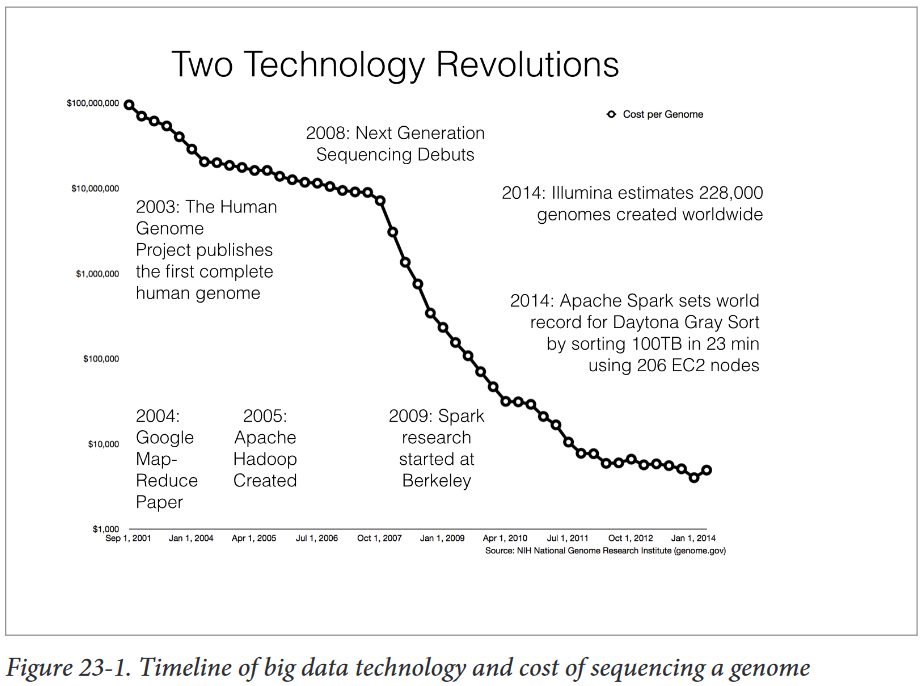
HADOOP



Apache Thrift and Google Protocol Buffers are both popular serialization frameworks,

and both are commonly used as a format for persistent binary data. There is limited

support for these as MapReduce formats; however, they are used internally in parts of

Hadoop for RPC and data exchange.

# INTRODUCTION

**Hadoop** est une infrastructure qui permet de gérer des ensembles de données volumineux dans un environnement informatique distribué.

# Apache Oozie

A compléter

# Apache Avro

A compléter

# Apache Parquet

A compléter

# Apache HivE

En matière de données, Hive présente trois fonctions principales : la synthèse, l’interrogation et l’analyse.

Il prend en charge des requêtes rédigées en langage HiveQL. Il traduit automatiquement les requêtes de type SQL en tâches MapReduce exécutées sur Hadoop. Parallèlement, HiveQL prend en charge les scripts MapReduce personnalisés qui se connectent aux requêtes. Hive autorise également la sérialisation/désérialisation des données, et accroît la flexibilité de la conception de schémas en intégrant un catalogue système appelé Hive-Metastore.

Selon le site Wiki d'Apache Hive, « Hive n'est pas conçu pour les charges de travail OLTP, et ne propose pas de requêtes en temps réel ou de mises à jour de niveau ligne. Il est optimisé pour les tâches en lots appliquées à des ensembles volumineux de données uniquement cumulatifs (comme les journaux de serveurs Web ou weblogs). »

Hive prend en charge les fichiers au format texte (appelés également fichiers plats), les fichiers SequenceFiles (fichiers plats constitués de paires clé/valeur binaires) et RCFiles (Record Columnar Files ; stockent des colonnes dans une table selon le mode d'une base de données en colonnes.)

# Apache Impala

**Impala** provides high-performance, low-latency SQL queries on data stored in popular Apache Hadoop file formats. The fast response for queries enables interactive exploration and fine-tuning of analytic queries, rather than long batch jobs traditionally associated with SQL-on-Hadoop technologies. (You will often see the term "interactive" applied to these kinds of fast queries with human-scale response times.)

Impala integrates with the Apache Hive metastore database, to share databases and tables between both components. The high level of integration with Hive, and compatibility with the HiveQL syntax, lets you use either Impala or Hive to create tables, issue queries, load data, and so on.

The following are some of the key advantages of Impala:

* Impala integrates with the existing CDH(Cloudera Distribution Hadoop) ecosystem, meaning data can be stored, shared, and accessed using the various solutions included with CDH. This also avoids data silos and minimizes expensive data movement.
* Impala provides access to data stored in CDH without requiring the Java skills required for MapReduce jobs. Impala can access data directly from the HDFS file system. Impala also provides a SQL front-end to access data in the HBase database system, or in the Amazon Simple Storage System (S3).
* Impala returns results typically within seconds or a few minutes, rather than the many minutes or hours that are often required for Hive queries to complete.
* Impala is pioneering the use of the Parquet file format, a columnar storage layout that is optimized for large-scale queries typical in data warehouse scenarios.

