University of East London

Pioneering Futures Since 1898

Strategic Hadoop Data Layer (HDFS) Design for Businesses

Dr Amin Karami

Associate Professor in AI&DS

a.karami@uel.ac.uk

www.aminkarami.com

CN7031 – Lecture 2

October 2025

Outline

- Proof of Concept (PoC), CapEx, and OpEx
- HDFS Cluster Design: Single-Node vs. Multi-Node
- Data Partitioning and Processing Time Calculations
- Comparison of Single-Node and Multi-Node Clusters
- Overview of MapReduce Programming
- Distributed Storage Systems: HDFS vs. Modern Solutions

Learning Outcomes

- Understand the role of HDFS in Big Data management
- Identify PoC setups and cost models (CapEx vs. OpEx)
- Compare single-node and multi-node cluster architectures
- Calculate partitioning and processing times effectively
- Learn the basics of MapReduce programming
- Compare HDFS with modern distributed storage solutions

Amin Karami

@AminKarami

Briefly speaking about my educational background, I completed my MSc D... >

aminkarami.com

Lecture 1&2: What is Big Data?

https://youtu.be/9nj3UPRu2yk

Tutorial 2: Working with HDFS

https://youtu.be/nnS3xjd6Wr8

PoC, CapEx, and OpEx Explained

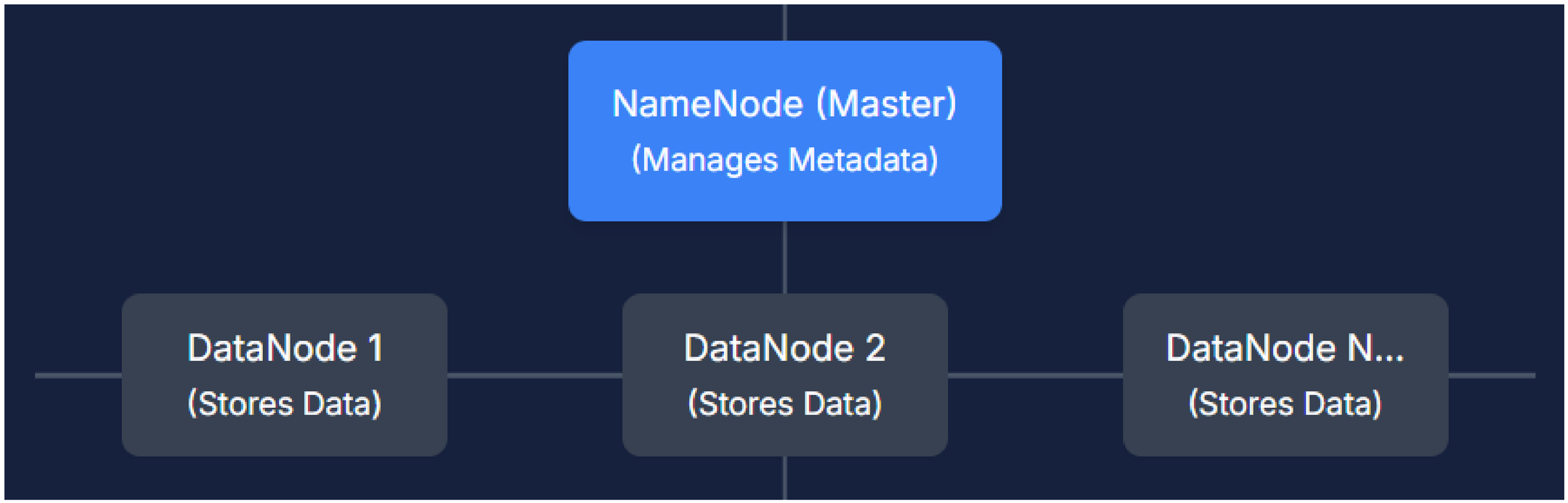
- Proof of Concept (PoC) refers to a small-scale, experimental setup or testing environment used to validate the feasibility, functionality, and potential of a particular system, technology, or approach before full-scale implementation.

- CapEx (Capital Expenditure) focuses on **what you buy** to build infrastructure, while OpEx (Operational Expenditure) focuses on **what you spend** to keep it running.

University of East London

HDFS: Harnessing the Data Deluge

This open-source framework architecture provides extreme scalability and fault tolerance by distributing data blocks and their replicas across many machines.



Single-node vs Multi-node HDFS Design

- Designing a cluster requires navigating the gap between:
  - A **simple single-node cluster**, suitable only for initial testing or a Proof of Concept (PoC) involving minimal resources , and
  - A **production-grade system** that can scale into the thousands of machines required for YB workloads.

Scaling Thresholds and Hardware Investment

Date Tier	Architecture	Hardware & Use Case Strategy
PoC/Dev (1GB-1TB)	Single-Node Cluster	Used for testing and rapid prototyping. Minimal resources (e.g., 4-6 CPU, 16-32GB RAM, 500GB-1TB disk). No fault tolerance required.
Mid-Scale (1TB-1PB)	Multi-Node Cluster	Standard production batch processing, requiring distribution of NameNode, DataNode, and resource management roles across multiple machines. Requires initial investment in 10GbE networking.
Hyperscale (1PB-1YB+)	Multi-Rack, De-coupled/Hybrid	Supports complex global workflows and massive storage footprints (e.g., Yahoo's 600+ petabytes or Netflix's global data processing). Management requires advanced tools like Apache Ambari and high-speed network interconnects (25GbE+).

