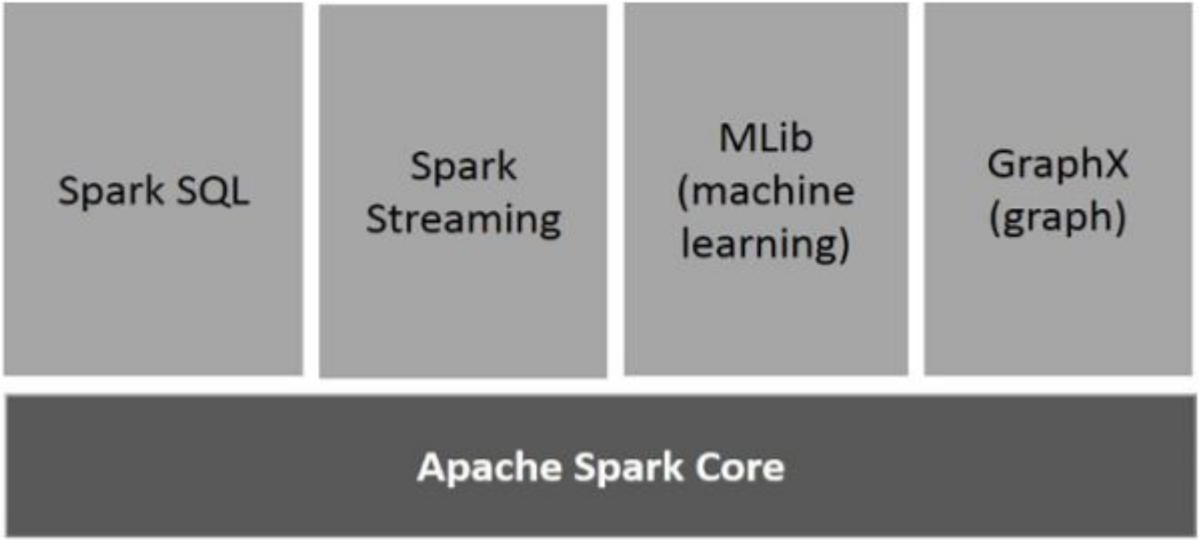
CS202 Project
Performance evaluation and comparison between distributed file systems

Group 13
Chen-Yang Yu 862052273
Po-Cheng Kuo 862029279

Spark

- Spark was introduced for speeding up the Hadoop computational computing software process.
- Resilient Distributed Datasets(RDD)
    - in-memory cluster computing