# Apache Hue

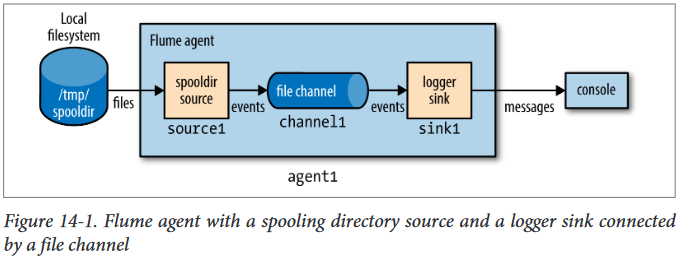
**Hue** est une interface web open-source sous licence Apache V2 prenant en charge Hadoop et son écosystème. Hue agrège les composants Apache Hadoop les plus courants en une seule interface en cherchant à améliorer l'expérience utilisateur. Son objectif principal est de permettre aux utilisateurs de ne pas se soucier de la complexité sous-jacente d' Hadoop et à limiter l'utilisation de ligne de commande.

## Application intégrées :

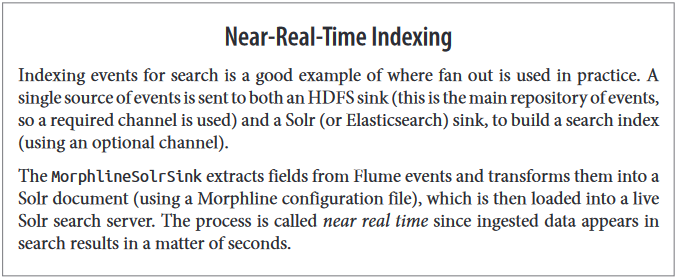
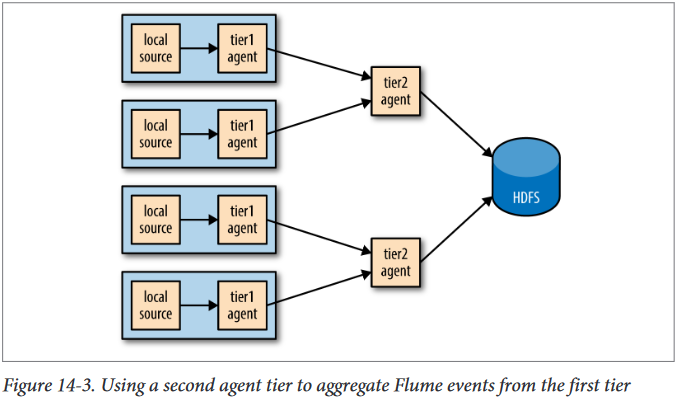
* Éditeur de requêtes pour [Apache Hive](https://fr.wikipedia.org/wiki/Apache_Hive), [Impala](https://fr.wikipedia.org/wiki/Impala_(Cloudera)) (incubation), [MySQL](https://fr.wikipedia.org/wiki/MySQL), Oracle, [PostgreSQL](https://fr.wikipedia.org/wiki/PostgreSQL), SparkSQL, [Apache Solr](https://fr.wikipedia.org/wiki/Apache_Solr) SQL, Apache Phoenix…
* La recherche dynamique des tableaux de bord avec Apache Solr
* notebook [Apache Spark](https://fr.wikipedia.org/wiki/Apache_Spark) et Apache Hadoop
* La planification de job et de workflows grâce à [Apache Oozie](https://fr.wikipedia.org/wiki/Apache_Oozie) et à l'éditeur de tableau de bord

# Apache Flume

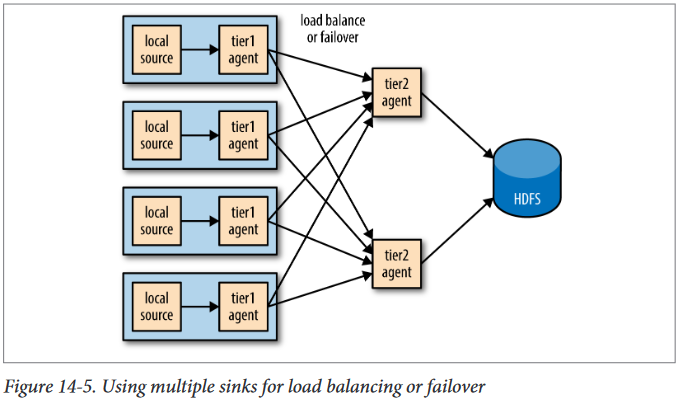
For this, you need the web clickstream data. The most common way to ingest web clickstream is to use Apache Flume. Flume is a scalable real-time ingest framework that allows you to route, filter, aggregate, and do "mini-operations" on data on its way in to the scalable processing platform.