Scaling Thresholds and Hardware Investment

Tier	Data Size	Nodes (Machines)	CPU (Cores)	RAM	Disk	Network	Fault Tolerance
PoC/Dev	50GB–1TB	1	4–6	16–32GB	500GB–2TB	1GbE	Not required
Mid-Scale	1TB–1PB	10–100	8–16 per node	64–128GB/node	8–12TB/node	10GbE	Replication (x3)
Hyperscale	1PB–50PB+	500–5000	16–64 per node	128–512GB/node	12–30TB/node	25GbE+	Replication (x3-5)

Task Type	Example	Throughput per Core
Simple	Filtering, lightweight aggregations, basic I/O	150–500 MB/s
Moderate	Joins, sorting, data transformations	100–300 MB/s
Complex	Machine learning (e.g., regression, clustering)	50–150 MB/s
Compute-Intensive	Deep learning, iterative model training	10–100 MB/s

Scenario 1: Data Processing for a Small E-commerce Platform

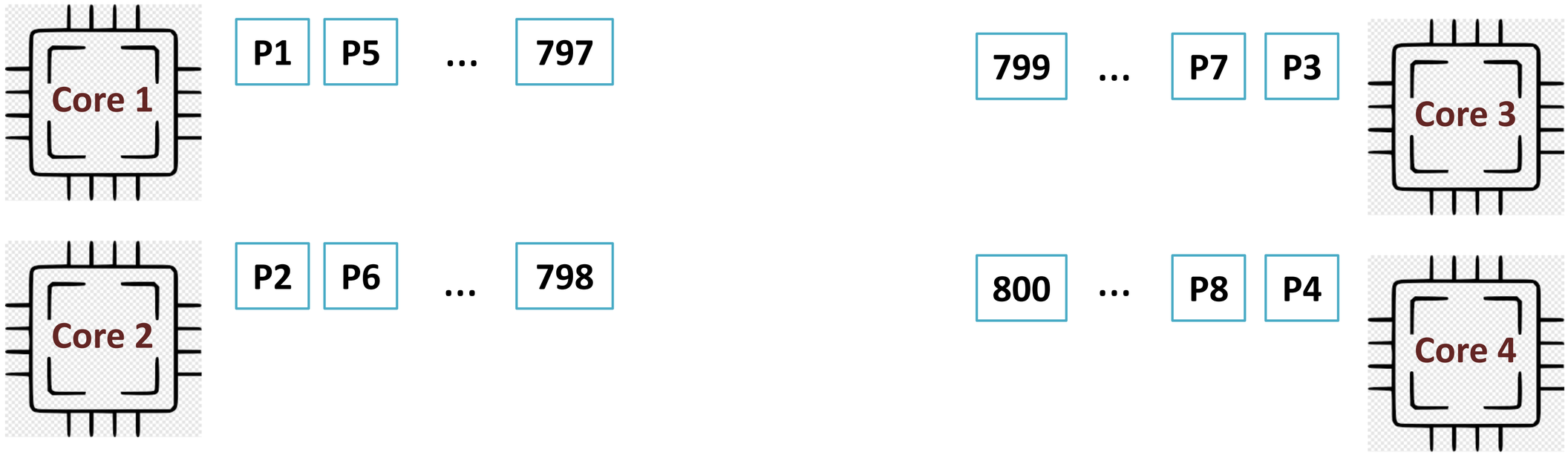
- **Dataset Size:** 100GB (includes browsing history, user activity logs, and Database).
- **Objective:** Test a recommendation engine with minimal resources.

Cluster (machine) Configuration:

Nodes: 1 (single-node cluster)
Cores: 4
RAM: 16GB
Disk: 1TB (HDD)
HDFS Block Size: 128MB (default)

Partitioning:

100GB÷128MB=800 partitions. All **800** partitions are managed by the single node



Throughput per Core:

Each core can process data at ~50MB/s

Processing Time for One Partition:

Partition Size (128MB) / Throughput per Core (50MB/s) = 2.56 seconds

---

University of East London

Time for 200 Partitions (800 partitions/ 4 cores) on 1 core:

Time per Core=200 partitions×2.56 seconds/partition
Time per Core=512 seconds (8 minutes 32 seconds)

Total time taken on 4 cores:

Since the 4 cores work in parallel, the total time for one epoch equals the time taken by one core to complete its 200 partitions = 512 seconds

What about serialization, not Parallelism?

512×4 = 2048 seconds (34 minutes 8 seconds).

Not possible, Due to the **16GB RAM limitation**, the **real-world serialization time** is likely to be much higher than **2048 seconds** because the system will not be able to hold all the data in memory at once.

