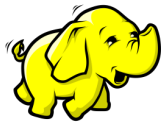
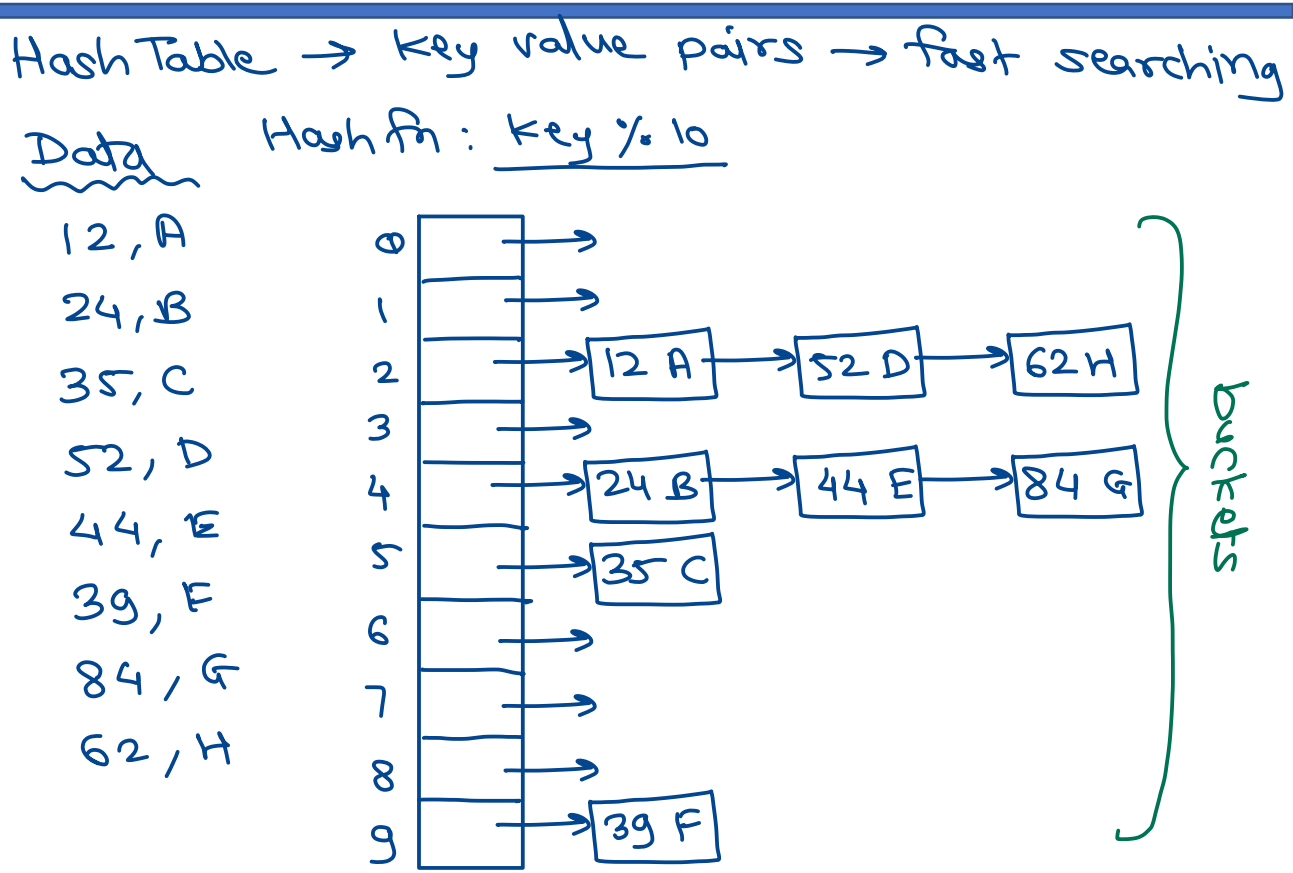


Apache Hive

Sunbeam Infotech

Bucketing

- Data in bucketed tables is divided into multiple files.
- When data is processed using MR job, number of reducers will be same as number of buckets.
- To insert data into bucketed table, it must be uploaded via staging table.
- Usually buckets are created on unique column(s) to uniformly divide data across multiple reducers. → Hash Partitioner
- It provides better sampling and speed-up map side joins.
- It is mandatory for DML operations.