Spark



Spark SQL

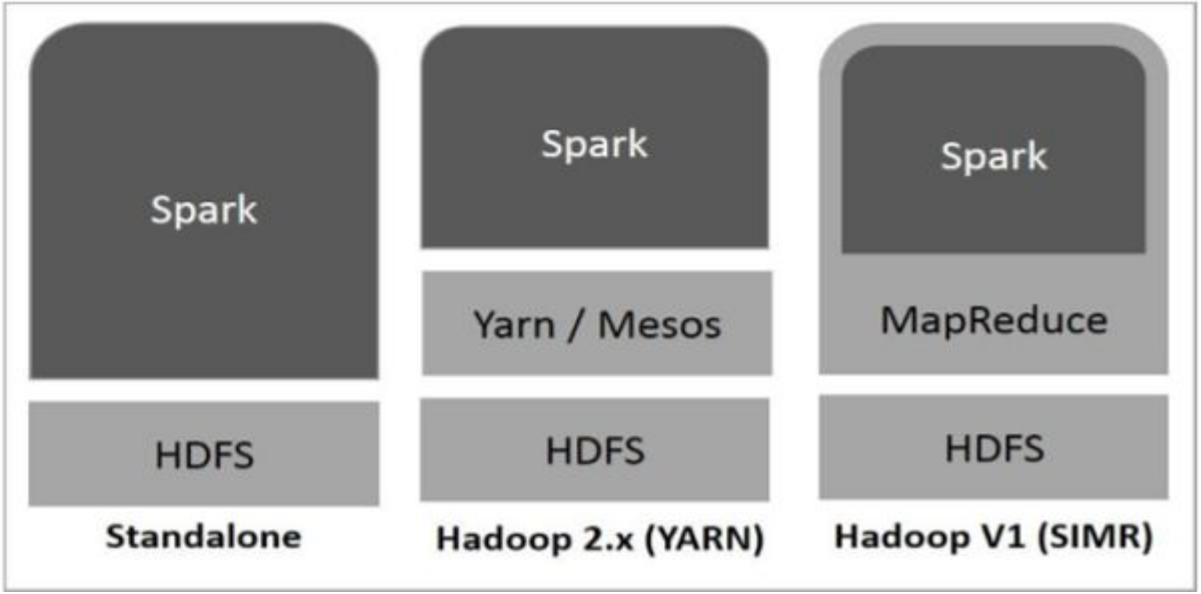
Spark Streaming

MLib (machine learning)

GraphX (graph)

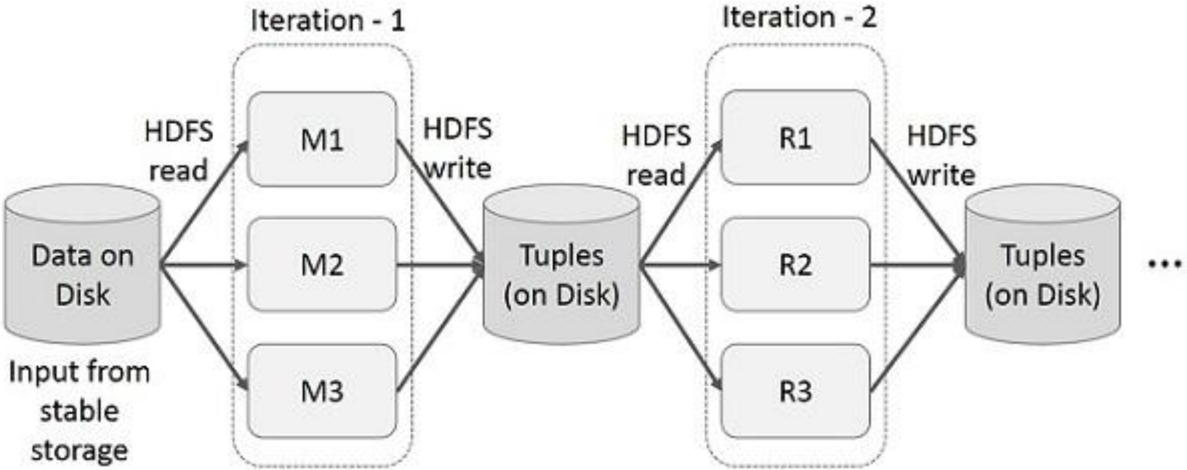
Apache Spark Core

Spark


Spark
HDFS
Standalone
Spark
Yarn / Mesos
HDFS
Hadoop 2.x (YARN)
Spark
MapReduce
HDFS
Hadoop V1 (SIMR)