Serialized Processing + DISK Read Time = ~57mins

$$T_{disk} = \frac{\text{Total size of the batch to be read (in MB or GB)}}{\text{Disk read speed (in MB/s or GB/s)}}$$

University of East London

Impact on Multi-Epoch Processing

For multi-epoch processing (e.g., in machine learning), the time for N epochs is proportional to the time for a single epoch:

If N=10 epochs:

Total Time=512×10=5120 seconds (1 hour 25 minutes 20 seconds)

Optimizations to Reduce Time

1- Increase Core count

2- Reduce Partition count (Increase the HDFS block size to 256MB or 512MB)

3- Upgrade Storage to SSD, offering significantly higher throughput (e.g., 250–500MB/s),

4- Use Compression: Parquet or ORC with Snappy

Scenario 2: Data Processing for a Small E-commerce Platform

- **Dataset Size:** ~100GB (includes browsing history, user activity logs, and Database).
- **Objective:** Test a recommendation engine with minimal resources.

Cluster Configuration:

Nodes: 5

Cores: 4 * 5 = 20

RAM: 16GB * 5 = 80GB

Disk: 1TB (HDD)

HDFS Block Size: 128MB (default)

Partitioning:
100GB÷128MB=800 partitions.

Partitions per Node: 800÷5 nodes = 160 partitions per node
Partitions per Core: Each node has 4 cores, so:
160 partitions÷4 cores per node=40 partitions per core

Throughput per Core: Each core can process data at ~50MB/s

Processing Time for One Partition:
Partition Size (128MB) / Throughput per Core (50MB/s) = 2.56 seconds

Time for 40 Partitions on 1 core:
Time per Core=40 partitions×2.56 seconds/partition = 102.4 seconds

Total time taken (1 epoch): All 4 cores on each node process their 40 partitions in parallel, and since there are 5 nodes, the entire workload is distributed across 20 cores. The time for one epoch is determined by the time taken by a single core to process its 40 partitions:
Total Time for 1 Epoch: Time per Core=102.4 seconds (1 minute 42.4 seconds)

Comparison: Scalability

This demonstrates the scalability and efficiency of distributing the workload in a multi-node cluster.

Metric	Single Node (1 node)	Multiple Node (5 nodes)
Total Partitions	800	800
Partitions per Node	800	160
Partition per Core	200	40
Processing Time per Partition	2.56 Sec	2.56 sec
Time per Core (Epoch)	512 Sec	102.4 sec
Total Time (Epoch)	512 sec	102.4 sec

Comparison: 10TB SQL workloads

Throughput per Core: 100MB/s (for better hardware and efficient processing) for SQL with **joins and sorting** (moderate complexity).

Metric	1 node	5 node	20 node	50 node
Total Cores	4	20	80	200
Total Partitions	81,920	81,920	81,920	81,920
Partitions per Node	81,920	16,384	4,096	1,638.4
Partition per Core	20,480	4,096	1,024	409.6
Time per Core (Epoch)	26,214.4 seconds (7h 17m)	5,242.88 seconds (1h 27m)	1,310.72 seconds (21m 51s)	524.3 seconds (8m 44.3s)
Total Time (Epoch)	26,214.4 seconds (7h 17m)	5,242.88 seconds (1h 27m)	1,310.72 seconds (21m 51s)	524.3 seconds (8m 44.3s)

Hadoop cluster: Installation and Deployment

- Setting up a Single Node Cluster:

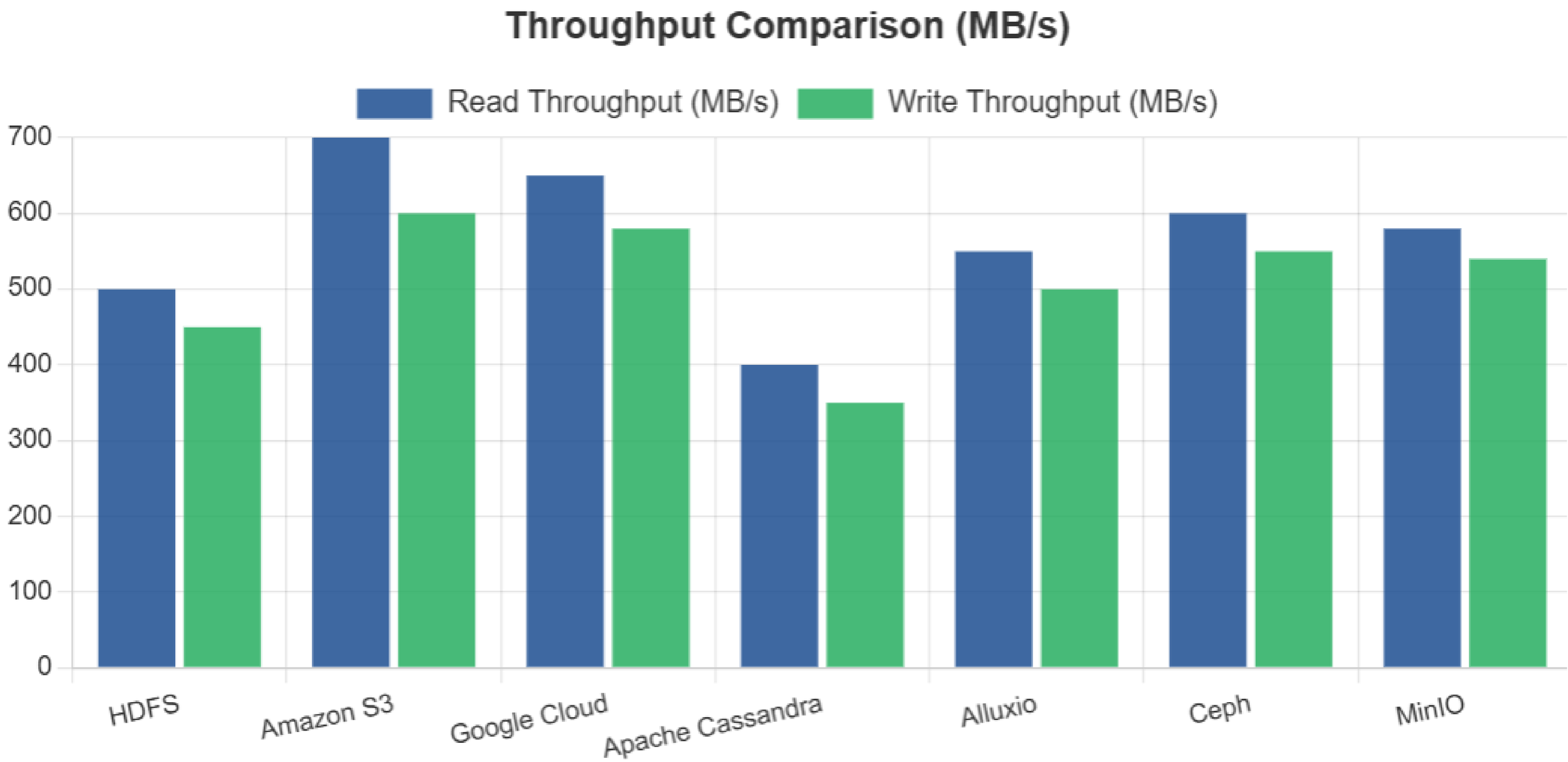
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