Hive Indexes

- Similar to RDBMS index.
- To speed up SELECT queries (searching & grouping).
- Indexes internally store addresses of records for given column values.
- Creating index is time-taking job (for huge data). If indexing is done under load, then clients query performance is too low.
- In Hive indexes are created, but deferred for build (using ALTER statement).
- CREATE INDEX query doesn't create index, rather keep ready for building later.
- Index building should be triggered explicitly, when server is less loaded.

optimization technique in Hive 2.x

emp → INDEX job

ANALYST ~~, ~~
CLERK ~~, ~~, ~~, ~~
MANAGER ~~, ~~, ~~
PRESIDENT ~~
SALESMAN ~~, ~~, ~~, ~~

Indexes not supported in Hive 3.x

Hive Indexes

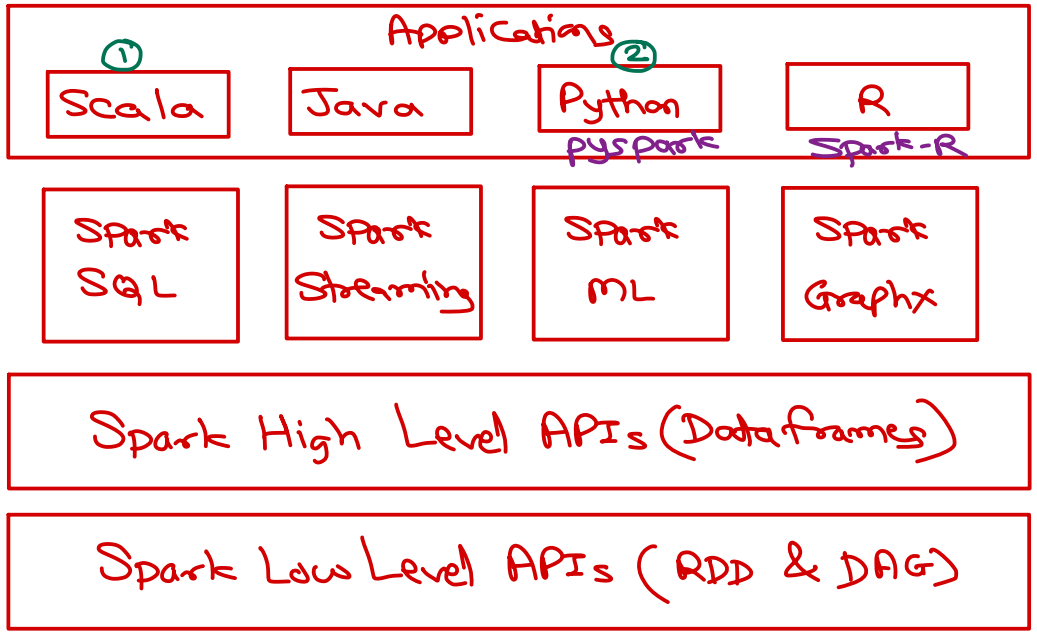
- In hive indexes are stored in HDFS (as hive tables).
- These indexes are build by different index handlers e.g. BITMAP, CompactHandler, …
- Compact:
  - Stores combination of indexed column value & its HDFS block id.
- Bitmap:
  - Stores combination of indexed column value & list of rows as bitmap.
  - Bitmap indexes work faster than Compact.
- Hive indexes are not supported from Hive 3.x onwards. Use materialized view instead to improve the performance.

Apache Spark

Sunbeam Infotech

Introduction

- Spark is Distributed computing framework, that can process huge amount of data.
- Spark can be used as eco-system of Hadoop or can be used as independent distributed computing framework.
- Developed by UCB AMPlabs division.
  - → Research projects on ML & algos for people
  - Spark is open-sourced under Apache.
- Further developed/maintained by DataBricks.
  - → enterprise Spark support & cloud hosting.
- Popular Spark vendors
  - DataBricks, AWS EMR, Cloudera, MapR
- Spark Toolkit
- Spark Philosophy
  - Unified → similar api for any language.
    similar performance (in high level api).
  - Compute Engine → works with any distributed storage
    e.g. HDFS, S3, AzureBlob,..
  - Libraries → third party libraries
    spark-packages.org



Hadoop vs Spark

- Distributed framework
  - Distributed storage + Distributed computing
- Hadoop is developed in Java (JVM based).
- Designed for commodity hardware.
  - Data is processed in RAM and spills on disk.
- In MapReduce job, mappers & reducers are executed as independent JVM processes.