Spark

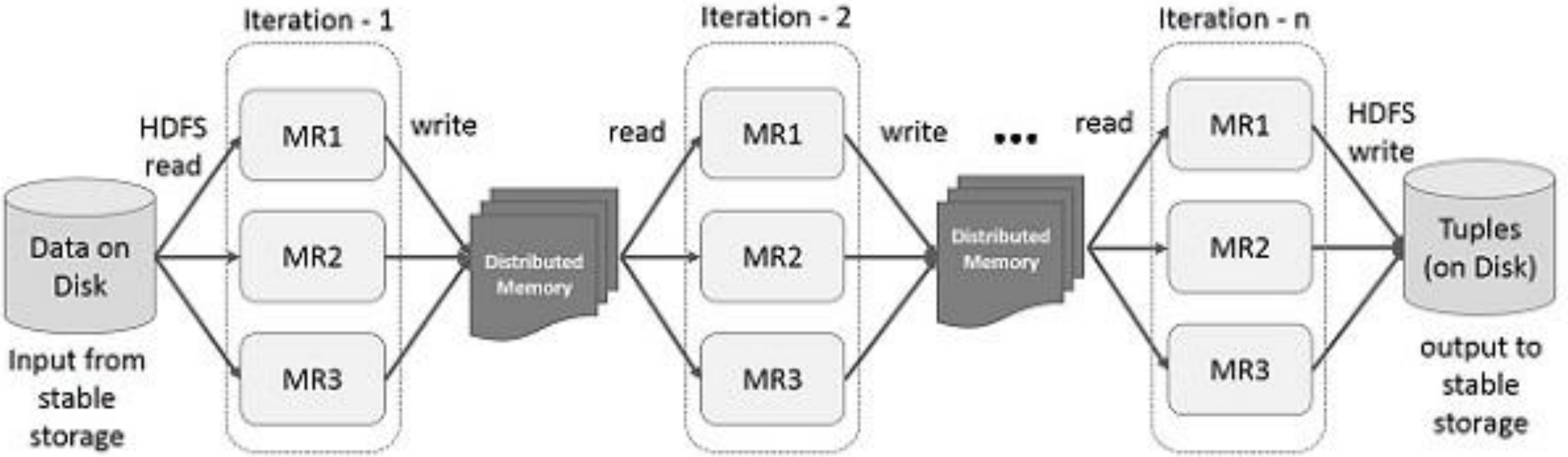
Hadoop mapreduce process



Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

Spark

iterative operations on Spark RDD

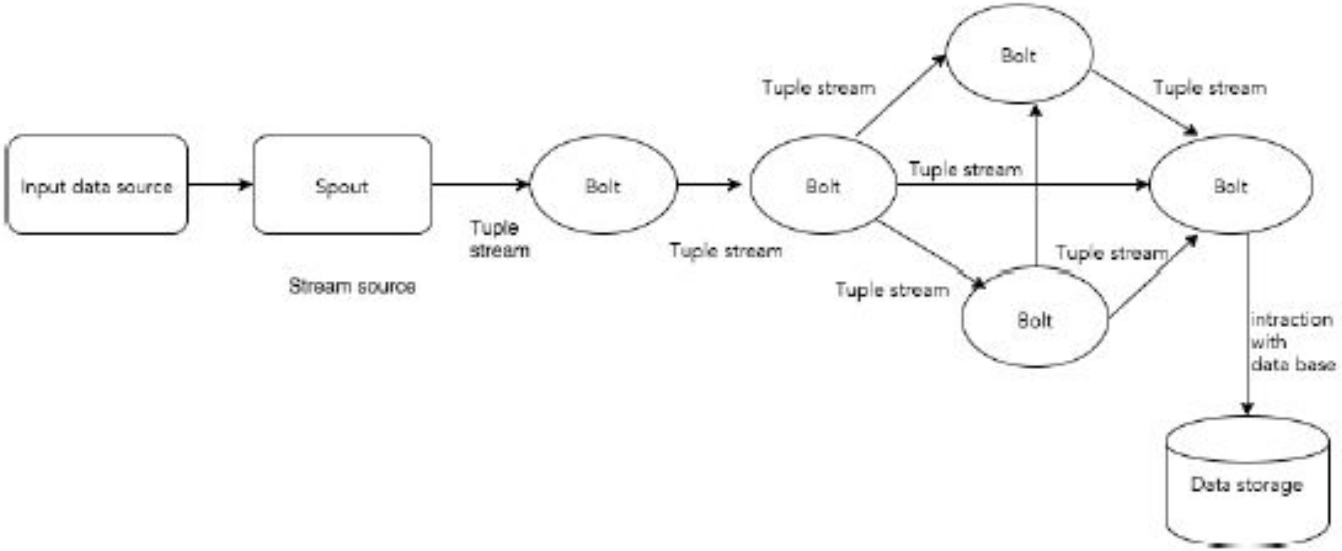


store intermediate results in a distributed memory instead of disk and make the system faster