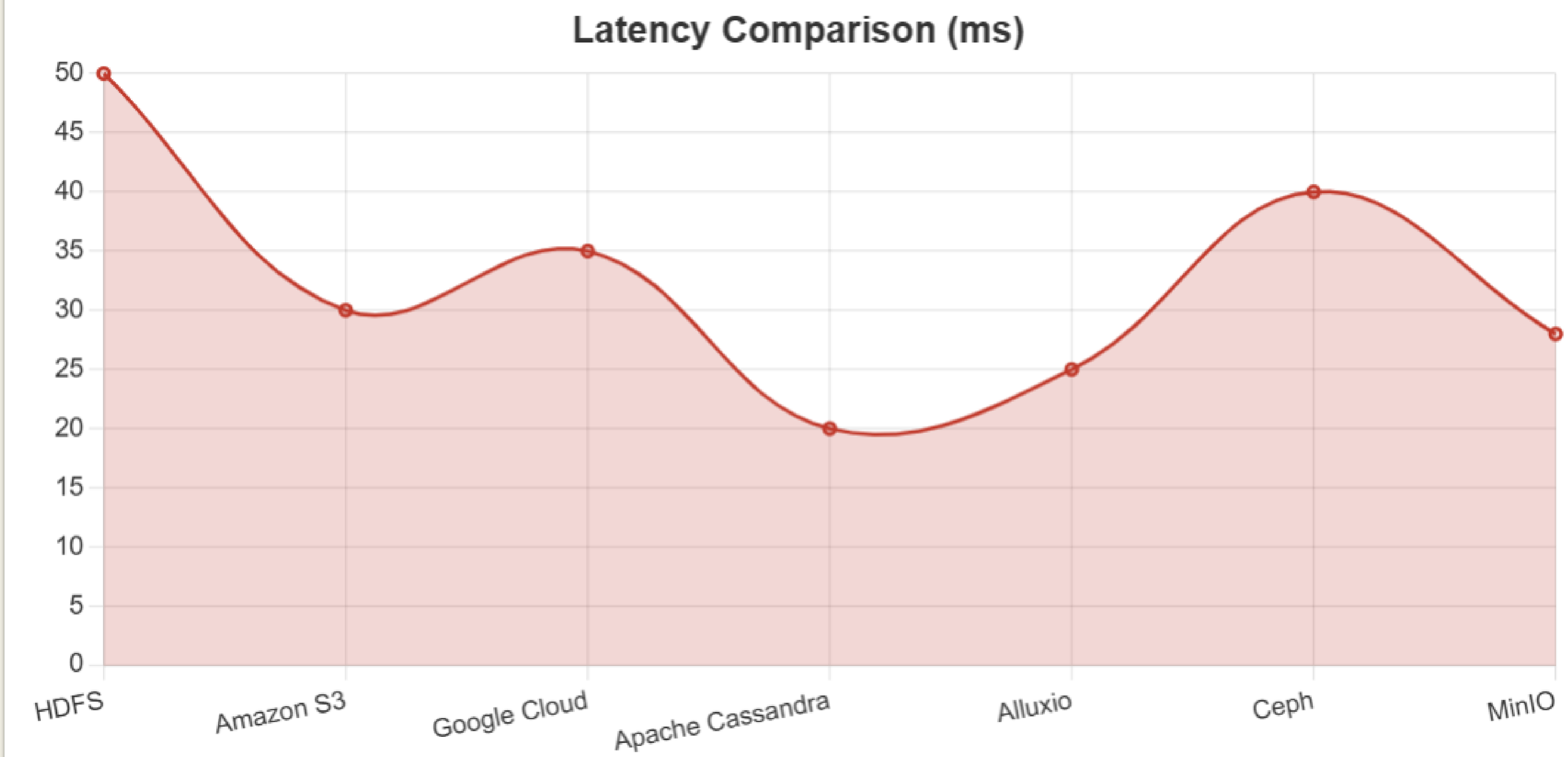
- Hadoop Cluster Setup:

