Big Data Management.
Sesión 03. Hadoop Distributed File System (HDFS).

Autor: Ziani El Ali, Adil.

adilziani.wordpress.com

26 de abril de 2020

Índice

Qué es HDSF

Recordemos antes **qué es Hadoop**: Hadoop no es un programa ni una base de datos, hadoop es un ecosistema, un proyecto Apache con un **conjunto de herramientas** enfocadas a la gestión de grandes volúmenes de datos y de diferente variedad, desde el almacenamiento hasta la explotación de los datos en un entorno distribuido.

HDFS (Hadoop Distributed File System) es un **sistema de ficheros distribuido** integrado por defecto en el ecosistema Apache Hadoop, basado en Google File System. HDFS está pensado para almacenar gran cantidad de datos de manera distribuida en un Cluster Hadoop con la idea de que los datos sean explotados en paralelo.

- HDFS está pensado para trabajar con ficheros de gran tamaño. Hoy en día existen proyectos hadoop con petabytes de datos.

Capacidad	Nodos	Clientes	Ficheros
10PB	10.000	100.000	100.000.000

- HDFS está desarrollado con la idea de que los casos de uso más usuales consisten en escribir datos una vez y leer muchas veces. Por tanto su arquitectura premia este caso de uso.
- La infraestructura consiste en cientos o miles de máquinas (nodos) con lo cual la probabilidad de que falle una máquina es alta. HDFS implementa técnicas de tolerancia a fallos.
- Hadoop permite integrarse con otros sistemas de ficheros como Amazon S3

Más en detalle?, vease (White, 2015)

Arquitectura HDFS

- Un único **Master** (NameNode)
  - Se encarga de gestionar el Namespace, contiene en catálogo global. Mantiene el catálogo en memoria $\sim 1GB/1PB$
  - Hace de coordenador sobre los DataNodes, contiene metadata.
  - Recibe conexiones de Client
  - Gestiona Block size, número replicas, TypeFiles, ...
    - Block size por defecto 128MB
    - Número replicas por defecto es 3 réplicas de cada bloque
  - Heartbeat hacia los workers para conecer su estado de operatibilidad.
    - Por defecto cada 3s se manda un mensaje (Heartbeat) a cada DataNode
    - Si no se reciben respuestas por algún DataNode en 10 minutos, NameNode ordena replicar los datos de dicho nodo.
- Diversos **DataNodes**, también llamados Slaves, Workers, Servers
  - Se encargan de guardar los bloques que reciben
- Existe la posibilidad de levantar un **secundary NameNode** que contenga un backup del SpaceName en caso de que falle el NameNode principal.