- Distributed framework
  - Distributed computing
  - Not tied up with particular storage
- Spark is developed in Scala (JVM based).
- Needs better hardware config.
  - Data is processed fully in RAM to achieve faster execution.
- In Spark job, tasks are executed as threads in Executor process.

PySpark Development

- terminal> python3 –m pip install pyspark
- In ~/.profile
  - export PYSPARK_PYTHON=python3
  - export PYSPARK_DRIVER_PYTHON=python3
  - export SPARK_HOME=$HOME/.local/lib/python3.6/site-packages/pyspark
  - export PATH=$HOME/.local/bin:$PATH
- terminal> pyspark
  - file = sc.textFile("/home/nilesh/spark-2.4.4-bin-hadoop2.7/LICENSE")
  - lines = file.map(lambda line: line.lower())
  - words = lines.flatMap(lambda line: line.split())
  - word1s = words.map(lambda word: (word,1))
  - wordcounts = word1s.reduceByKey(lambda acc,cnt: acc + cnt)
  - result = wordcounts.collect()
  - print(result)

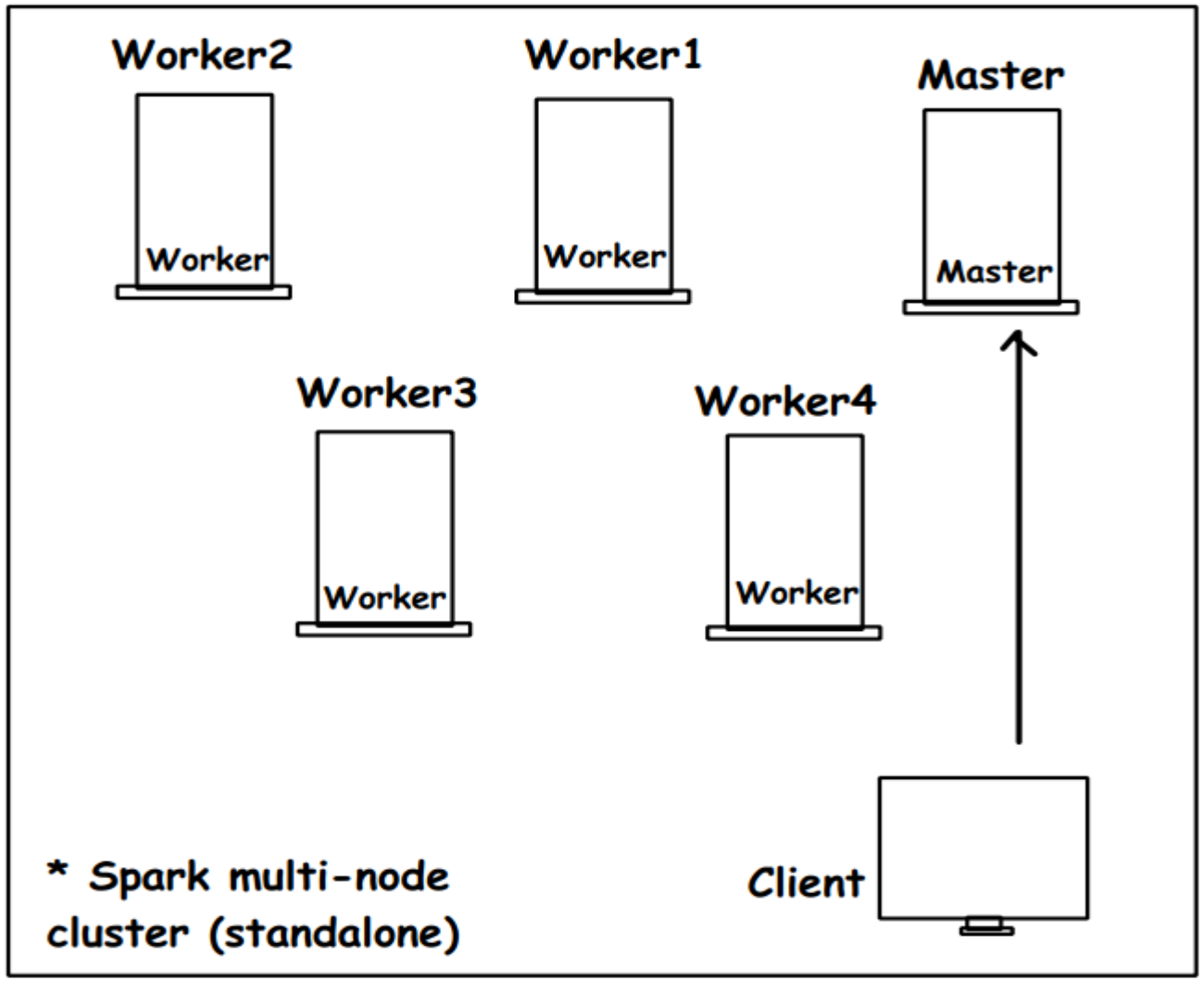
PySpark Development (PyCharm)