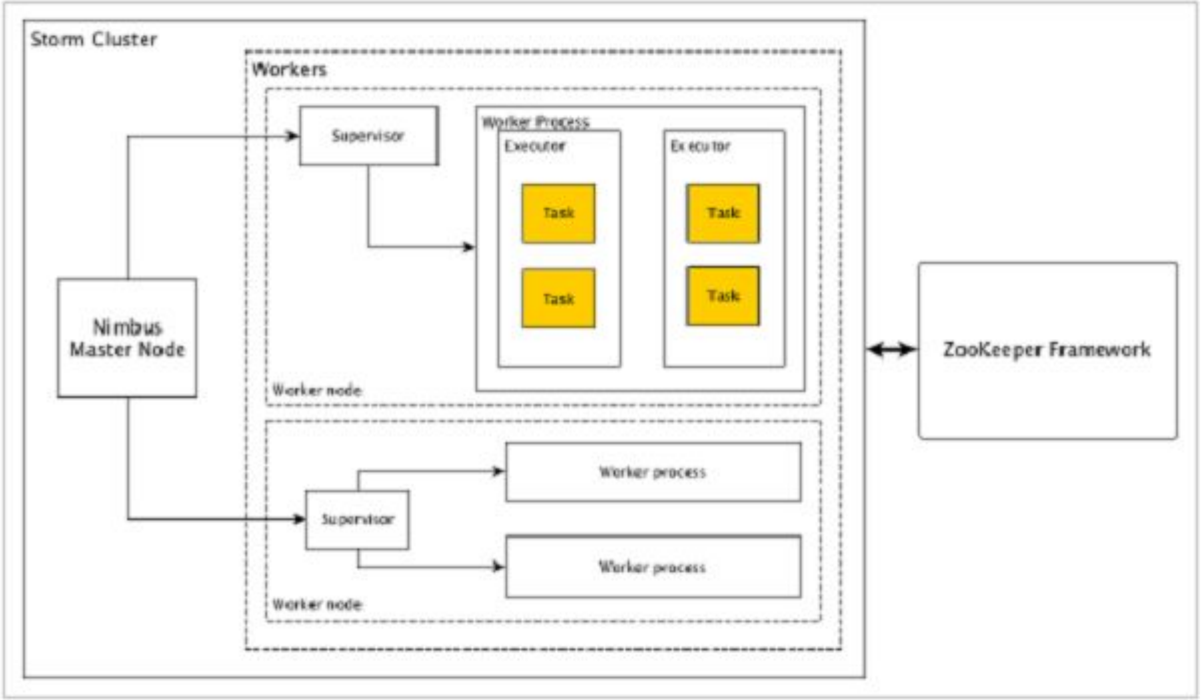
Storm

- Real-time stream processing
- A Storm streaming process can access tens of thousands messages per second on cluster.
- Local mode − This mode is used for development, testing, and debugging because it is the easiest way to see all the topology components working together. In this mode, we can adjust parameters that enable us to see how our topology runs in different Storm configuration environments. In Local mode, storm topologies run on the local machine in a single JVM.

Storm


Input data source
Spout
Stream source
Tuple stream
Bolt
Tuple stream
Bolt
Tuple stream
Bolt
Tuple stream
Tuple stream
Bolt
Tuple stream
Tuple stream
Bolt
intraction with data base
Data storage

Storm


Storm Cluster
Workers
Supervisor
Worker Process
Executor
Executor
Task
Task
Task
Task
Nimbus
Master Node
Worker node
ZooKeeper Framework
Worker process
Supervisor
Worker process
Worker node

Testing Environment

Local host:

1.3 GHz Intel Core i5

8 GB 1600 MHz DDR3

121.3 GB APPLE SSD SD0128F

Package:

Apache hadoop/3.1.0 in pseudo distributed mode

mahout/0.13.0

jdk/1.8.0_171

Apache spark/2.3.1 in Local mode

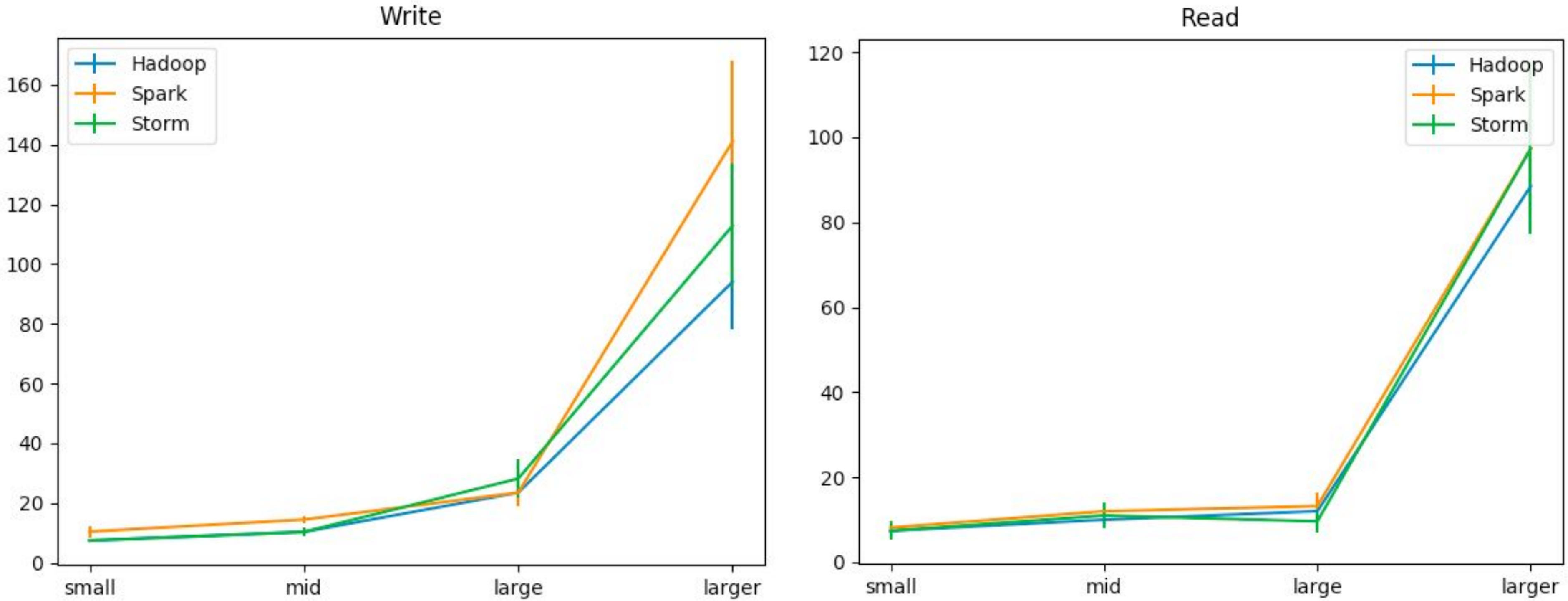
scala/2.12.6

python/3.6.5

Apache storm/1.22 in standalone mode

zookeeper/3.4.10

Read/ Write



Time (sec) consumption on different size of data. Small :285KB, Medium :240MB, Large: 756MB, Larger: 5.33GB