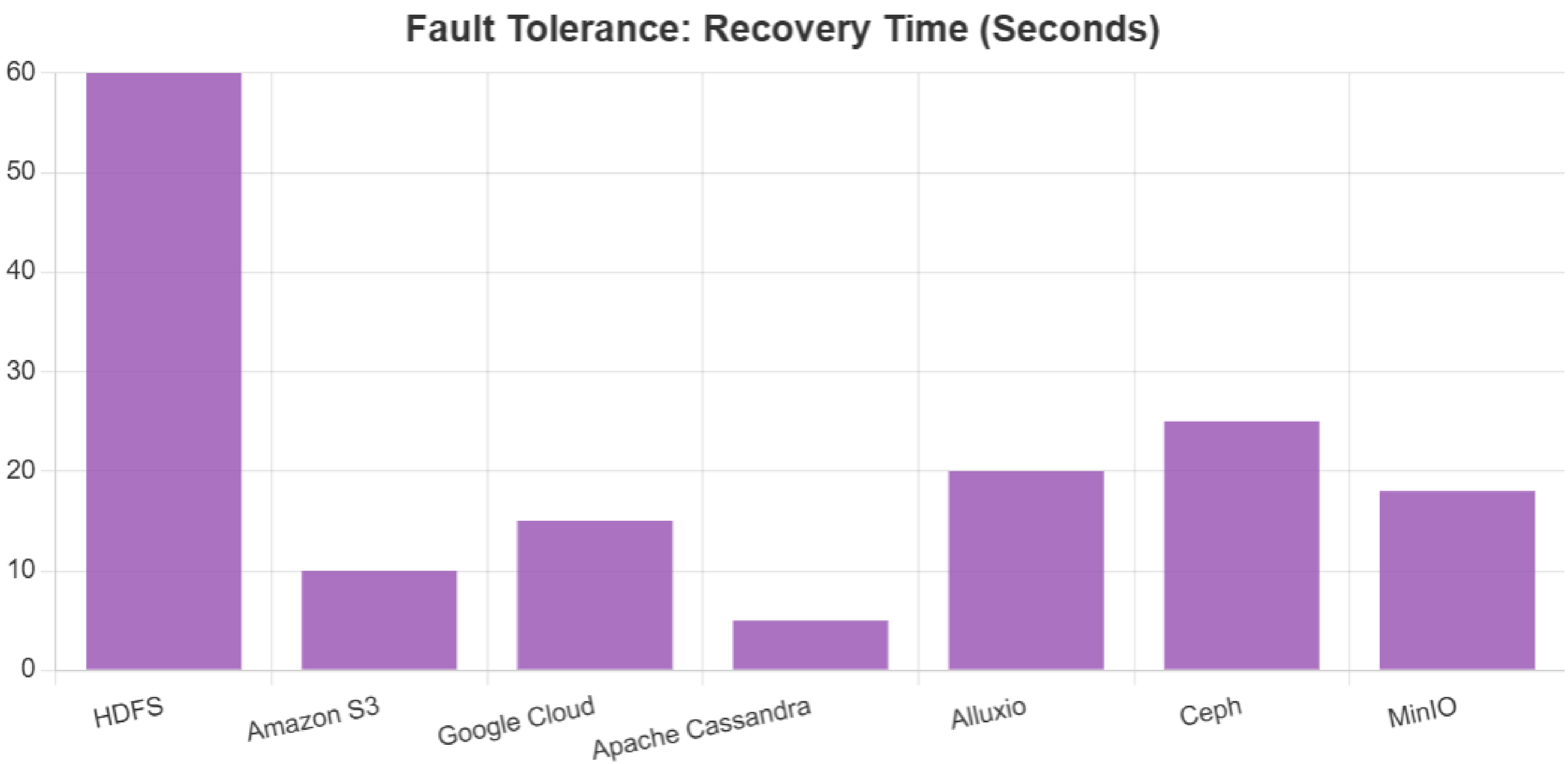
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html

