadoop

- Reverse engineered Google's MapReduce!
- In December 2004, Google Research published a paper on the MapReduce algorithm, which allows very large-scale computations to be trivially parallelized across large clusters of servers.
- Cutting and Mike Cafarella, realizing the importance of this paper, created the open-source Hadoop framework that allows applications based on the MapReduce paradigm to be run on large clusters of commodity hardware. Cutting was an employee of Yahoo!, where he led the Hadoop project.
- The software framework that supports HDFS, MapReduce and other related entities is called the project Hadoop or simply Hadoop, an open source Apache project.

Hadoop vs. Google



Hadoop Ecosystem*



- HDFS – Distributed file system that provides high-throughput access to application data [HADOOP].
- YARN – Framework for job scheduling and cluster resource management [HADOOP].
- Spark – Fast and general-purpose cluster computing system [SPARK].
- Spark Streaming - Extension of the Spark API for scalable, high-throughput, fault-tolerant stream processing of live data streams [SPARK_STREAM].
- Hive – Data warehouse infrastructure that provides data summarization and ad hoc querying [HADOOP].
- HCatalog – Service providing access to Hive Metastore data for various Hadoop components and other third party tools.
- Hue - Hue is a Web interface for analyzing data with Apache Hadoop [HUE].
- HBase - Scalable, distributed database that supports structured data storage for large tables [HADOOP].
- Oozie - Workflow scheduler system to manage Apache Hadoop jobs [OOZIE].
- Solr –Open source enterprise search platform built on Apache Lucene [SOLR].
- Zookeeper – High-performance coordination service for distributed applications [HADOOP].

*Depending on the distribution there can be different or more Hadoop related components.

Hadoop Distributed File System (HDFS)

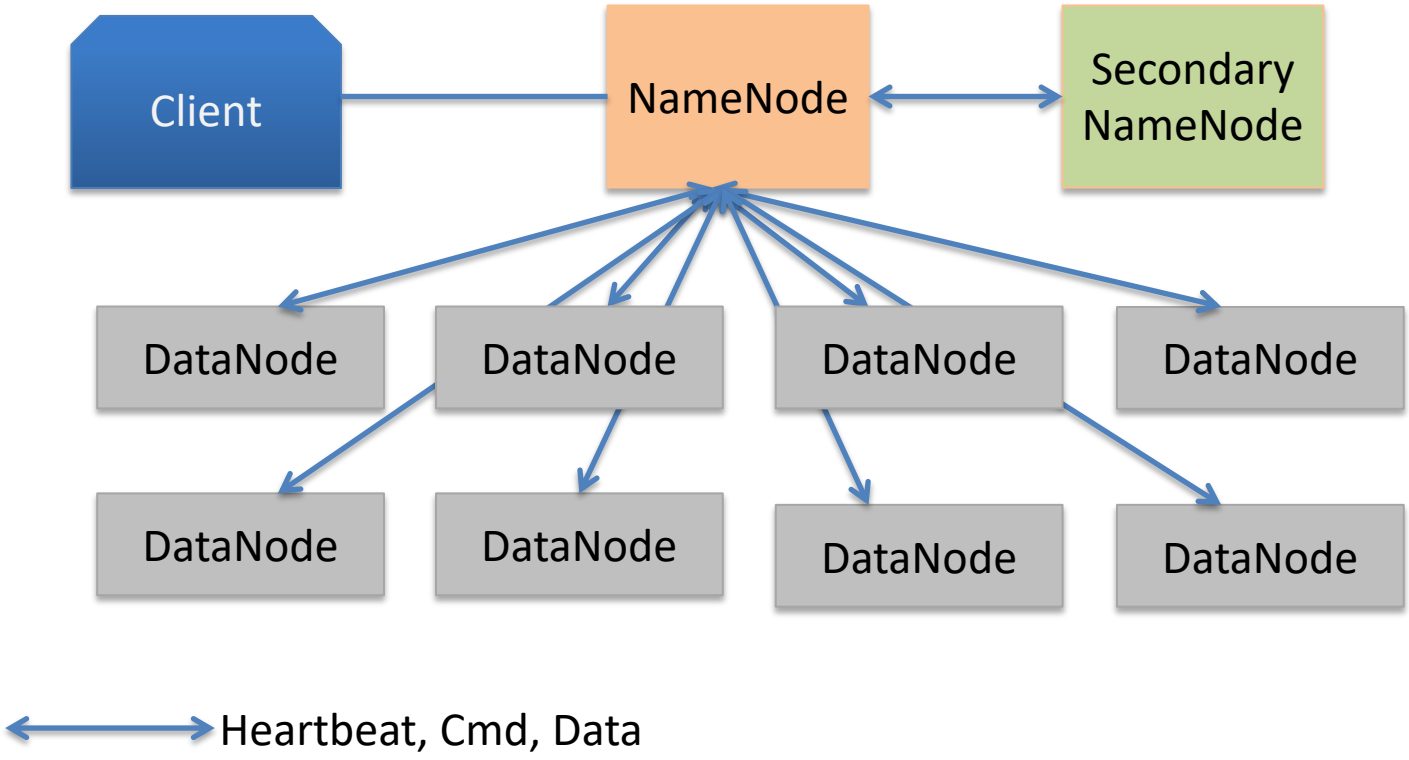
- Highly fault-tolerant
- High throughput
- Suitable for applications with large data sets
- Streaming access to file system data
- Can be built out of commodity hardware
- Master-Slave design
- Master Node
  - Single NameNode for managing metadata
- Slave Nodes
  - Multiple DataNodes for storing data
- Other
  - Secondary NameNode as a backup