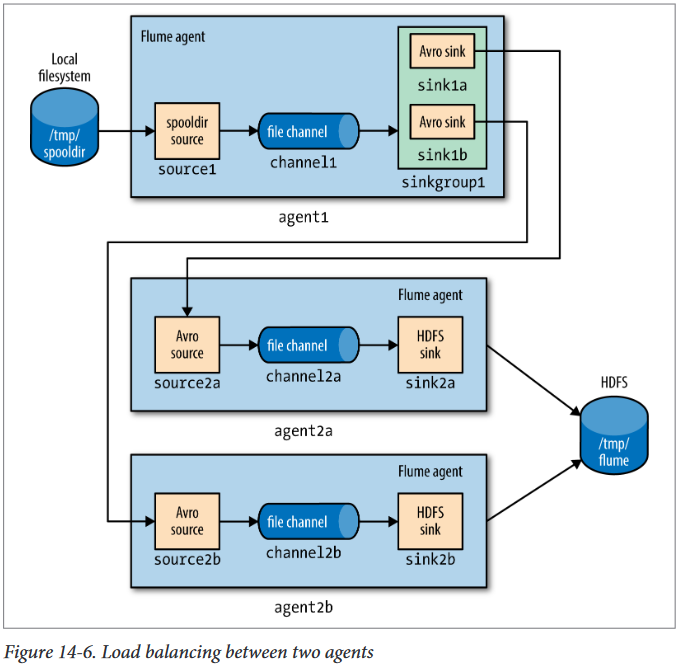


# 



# 





# 

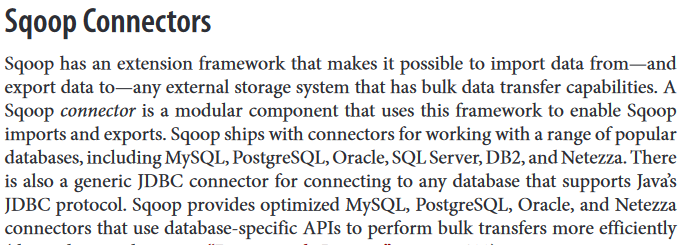
# 

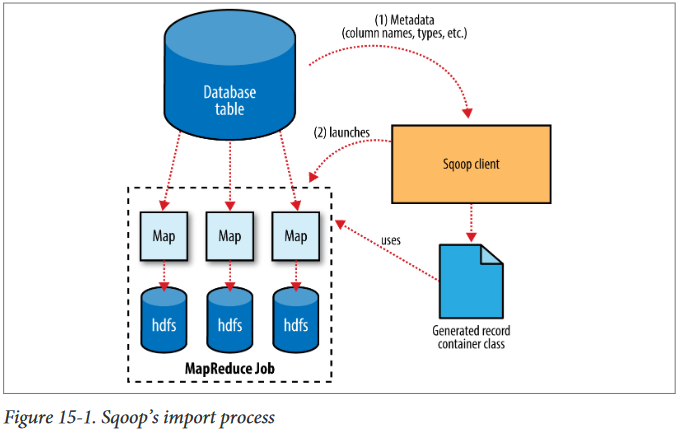
# 

# Apache Sqoop

**Sqoop** est une [interface en ligne de commande](https://fr.wikipedia.org/wiki/Interface_en_ligne_de_commande) de l'application pour transférer des données entre des [bases de données relationnelles](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es_relationnelle) et [Hadoop](https://fr.wikipedia.org/wiki/Hadoop). Il prend en charge le chargement différentiels d'une seule table ou d'une requête [SQL](https://fr.wikipedia.org/wiki/Select_(SQL)) ainsi que des tâches enregistrées qui peuvent être exécutées plusieurs fois pour importer les mises à jour effectuées dans une base de données depuis la dernière importation. Les imports peuvent également être utilisés pour remplir les tables dans [Hive](https://fr.wikipedia.org/wiki/Apache_Hive) ou [HBase](https://fr.wikipedia.org/wiki/HBase). les Exportations peuvent être utilisés pour mettre les données de Hadoop dans une base de données relationnelle. Le nom Sqoop est un [mot valise](https://fr.wikipedia.org/wiki/Mot_valise) constitué de sql et de hadoop.En mars 2012 Sqoop est devenu un projet haut niveau [d'Apache](https://fr.wikipedia.org/wiki/Apache_Software_Foundation) .

**Apache Sqoop, which is part of CDH, is that tool. The nice thing about Sqoop is that we can automatically load our relational data from MySQL into HDFS, while preserving the structure. With a few additional configuration parameters, we can take this one step further and load this relational data directly into a form ready to be queried by Apache Impala, the MPP analytic database included with CDH, and other workloads.**



 The MapReduce job launched by Sqoop uses an InputFormat that can read sections of a table from a database via JDBC. The DataDrivenDBInputFormat provided with Hadoop partitions a query’s results over several map tasks. Reading a table is typically done with a simple query such as:

SELECT col1, col2, col3,... FROM tableName

But often, better import performance can be gained by dividing this query across multiple nodes. This is done using a splitting column. Using metadata about the table, Sqoop will guess a good column to use for splitting the table (typically the primary key for the table, if one exists). The minimum and maximum values for the primary key column are retrieved, and then these are used in conjunction with a target number of tasks to determine the queries that each map task should issue.