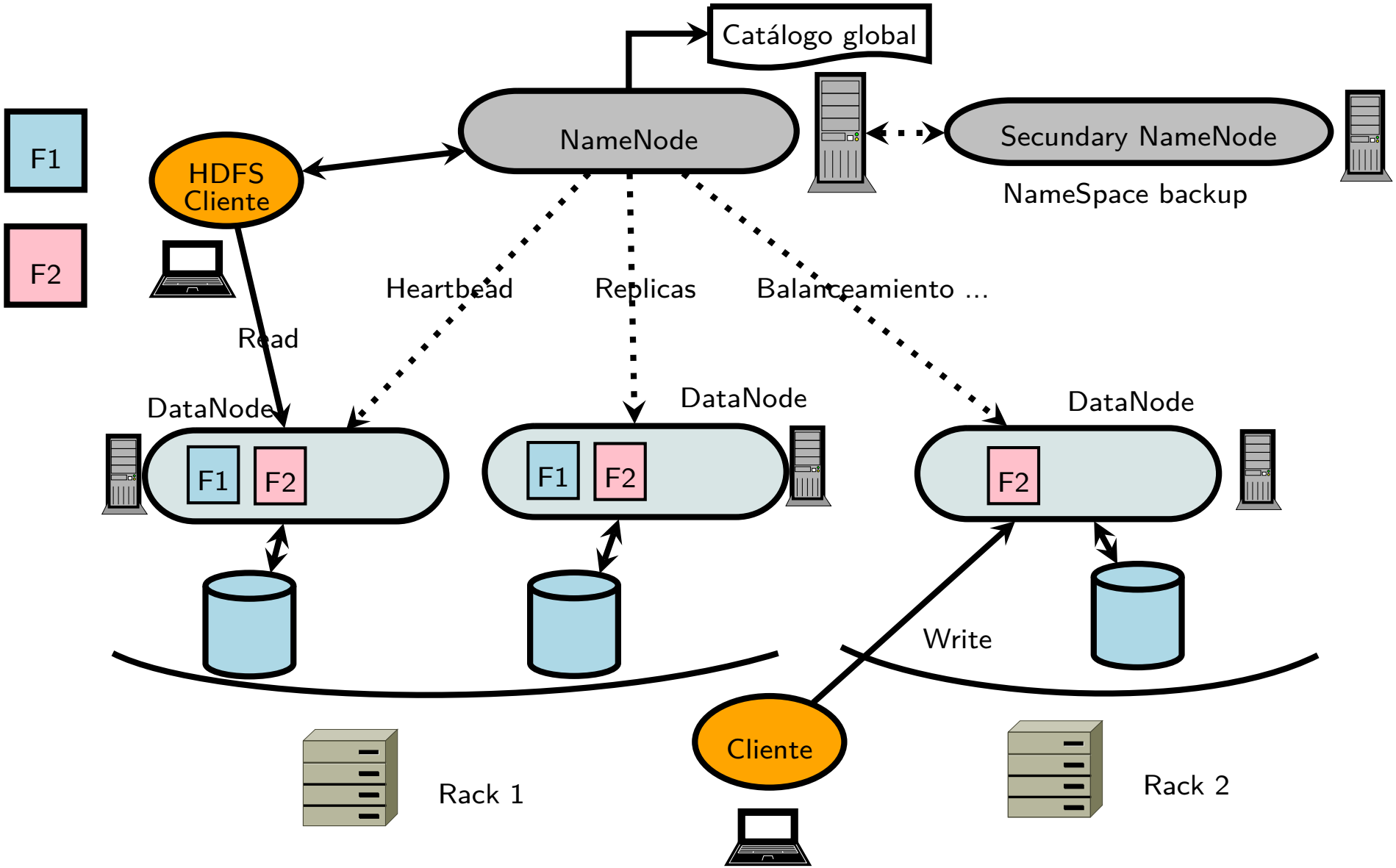
Fragmentación, Réplicas y balanceamiento

- **Fragmentación**
  Los ficheros se fragmentan según el formato en que se van a almacenar. Cada partición del fichero es un conjunto de bloques que se distribuyen sobre los datanodes.
  La distribución permite ejecutar jobs en paralelo.
  Cómo definir el tamaño de los bloques para tener un número de particiones interesante?

- **Réplicas**
  Por defecto se hacen 3 réplicas de cada bloque. Replicar permite disponer de un sistema tolerante a fallos. Si un worker falla, el dato existirá en otro worker.
  Y si nos interesa otro número de réplicas?

- **Balanceamiento**
  Es actualmente un challange pues HDFS distribuye los bloques de forma random que solo en ficheros grandes, con muchos bloques, se da un balanceamiento "uniforme" entre los datanodes. Existen proyectos que intentan balancear el cluster.
  El balanceamiento sirve para no sobrecargar a algunos datanodes mas que otros.
  Cómo podemos mejorar el balanceamiento de un fichero mal distribuido debido a un balanceamiento azaroso?[1]

[1] Responderemos a estas cuestiones en la parte práctica de la sesión


Catálogo global
NameNode
Secundary NameNode
NameSpace backup
F1
F2
HDFS
Cliente
Heartbeat
Replicas
Balanceamiento ...
Read
DataNode
DataNode
DataNode
F1
F2
F1
F2
F2
Write
Cliente
Rack 1
Rack 2

File Formats en HDFS

Diseños de particionado:

- **Horizontal**
  - Para ficheros más propensos a ser escaneados en horizontal y en su totalidad.
  - Ejemplos de diseños: **SequenceFile, Avro**
- **Vertical**
  - Para ficheros más propensos a consultarse en proyecciones. Leer subconjunto de columnas.
  - Ejemplos de diseños: Yahoo Zebra
- **Híbrido**
  - Para ficheros más propensos a consumirse en proyecciones y por particiones. Combinación de particionamiento horizontal y vertical.
  - Ejemplos de diseño; **Parquet**

HDFS por defecto almacena los datos sin definir ningún deseño ni esquema: *PlainText*,csv,JSON. No es eficiente para realizar consultas y se debe, a menudo, parsear al formato del documento.