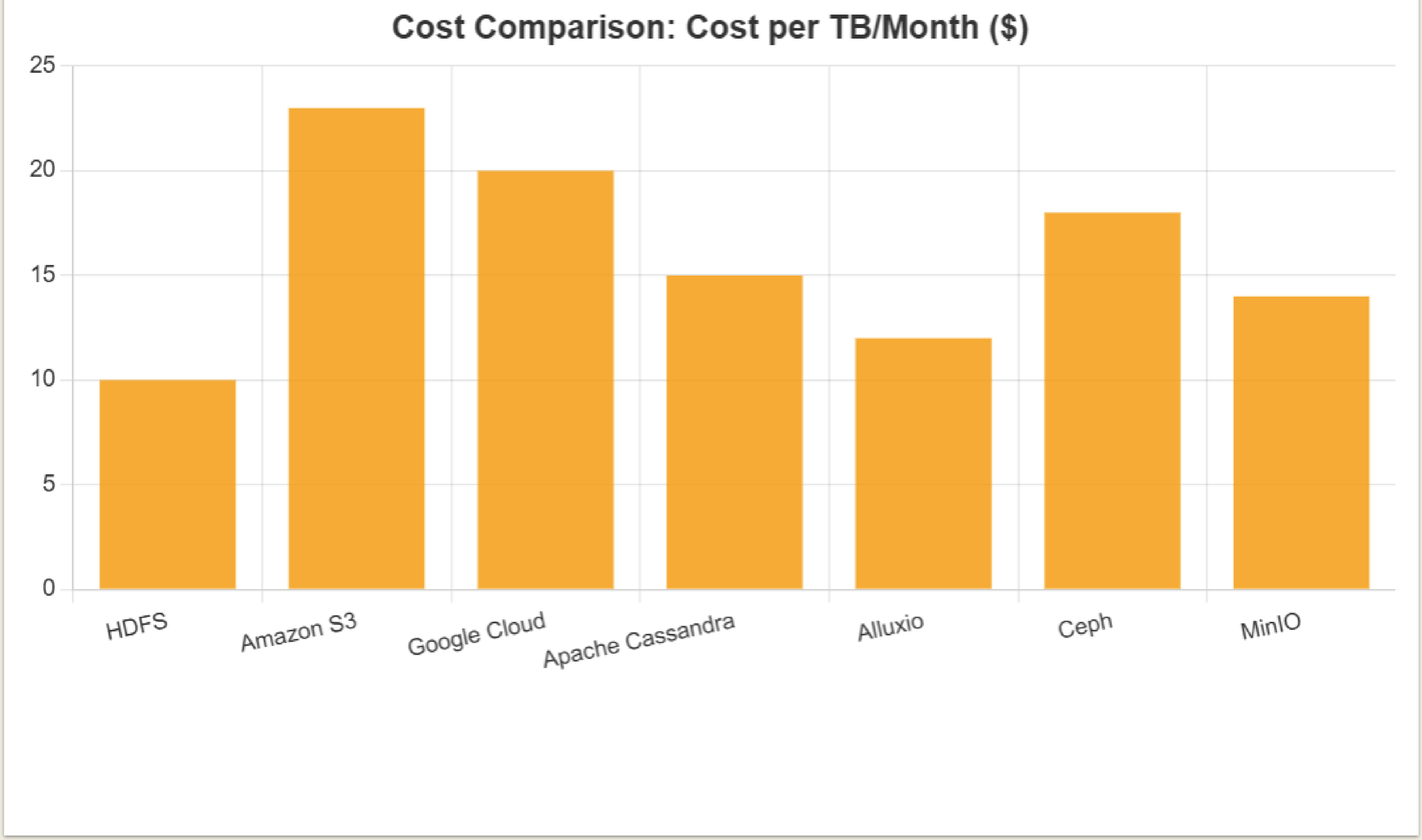
The Storage Evolution

- Many modern storage providers and distributed systems, such as **Amazon S3, Google Cloud Storage, and Ceph**, adopt similar principles of distributed storage, such as:
  - **Data Replication**: Storing multiple copies of data across different nodes to ensure fault tolerance.
  - **Scalability**: Distributing data across clusters to handle large-scale workloads.
  - **Fault Tolerance**: Recovering from hardware or network failures without losing data.
- However, standalone HDFS often appears weaker compared to these modern storage providers due to several critical factors:


Throughput Comparison (MB/s)
Read Throughput (MB/s)
Write Throughput (MB/s)
700
600
500
400
300
200
100
0
HDFS
Amazon S3
Google Cloud
Apache Cassandra
Alluxio
Ceph
MinIO


Latency Comparison (ms)
50
45
40
35
30
25
20
15
10
5
0
HDFS
Amazon S3
Google Cloud
Apache Cassandra
Alluxio
Ceph
MinIO


Fault Tolerance: Recovery Time (Seconds)
60
50
40
30
20
10
0
HDFS
Amazon S3
Google Cloud
Apache Cassandra
Alluxio
Ceph
MinIO


Cost Comparison: Cost per TB/Month ($)
25
20
15
10
5
0
HDFS
Amazon S3
Google Cloud
Apache Cassandra
Alluxio
Ceph
MinIO