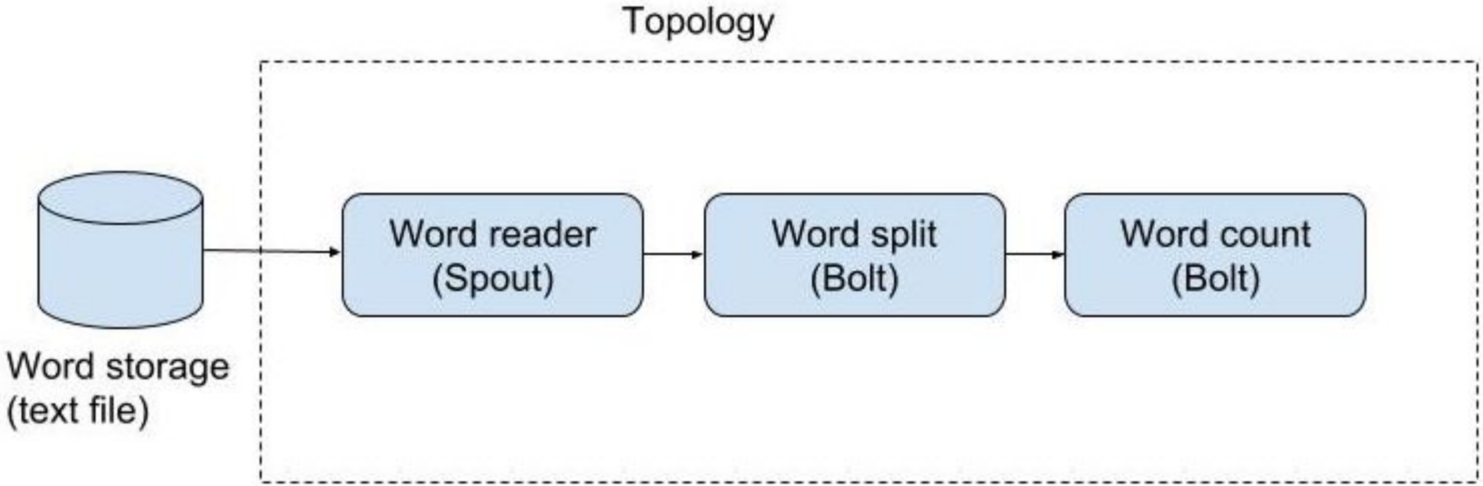
Word Count

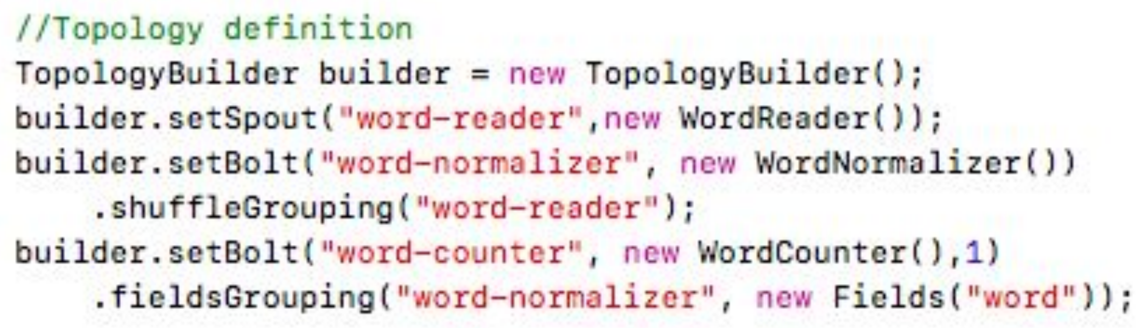
- Accumulate the frequency of each word

- Information retrieving like search engine

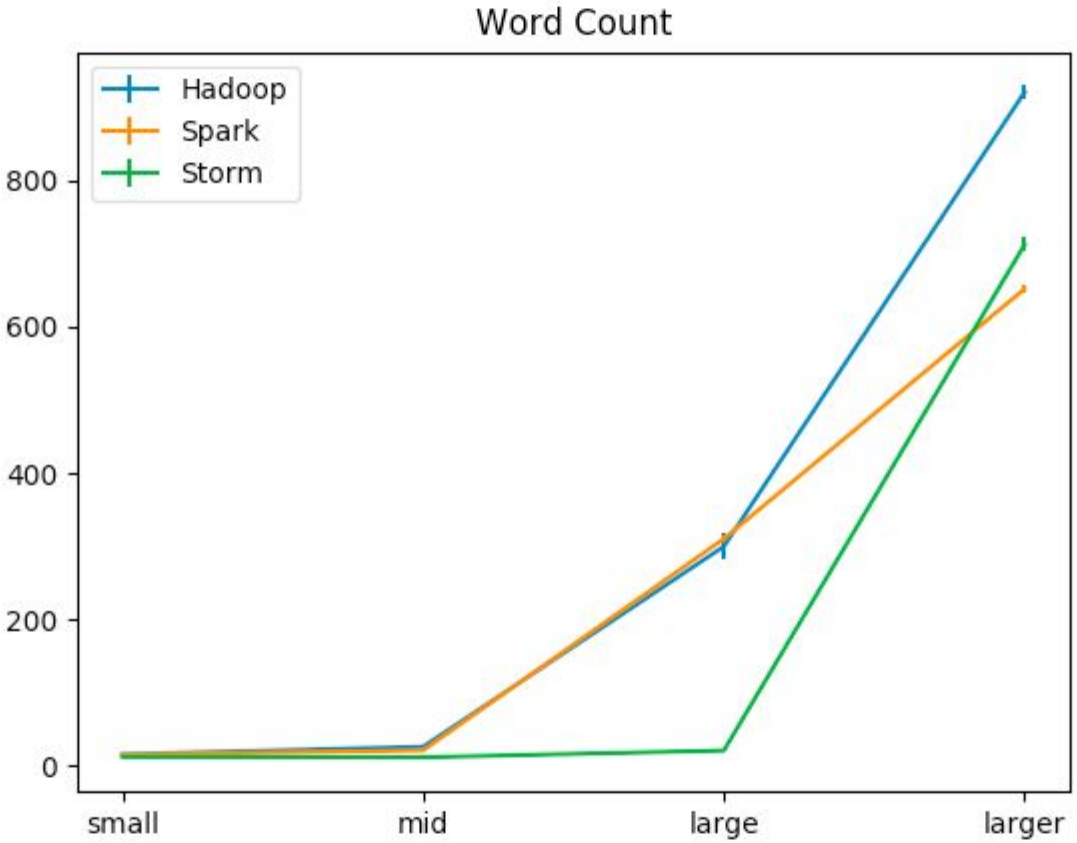
- Small Data: word_count.txt (58 KB)
  Medium Data: t8_shakespeare (5.5 MB)
  Large Data: dewiki_page_meta.xml (756 MB)
  Larger Data: enwiktionary.xml (5.33 GB)