Considerar un diseño a la hora de almacenar ficheros no sólo ayuda al particionado sino también a la manera de acceder a los datos. **Y cómo escoger un formato para almacenar datos?**

Diseño horizontal: *SequenceFile*

- Formato de entrada/salida ampliamente utilizado en HDFS
- Su diseño es el de un fichero plano de pares Key-value con la siguiente organización

Header						Key-value pairs				
						Record				sync-marker
version 4Bytes	keyClassName String 4Bytes	valueClassName String 4Bytes	compression Boolean 1Bytes	blockCompression Boolean 1Bytes	compressor class String 4Bytes	Record length 4Bytes	Key length 4Bytes	Key Long	Value	16 Bytes

  - Header de tamaño fijo
    * Version: "SEQX" donde X indica la versión.Ej SEQ4 o SEQ6
    * compression: indica si los registros en forma key-value están compremidos
    * blockCompression: especifica si la compresión de bloques está activada para key-value
  - Key-value pairs
    * Los registros se almacenan en formato key-value formando records que se almacenan en bloques
    * Cada x bytes, encontramos un marcador **sync-marker**, que permite acceder a un punto aleatorio en el fichero

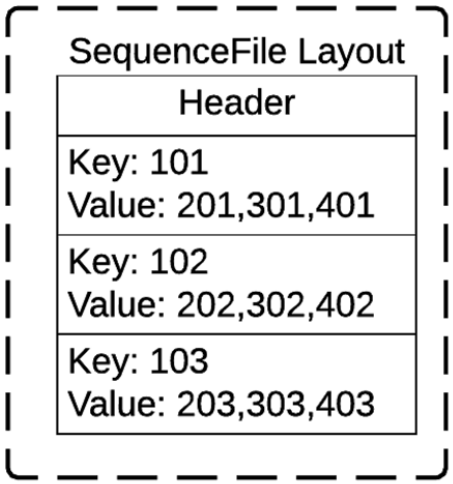


Table 1

A	B	C	D
101	201	301	401
102	202	302	402
103	203	303	403

Figura 1: By Rana Faisal Mounir (R.F. y col., 2016)

Diseño Horizontal: *Avro*

- Formato de datos binario compacto y rápido.
- Ampliamente utilizado para serialización.
- Header con esquema que se almacena con el fichero en formato JSON. Cada partición contiene el esquema que ayuda en la lectura de los datos.
- La organización de las particiones es:

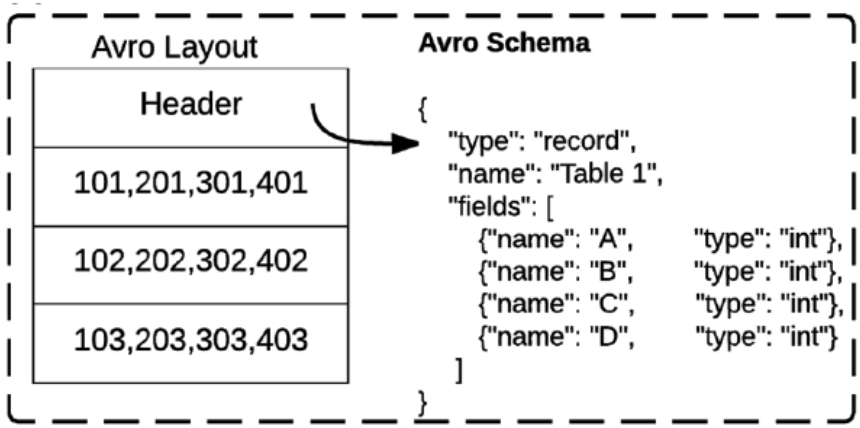
Header				Record				sync-marker
version 5Bytes	Schema JSON	compression Codec 4Bytes	sync-marker 16Bytes	#Rows in each block 8Bytes	Row size 8Bytes	Row	Row	16 Bytes

  - Schema describe el fichero en formato JSON
  - sync-marker después de cada bloque lo que permite que los archivos puedan dividirse en bloques y puedan ser compremidos, los metadatos almacenados permitirán a ir de bloque en bloque.

- En caso de que el fichero cambia su esquema, es posible modificar el schema.