Classification of Storage Systems

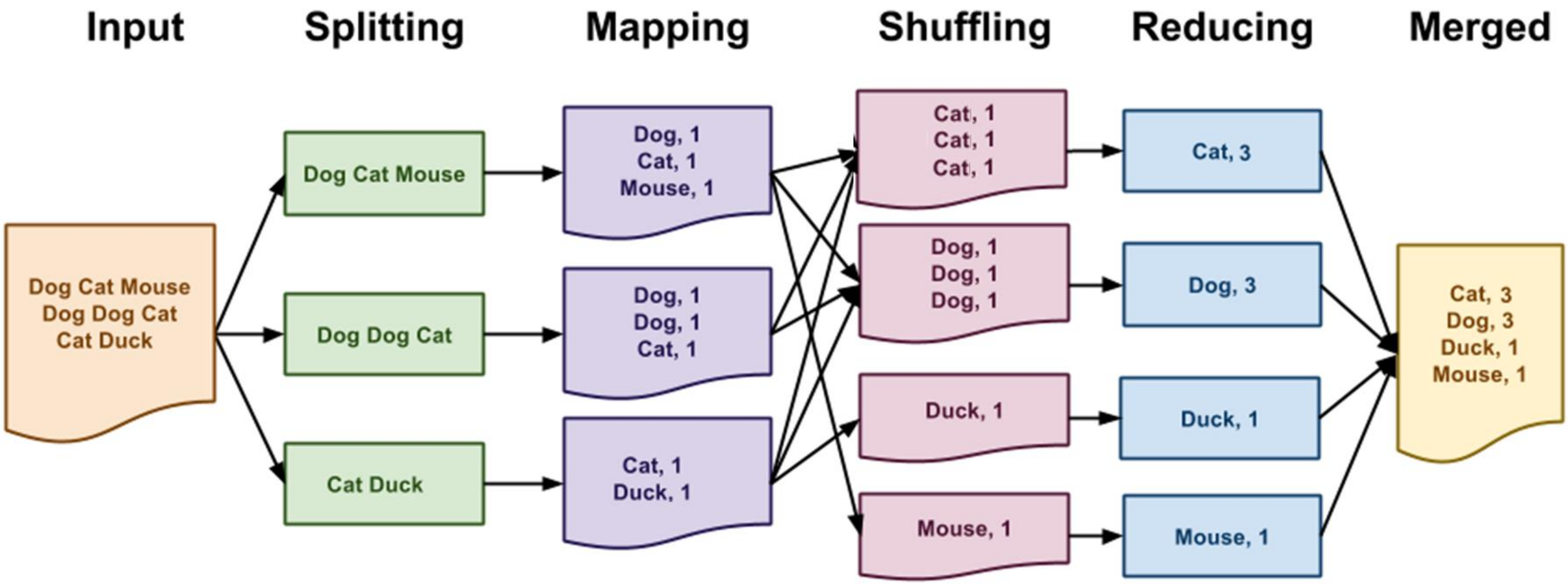
Storage System	Theoretical Foundation	Primary Usage	Type of Storage
HDFS	Distributed File System (Blocks)	Batch processing, Big Data Analytics	File-based
Amazon S3	Object Storage	Cloud-native storage, Backup, Archiving	Object-based
Google Cloud Storage	Object Storage	Cloud-native storage, AI/ML pipelines	Object-based
Apache Cassandra	NoSQL Database with Distributed Storage	Real-time analytics, Low-latency applications	Key-value and Columnar
Alluxio	Distributed Virtual File System	Data orchestration across hybrid cloud	File/Memory-based
Ceph	Distributed Object Storage	Hybrid cloud storage, High availability	Object-based
MinIO	S3-Compatible Object Storage	Private cloud storage, Kubernetes-native	Object-based

HDFS Weakness vs Modern Storage Systems

#	Factor	Description	HDFS	Modern Storage Systems
1	Cloud-Native Features	Lack of support for global availability, dynamic scaling, and managed infrastructure.	Limited	Fully integrated with cloud-native services like auto-scaling and multi-region replication.
2	Latency	Higher latency for small, frequent read/write operations due to block-based storage design.	High	Optimized for low latency using caching and object-based storage.
3	Maintenance Overhead	Manual configuration and cluster management for scaling, fault tolerance, and updates.	High	Minimal; fully managed by service providers.
4	Integration with Modern Workloads	Limited support for real-time processing, object storage APIs, and hybrid applications.	Limited	Fully integrates with AI/ML pipelines, real-time analytics, and hybrid cloud.
5	Fault Recovery Time	Slower block replication and recovery during node failures.	Slow	Fast recovery using multi-zone replication or erasure coding.
6	Cost Efficiency	High upfront costs for hardware, maintenance, and skilled personnel.	High	Pay-as-you-go model with minimal upfront investment.
7	Innovation Stagnation	Limited evolution beyond batch processing for big data workloads.	Slow	Continuous innovation, integrating with serverless architectures and modern workloads.
8	Security and Compliance	Lacks advanced security features and certifications for enterprise-grade usage.	Basic	Advanced; includes encryption, fine-grained access control, and global compliance certifications.

MapReduce Overview

- Write a code that utilizes MapReduce in Hadoop to perform a word count on a large text dataset.



WordCount: Mapper phase

- **Create <Key-Value>**

```
mapper.py
#!/usr/bin/python
import sys
#Word Count Example
# input comes from standard input STDIN
for line in sys.stdin:
line = line.strip() #remove leading and trailing whitespaces
words = line.split() #split the line into words and returns as a list
for word in words:
#write the results to standard output STDOUT
print'%s    %s' % (word,1) #Emit the word
```