Fault tolerance

- Failure is the norm rather than exception
- A HDFS instance may consist of thousands of server machines, each storing part of the file system's data.
- Since we have a huge number of components and each component has non-trivial probability of failure, it means that there is always some component that is non-functional.
- Hence, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

HDFS Architecture

NameNode: keeps the metadata, the name, location and directory
DataNode: provides storage for blocks of data

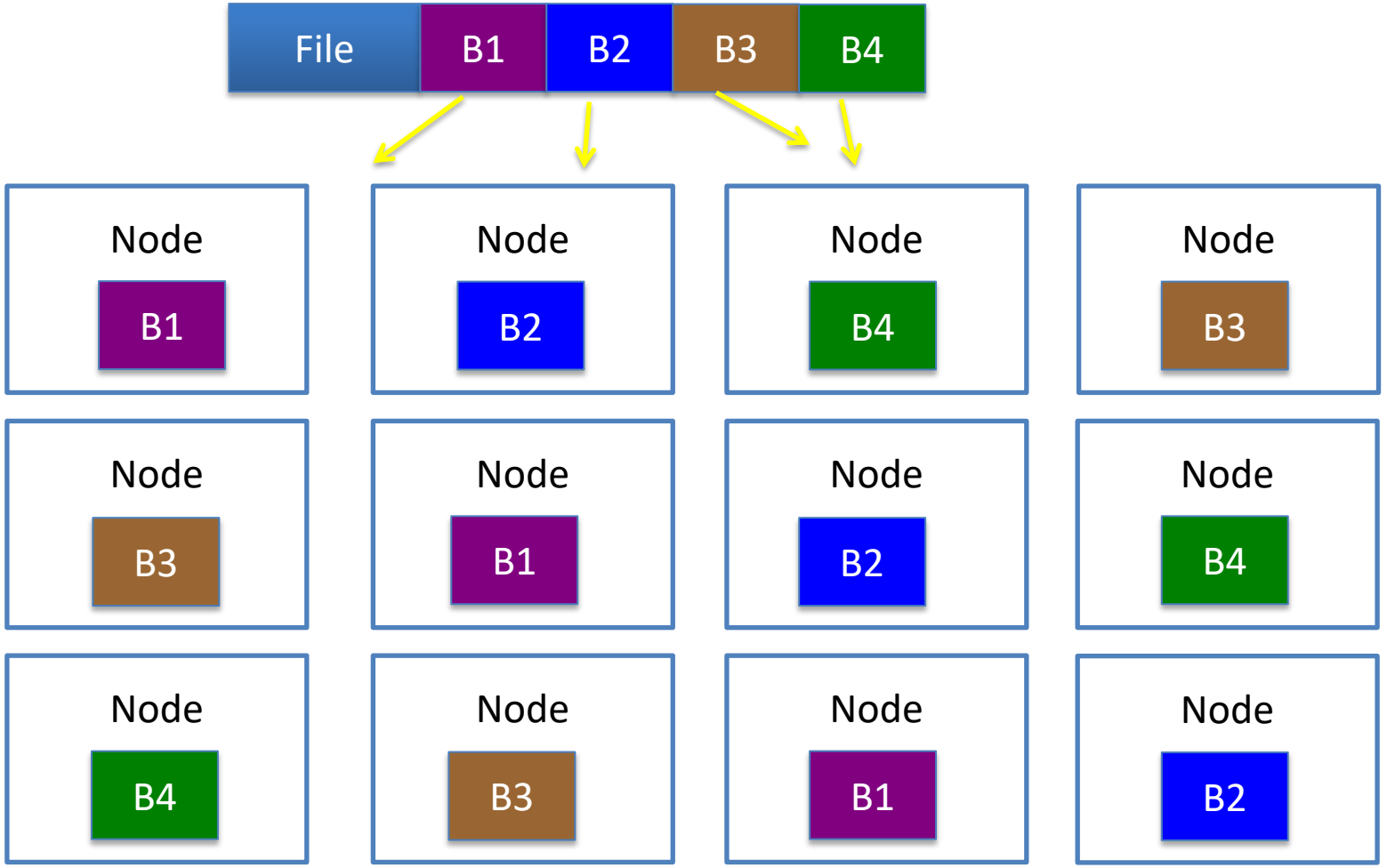


Namenode and Datanodes

- Master/slave architecture
- HDFS cluster consists of a single **Namenode**, a master server that manages the file system namespace and regulates access to files by clients.
- There are many **DataNodes** using commodity servers.
- The DataNodes manage storage attached to the nodes that they run on.
- HDFS exposes a file system namespace and allows user data to be stored in files.
- A file is split into one or more blocks and set of blocks are stored in DataNodes.
- DataNodes: serves read, write requests, performs block creation, deletion, and replication upon instruction from Namenode.

HDFS Block Replication

What happens if node(s) fail? Replication of Blocks for fault tolerance



HDFS Block Replications

- HDFS files are divided into blocks
  - Blocks: read/write units
  - Default size is 128MB
  - It makes HDFS good for storing larger files!
- HDFS blocks are replicated multiple times
  - One block stored at multiple location, also at different racks (usually 3 times)
  - This makes HDFS storage fault tolerant and faster to read

Data Replication

- HDFS is designed to store very large files across machines in a large cluster.
- Each file is a sequence of blocks.
- All blocks in the file except the last are of the same size.
- Blocks are replicated for fault tolerance.
- Block size and replicas are configurable per file.
- The Namenode receives a Heartbeat and a BlockReport from each DataNode in the cluster.
- BlockReport contains all the blocks on a Datanode.

References

- [HUE] - http://gethue.com/
- [HADOOP] - http://hadoop.apache.org/
- [OOZIE]- http://oozie.apache.org/
- [SOLR]- http://lucene.apache.org/solr/