For example, suppose the widgets table had 100,000 entries, with the id

column containing values 0 through 99,999. When importing this table, Sqoop would determine that id is the primary key column for the table. When starting the MapReduce job, the DataDrivenDBInputFormat

used to perform the import would issue a statement such as :

SELECT MIN(id), MAX(id) FROM widgets ;

These values would then be used to interpolate over the entire range of data. Assuming we specified that five map tasks should run in parallel (with -m 5), this would result in each map task executing queries such as

SELECT id, widget\_name, ... FROM widgets WHERE id >= 0 AND id < 20000,

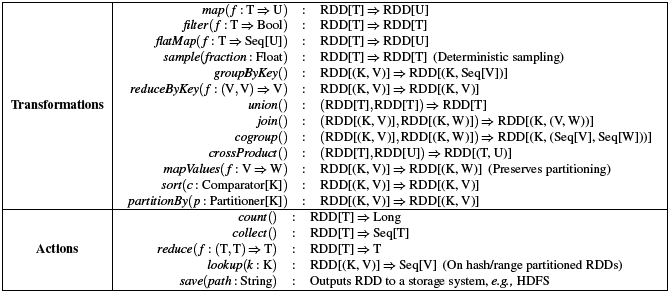
SELECT id, widget\_name, ... FROM widgets WHERE id >= 20000 AND id < 40000

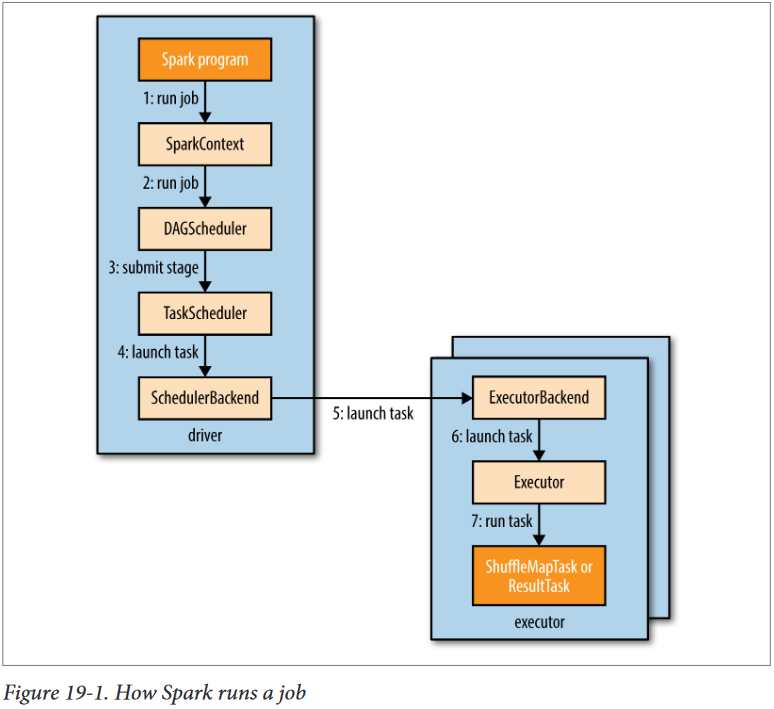
, and so on.

# Apache Pig

A compléter

# Apache Spark





## Shuffle map tasks

As the name suggests, shuffle map tasks are like the map-side part of the shuffle in

MapReduce. Each shuffle map task runs a computation on one RDD partition and,

based on a partitioning function, writes its output to a new set of partitions, which

are then fetched in a later stage (which could be composed of either shuffle map

tasks or result tasks). Shuffle map tasks run in all stages except the final stage.

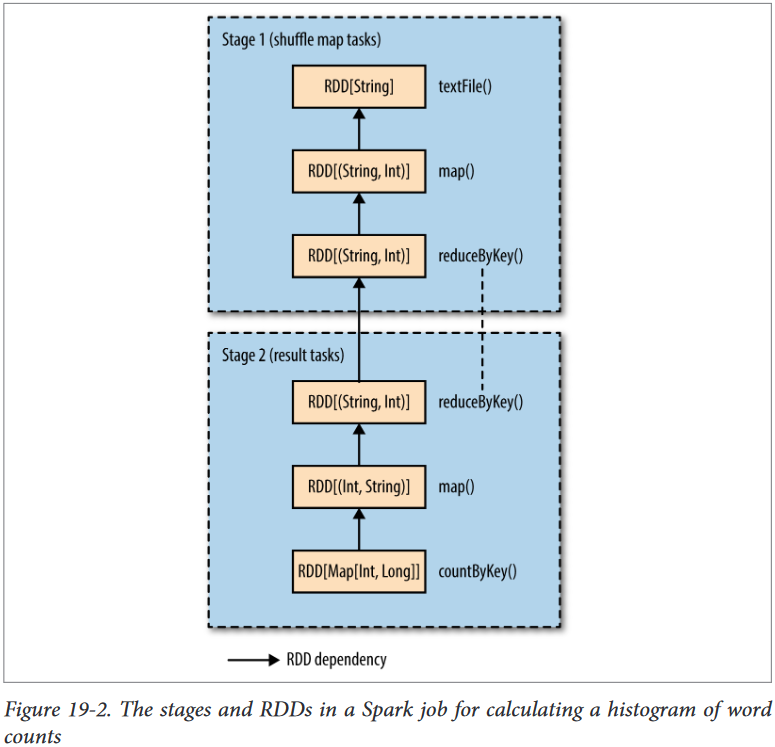
## Result tasks

Result tasks run in the final stage that returns the result to the user’s program (such