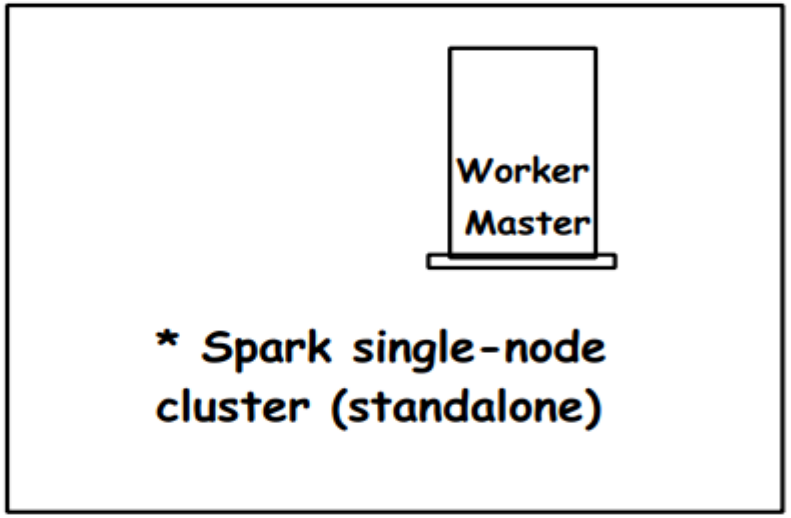
- PyCharm -> New Project
  - Select project location
  - Existing interpreter -> Python3.x
- Create Python file (hello.py)
  - from pyspark import SparkConf
  - from pyspark import SparkContext
  - conf = SparkConf().setAppName("Demo01").setMaster("local")
  - sc = SparkContext(conf=conf)
  - file = sc.textFile("/home/nilesh/spark-2.4.4-bin-hadoop2.7/LICENSE")
  - lines = file.map(lambda line: line.lower())
  - words = lines.flatMap(lambda line: line.split())
  - word1s = words.map(lambda word: (word,1))
  - wordcounts = word1s.reduceByKey(lambda acc,cnt: acc + cnt)
  - result = wordcounts.collect()
  - print(result)

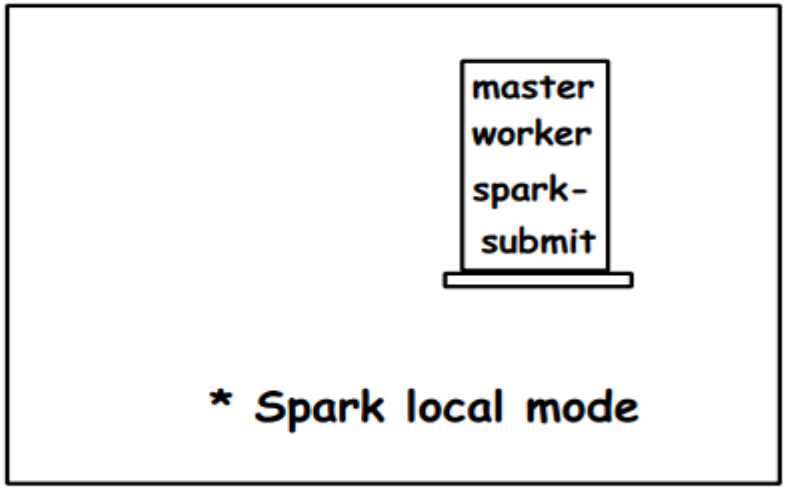
Spark RDD

- Resilient Distributed Dataset
  - Resilient
  - Distributed
  - Dataset
- RDD characteristics
  - Immutable
  - Lazily evaluated
  - Resilient

Spark Installation Modes







Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>