- [SPARK] - http://spark.apache.org/docs/latest/
- [SPARK_STREAM] - http://spark.apache.org/docs/latest/streaming-programming-guide.html
- [HIVE_WIKI] - https://cwiki.apache.org/confluence/display/Hive/Design
- [BIGTABLE] - *Bigtable: A Distributed Storage System for Structured Data*, Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber - https://research.google.com/archive/bigtable.html
- [HBASE_ARCH] - http://hbase.apache.org/book.html#_architecture
- [OSVIRT_WIKI] - https://en.wikipedia.org/wiki/Operating-system-level_virtualization
- [SYS_PERF] - *Systems Performance: Enterprise and the Cloud*, Brendan Gregg, Prentice Hall, October 2013
- [LXC_WIKI] - https://en.wikipedia.org/wiki/LXC
- [CGRP_WIKI] - https://en.wikipedia.org/wiki/Cgroups
- [NMSP_CGRP] - http://www.netdevconf.org/1.1/proceedings/slides/rosen-namespaces-cgroups-lxc.pdf
- [SPARK] – http://spark.apache.org/docs/latest/
- [SPARK_STRM] - http://spark.apache.org/docs/latest/streaming-programming-guide.html
- [BORG_OMEGA_KUBE] - *Borg, Omega, and Kubernetes, Lessons learned from three container management systems over a decade,* ACMQUEUE Jan-Feb 2016, BRENDAN BURNS, BRIAN GRANT, DAVID OPPENHEIMER, ERIC BREWER, AND JOHN WILKES, GOOGLE INC.
- [KUBE_IO] – https://kubernetes.io
- [OPEN_STACK_CPT] - https://docs.openstack.org/admin-guide/common/get-started-conceptual-architecture.html
- [GOOG_MAP] - *MapReduce: Simplied Data Processing on Large Clusters,* Jeffrey Dean and Sanjay Ghemawat, Google Inc.*, https://research.google.com/archive/mapreduce-osdi04.pdf*
- [DOCKER_LNX_JNL] - *Docker: Lightweight Linux Containers for Consistent Development and Deployment*, http://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment?page=0,1
- [DOCKER_OVERVIEW] - https://docs.docker.com/engine/understanding-docker/
- [CONTAINER_SAAS] - *Towards a container-based architecture for multi-tenant SaaS applications,* Eddy Truyen, Dimitri Van Landuyt, Vincent Reniers, Ansar Rafique, Bert Lagaisse, Wouter Joosen, iMinds-DistriNet, KU Leuven, ARM 2016
- [KUBE_MULTI] - https://github.com/kubernetes/kube-deploy/tree/master/docker-multinode
- Some of the figures are from: Apache Hadoop and Spark: Introduction and Use Cases for Data Analysis presentation by Afzal Godil