as the result of a count()). Each result task runs a computation on its RDD partition,

then sends the result back to the driver, and the driver assembles the results from

each partition into a final result (which may be Unit, in the case of actions like saveAsTextFile()).



YARN client mode is required for programs that have any interactive component, such

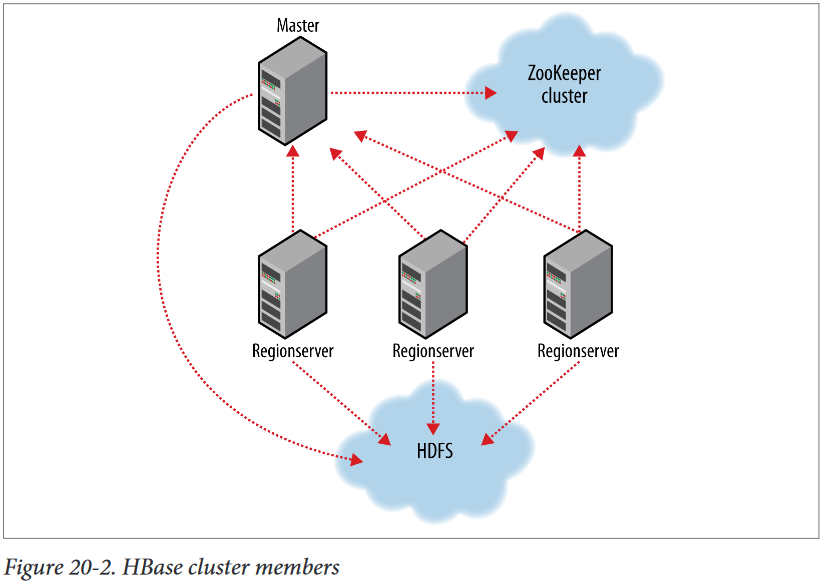
as spark-shell or pyspark. Client mode is also useful when building Spark programs,

since any debugging output is immediately visible.

# 

# 

# Apache Hbase



# Apache Solr

Solr (prononcé "solar") est une [plateforme logicielle](https://fr.wikipedia.org/wiki/Plate-forme_(informatique)) de [moteur de recherche](https://fr.wikipedia.org/wiki/Moteur_de_recherche) s'appuyant sur la bibliothèque de recherche [Lucene](https://fr.wikipedia.org/wiki/Lucene), créée par la [Fondation Apache](https://fr.wikipedia.org/wiki/Apache_Software_Foundation) et distribuée et conçue sous [licence libre](https://fr.wikipedia.org/wiki/Licence_libre).

*Solr* utilise le langage [Java](https://fr.wikipedia.org/wiki/Java_(langage)) et est exécuté par un [conteneur de servlets](https://fr.wikipedia.org/wiki/Conteneur_de_servlets)[1](https://fr.wikipedia.org/wiki/Apache_Solr#cite_note-1), comme [Tomcat,](https://fr.wikipedia.org/wiki/Apache_Tomcat) jusqu'à la version 5.0 puis devient un standalone Java. Il communique avec le client à l'aide d'une [interface de programmation](https://fr.wikipedia.org/wiki/Interface_de_programmation) en [XML](https://fr.wikipedia.org/wiki/Extensible_Markup_Language) et [JSON](https://fr.wikipedia.org/wiki/JavaScript_Object_Notation), généralement via le protocole [HTTP](https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol).

**https://www.cloudera.com/developers/get-started-with-hadoop-tutorial/exercise-1.html**

**You may notice that we told Sqoop to import the data into Hive but used Impala to query the data. This is because Hive and Impala can share both data files and the table metadata. Hive works by compiling SQL queries into MapReduce jobs, which makes it very flexible, whereas Impala executes queries itself and is built from the ground up to be as fast as possible, which makes it better for interactive analysis. We'll use Hive later for an ETL (extract-transform-load) workload.**

**Big Data technologies**

As already stated Big data does not refer only to the huge volume of data but the technologies that can store, process and analyze that data.

* **Apache Hadoop** – Hadoop is like synonymous to Big data. As there is a whole ecosystem build around Hadoop to work with Big data. MapReduce, Hive, Pig for data processing. HDFS as a data layer for storing data. Flume, Kafka, Sqoop for data ingestion. Oozie for job scheduling. Zookeeper for operational services.
* **NoSql Databases** – NoSql Databases doesn’t follow the rigid structure of relational databases and provide fast performance while storing data especially unstructured or semi-structured data. Examples of NoSql databases are MongoDB, Cassandra, Hbase.
* **R**- R is one of the most popular statistical analysis package which helps with analysis of the data.