WordCount: Reducer phase

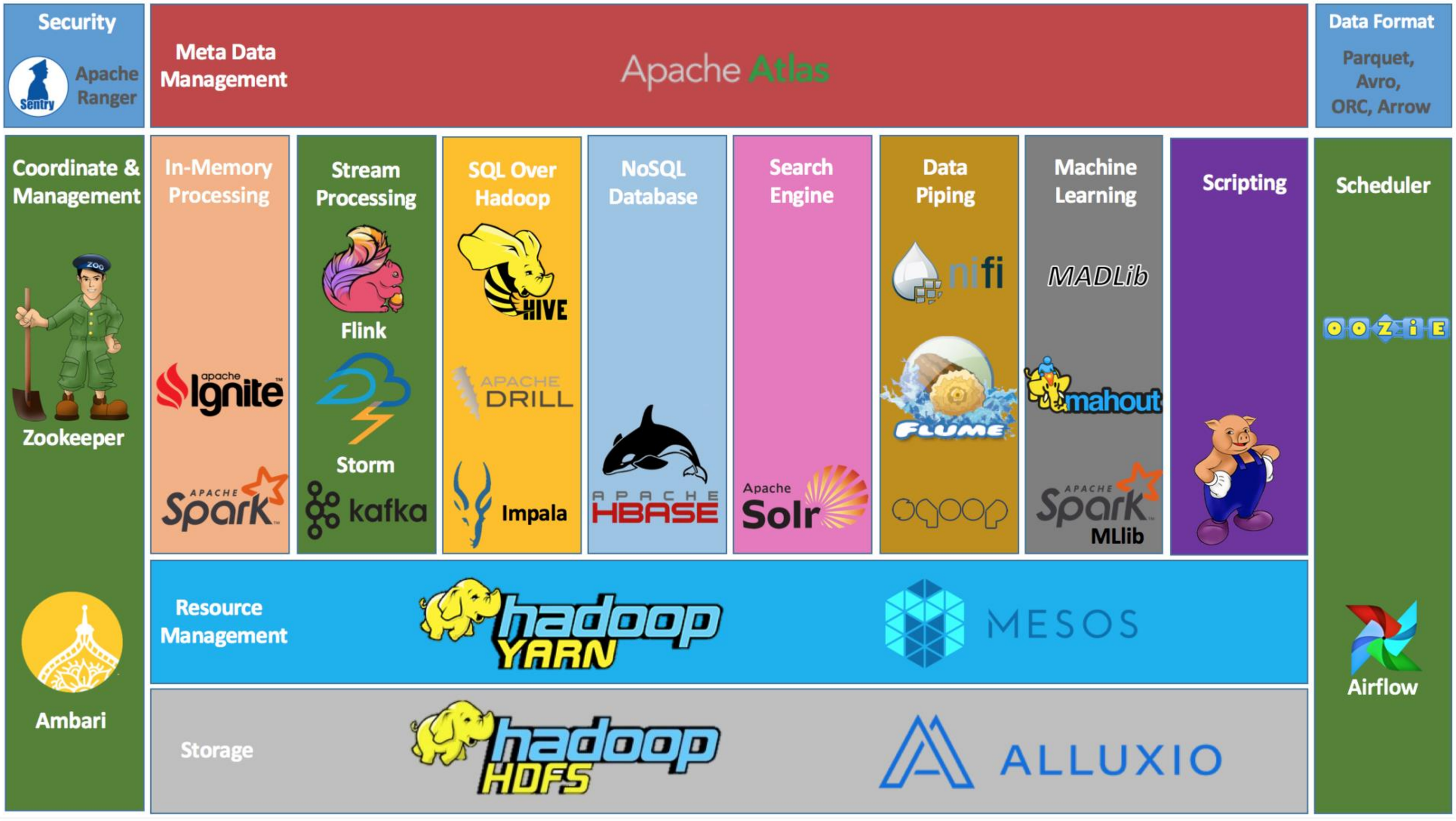
```
#!/usr/bin/python
import sys
from operator import itemgetter
# using a dictionary to map words to their counts
current_word = None
current_count = 0
word = None
# input comes from STDIN
for line in sys.stdin:
line = line.strip()
word,count = line.split('    ',1)
try:
count = int(count)
except ValueError:
continue
if current_word == word:
current_count += count
else:
if current_word:
print '%s    %s' % (current_word, current_count)
current_count = count
current_word = word
if current_word == word:
print '%s    %s' % (current_word,current_count)
```

Aggregation on Keys

MapReduce programming Challenges

- Convert the problem to Key-Value format
- How to convert SQL query, if-then-else, Loops and other logical blocks and coding to Key-Value format
- **Solution**: Hadoop ecosystem

Hadoop ecosystem



Cloudera

- Cloudera provides a distribution of Hadoop called **Cloudera Distribution of Hadoop (CDH),** which includes a set of services and products that you can use to deploy, manage, and monitor your Hadoop cluster, also offers professional support and training services.

https://www.cloudera.com/downloads/cdh.html

CLOUDERA

Apache Ambari

- Apache Ambari is a software that is open-source and used for **managing and monitoring Hadoop clusters**. It offers a user-friendly web interface for the management and monitoring of Hadoop services. Additionally, it automates the installation and configuration of Hadoop services across the cluster.

Apache Ambari

Summary

- HDFS ensures scalability and fault tolerance.
- PoC is for testing; production needs multi-node clusters.
- Partitioning helps estimate processing times.
- Optimizations include more cores, SSDs, and compression.
- Multi-node clusters scale better than single-node setups.
- MapReduce enables distributed data processing.
- Modern storage systems surpass standalone HDFS.

University of East London