Storm



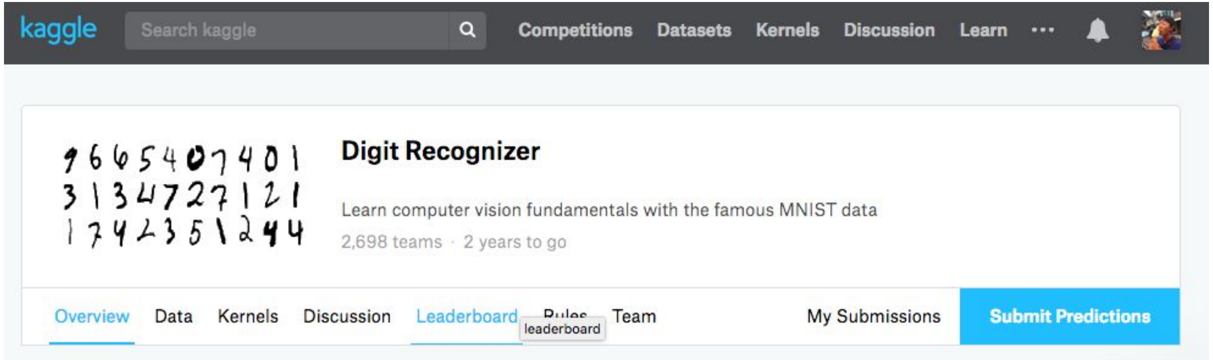


```
//Topology definition
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("word-reader",new WordReader());
builder.setBolt("word-normalizer", new WordNormalizer())
    .shuffleGrouping("word-reader");
builder.setBolt("word-counter", new WordCounter(),1)
    .fieldsGrouping("word-normalizer", new Fields("word"));
```