In a nutshell, this is what Hadoop provides: a reliable, scalable platform for storage and

analysis. What’s more, because it runs on commodity hardware and is open source,

Hadoop is affordable.\\

For all its strengths, MapReduce is fundamentally a batch processing system, and is not

suitable for interactive analysis. You can’t run a query and get results back in a few

seconds or less. Queries typically take minutes or more, so it’s best for offline use, where

there isn’t a human sitting in the processing loop waiting for results.

**Interactive SQL**

By dispensing with MapReduce and using a distributed query engine that uses dedicated “always on” daemons (like Impala) or container reuse (like Hive on Tez),

it’s possible to achieve low-latency responses for SQL queries on Hadoop while still

scaling up to large dataset sizes.

**Iterative processing**

Many algorithms—such as those in machine learning—are iterative in nature, so

it’s much more efficient to hold each intermediate working set in memory, com‐

pared to loading from disk on each iteration. The architecture of MapReduce does

not allow this, but it’s straightforward with Spark, for example, and it enables a

highly exploratory style of working with datasets.

**Stream processing**

Streaming systems like Storm, Spark Streaming, or Samza make it possible to run

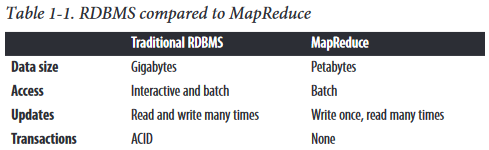
real-time, distributed computations on unbounded streams of data and emit results

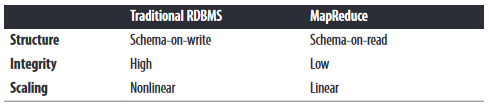
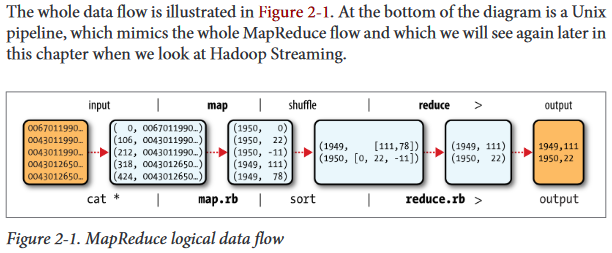
to Hadoop storage or external systems.

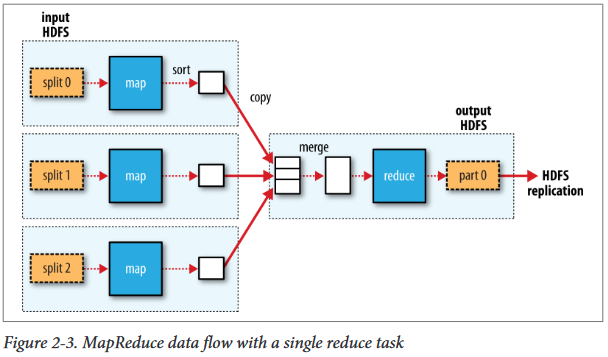
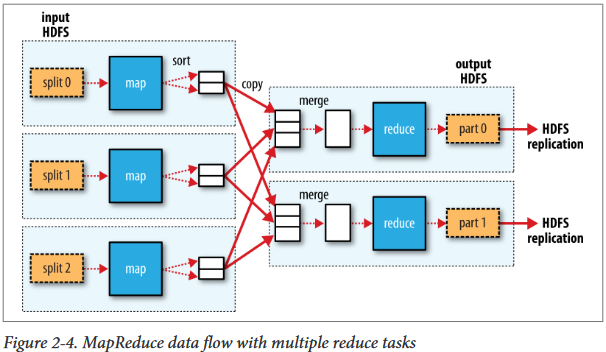
**Search**

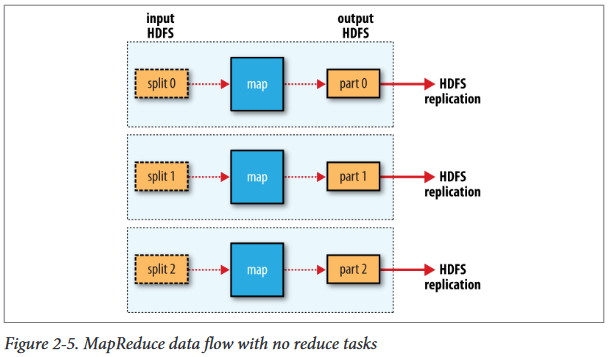
The Solr search platform can run on a Hadoop cluster, indexing documents as they

are added to HDFS, and serving search queries from indexes stored in HDFS.









## Areas where HDFS is not a good fit today :

**Low-latency data access :**

Applications that require low-latency access to data, in the tens of milliseconds

range, will not work well with HDFS. Remember, HDFS is optimized for delivering

a high throughput of data, and this may be at the expense of latency. HBase is currently a better choice for low-latency access.

**Lots of small files :**

Because the namenode holds filesystem metadata in memory, the limit to the num‐

ber of files in a filesystem is governed by the amount of memory on the

namenode.

As a rule of thumb, each file, directory, and block takes about 150 bytes. So, for

example, if you had one million files, each taking one block, you would need at least

300 MB of memory. Although storing millions of files is feasible, billions is beyond

the capability of current hardware.

**Multiple writers, arbitrary file modifications :**

Files in HDFS may be written to by a single writer. Writes are always made at the

end of the file, in append-only fashion. There is no support for multiple writers or

for modifications at arbitrary offsets in the file. (These might be supported in the

future, but they are likely to be relatively inefficient.)

## Config Hadoop Pseudo-Distributed:

File *.bashrc* :

export JAVA\_HOME=/usr/lib/jvm/java-8-oracle

export HADOOP\_HOME=/home/djebali/hadoop-3.1.1

export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin

|  |
| --- |
| **Si configuration perso :** |
| export HADOOP\_CONF\_DIR=/home/djebali/hadoop\_conf\_dir |

Command :

ssh-keygen -t rsa -P '' -f ~/.ssh/id\_rsa

cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

ssh localhost

hdfs namenode -format

start-dfs.sh

start-yarn.sh

mr-jobhistory-daemon.sh start historyserver

|  |
| --- |
| **Si configuration perso :** |
| start-dfs.sh --config path-to-config-directory |
| start-yarn.sh --config path-to-config-directory |
| mr-jobhistory-daemon.sh --config path-to-config-directory start historyserver |

mr-jobhistory-daemon.sh stop historyserver

stop-yarn.sh

stop-dfs.sh

hadoop fs -mkdir -p /user/$USER

**Browse the web interface** – You can also check the web interfaces for Namenode and YARN resource manager after the daemons are started.

**NameNode** – http://localhost:50070/

**ResourceManager** – http://localhost:8088/

## Basic Filesystem Operations

Start by copying a file from the local filesystem to HDFS :

**hadoop fs -copyFromLocal /media/sf\_my\_linux/hadoop-book/input/docs/quangle.txt hdfs://localhost/user/djebali/quangle.txt**

This command invokes Hadoop’s filesystem shell command ***fs***, which supports a number of subcommands—in this case, we are running -copyFromLocal. The local file *quangle.txt*

is copied to the file */user/djebali/quangle.txt* on the HDFS instance running on

localhost. In fact, we could have omitted the scheme and host of the URI and picked up

the default, hdfs://localhost, as specified in core-site.xml :

**hadoop fs -copyFromLocal /media/sf\_my\_linux/hadoop-book/input/docs/quangle.txt /user/djebali/quangle.txt**

**≈**

**hadoop fs -copyFromLocal /media/sf\_my\_linux/hadoop-book/input/docs/quangle.txt quangle.txt**

Copy the file back to the local filesystem and check whethers it’s the same :

**hadoop fs -copyToLocal quangle.txt quangle.copy.txt**

We check the SHA signature of the file :

**md5sum /media/sf\_my\_linux/hadoop-book/input/docs/quangle.txt quangle.copy.txt**

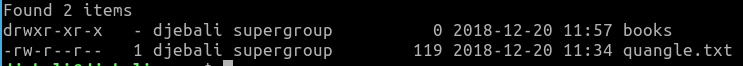
The MD5 digests are the same, showing that the file survived its trip to HDFS and is

back intact.

Finally, let’s look at an HDFS file listing. We create a directory first just to see how it is displayed in the listing:

**hadoop fs -mkdir books**

**hadoop fs -ls .**



Puisque quangle.txt est une copie un 1 est affiché car nous avons initialisé le facteur de réplication à 1. Le concept de réplication ne s’applique pas au répertoire, car les dossiers ne sont pas traités comme des ***metadata*** et sont sauvegardés par le ***namenode,*** non pas par le ***datanodes.***

## Hadoop Filesystems HDFS

## YARN(Yet Another Ressource Negotiator)

Apache YARN (Yet Another Resource Negotiator) is Hadoop’s cluster resource man‐

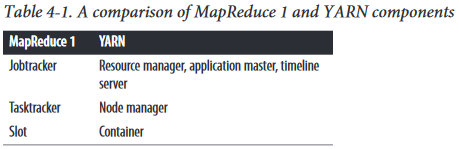
agement system. YARN was introduced in Hadoop 2 to improve the MapReduce im‐

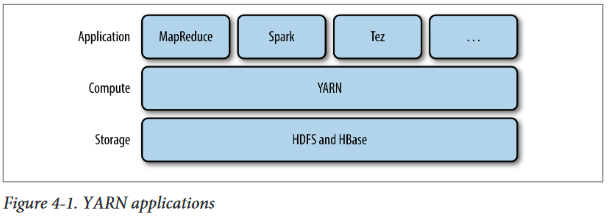
plementation, but it is general enough to support other distributed computing para‐