Time consumption (sec) on different size of text data

K-means



Digit Recognizer

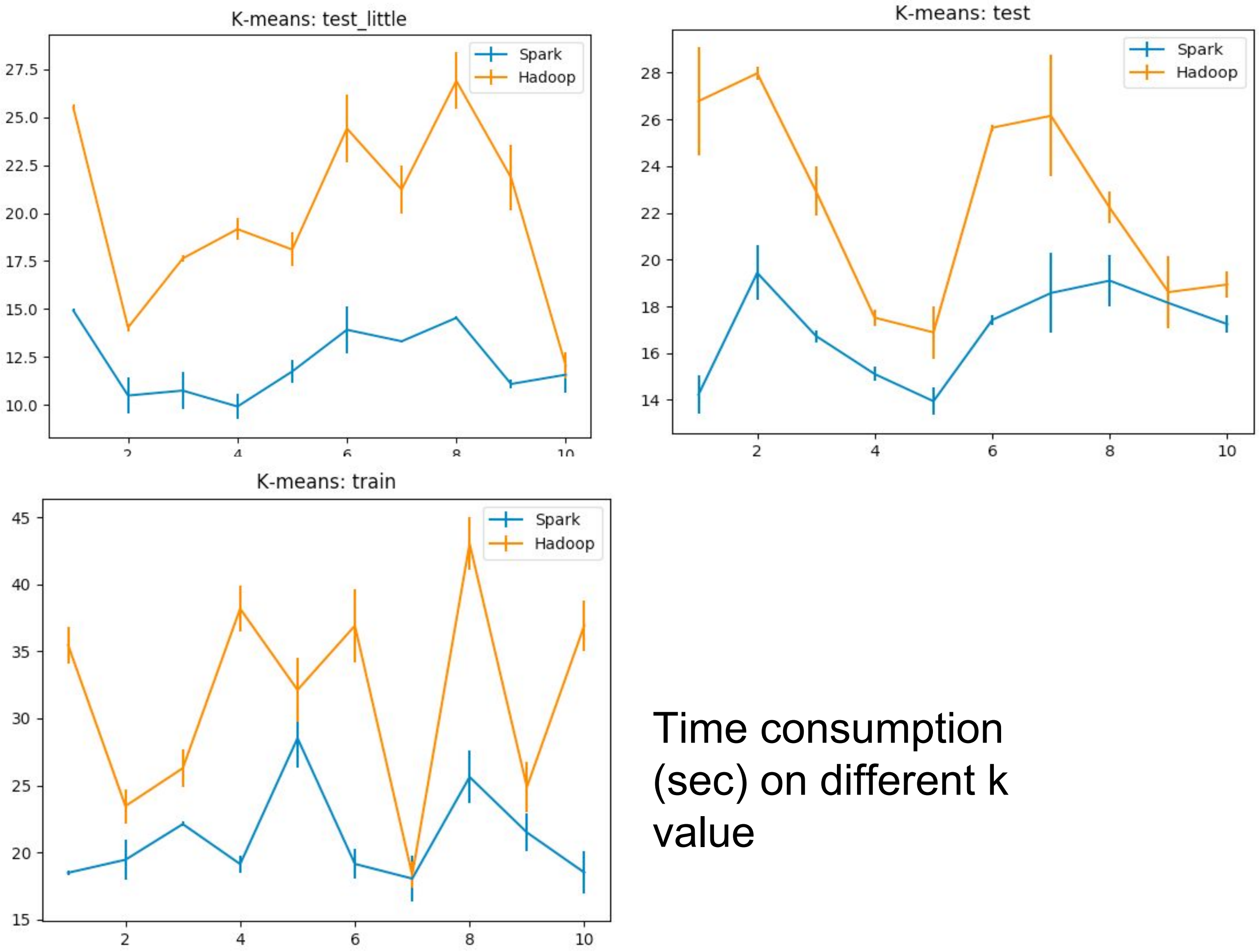
Learn computer vision fundamentals with the famous MNIST data

2,698 teams · 2 years to go

- https://www.kaggle.com/c/digit-recognizer/data
  Given raw data of image and label, predict the unlabeled data

- No need label when using k-means

- Test_little.csv: 3600 data points x 783 pixels
  Test.csv: 14000 data points
  Train.csv: 25500 data points



Time consumption (sec) on different k value

Conclusion & Future work

- Easy development environment excepted some package dependency

- Most of our experiments run under single host mode

- Other performance evaluation of file system
  Scalability: When # of clients increases
  Fault tolerance: server & host

- Other applications: Neural network