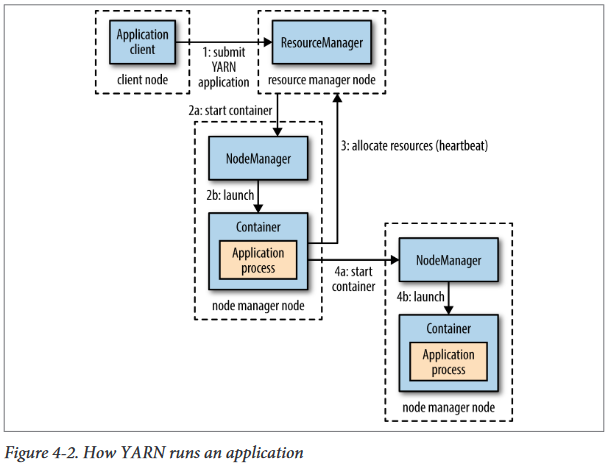
digms as well.

YARN provides APIs for requesting and working with cluster resources, but these APIs are not typically used directly by user code. Instead, users write to higher-level APIs provided by distributed computing frameworks, which themselves are built on YARN and hide the resource management details from the user. The situation is illustrated in Figure 4-1, which shows some distributed computing frameworks (MapReduce, Spark,and so on) running as YARN applications on the cluster compute layer (YARN) and the cluster storage layer (HDFS and HBase).

There is also a layer of applications that build on the frameworks shown in Figure 4-1, Pig, Hive, and Crunch are all examples of processing frameworks that run on MapReduce, Spark, or Tez (or on all three), and don’t interact with YARN directly.

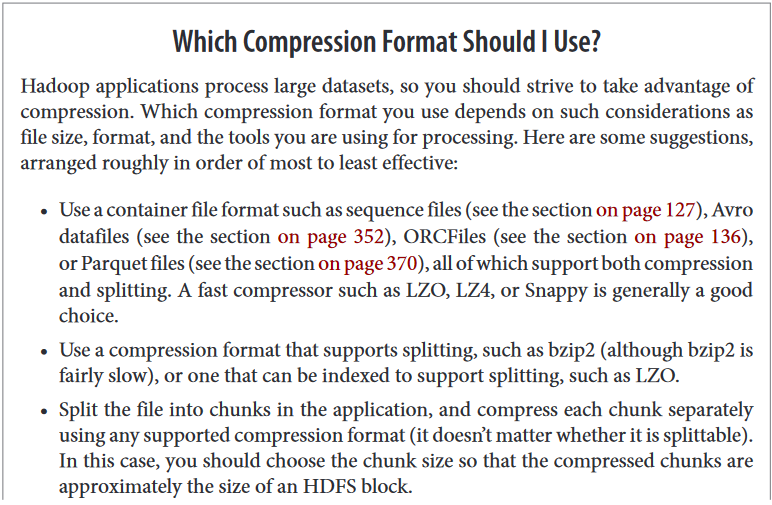
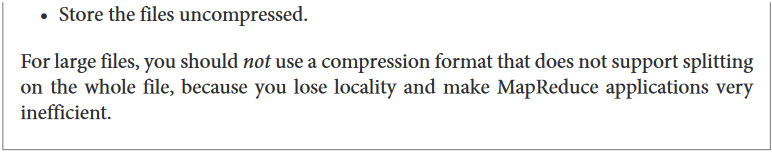






DRF Dominant Resource Fairness permet de préciser à YARN lors de l’allocation de ressources pour la création de conteneurs pour les besoins d’une application de prendre en compte non seulement les besoins en mémoire mais aussi les besoins en CPU.

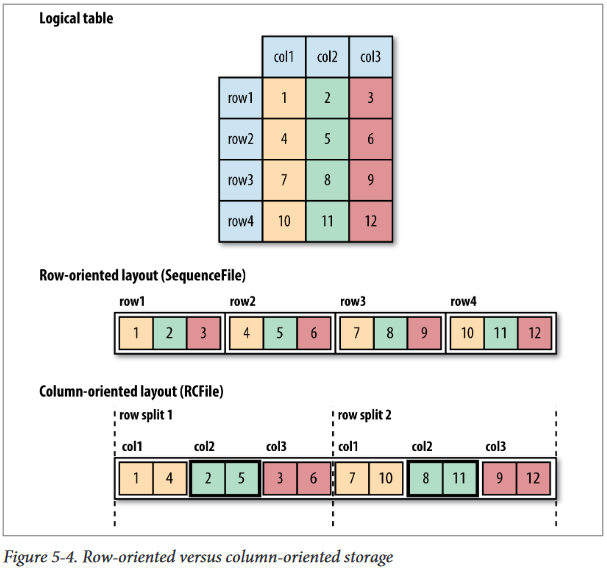
Hadoop use checksum ti verify the integrity of data. Data blocks with corrupted checksum are report to the namenode and after hadoop try to correct them by use the copy of the data for replacing them.



# Apache Crunch

A compléter

# Sequence File



# Hadoop configuration

# MapReduce Application Mechanism