Table 1

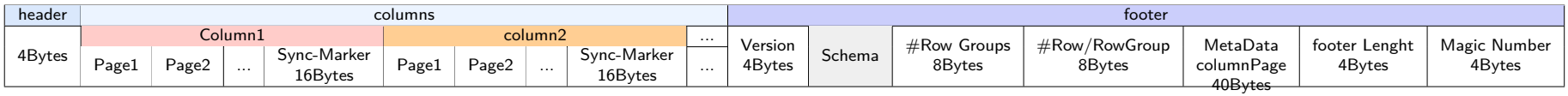
A	B	C	D
101	201	301	401
102	202	302	402
103	203	303	403



By Rana Faisal Mounir

Diseño Híbrido: *Parquet*

- Formato orientado a columnas y que también particiona el fichero en *Row Groups* (particionado horizontal)
- Permite estructuras anidadas en lugar de únicamente ficheros planos
- Los datos se particionan primero en *Row Groups* lo que define el grado de paralelismo, y posteriormente en columnas que a su vez forman *column pages*.
- Los ficheros *Parquet* se organizan en *Header*, *Columns*, *Footer*:



  - Header: contiene el formato del fichero, txt, JSON,etc
  - Sync-Markers: después de cada column
  - footer
    * Versión de Parquet
    * Schema del fichero almacenado
    * Column metadata: encodings, paths, número de valores, tamaño compremido, tamaño sin compremir, [2]
    * Magic number: número de la partición

- Parquet computa estadísticas sobre los atributos del fichero, esto permite computar estadísticas más rápido.

[2] Para más detalles `https://parquet.apache.org/documentation/latest/`

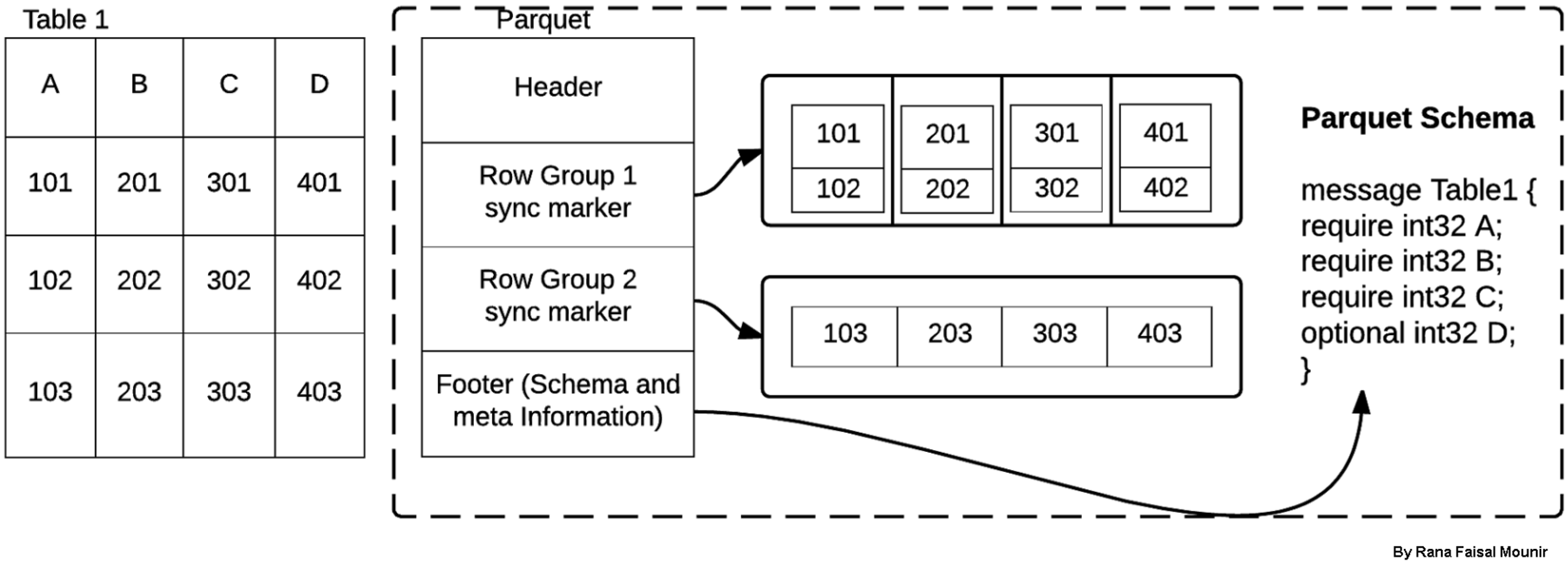


Table 1
A
B
C
D
101
201
301
401
102
202
302
402
103
203
303
403
Parquet
Header
Row Group 1
sync marker
Row Group 2
sync marker
Footer (Schema and
meta Information)
101
201
301
401
102
202
302
402
103
203
303
403
Parquet Schema
message Table1 {
require int32 A;
require int32 B;
require int32 C;
optional int32 D;
}
By Rana Faisal Mounir

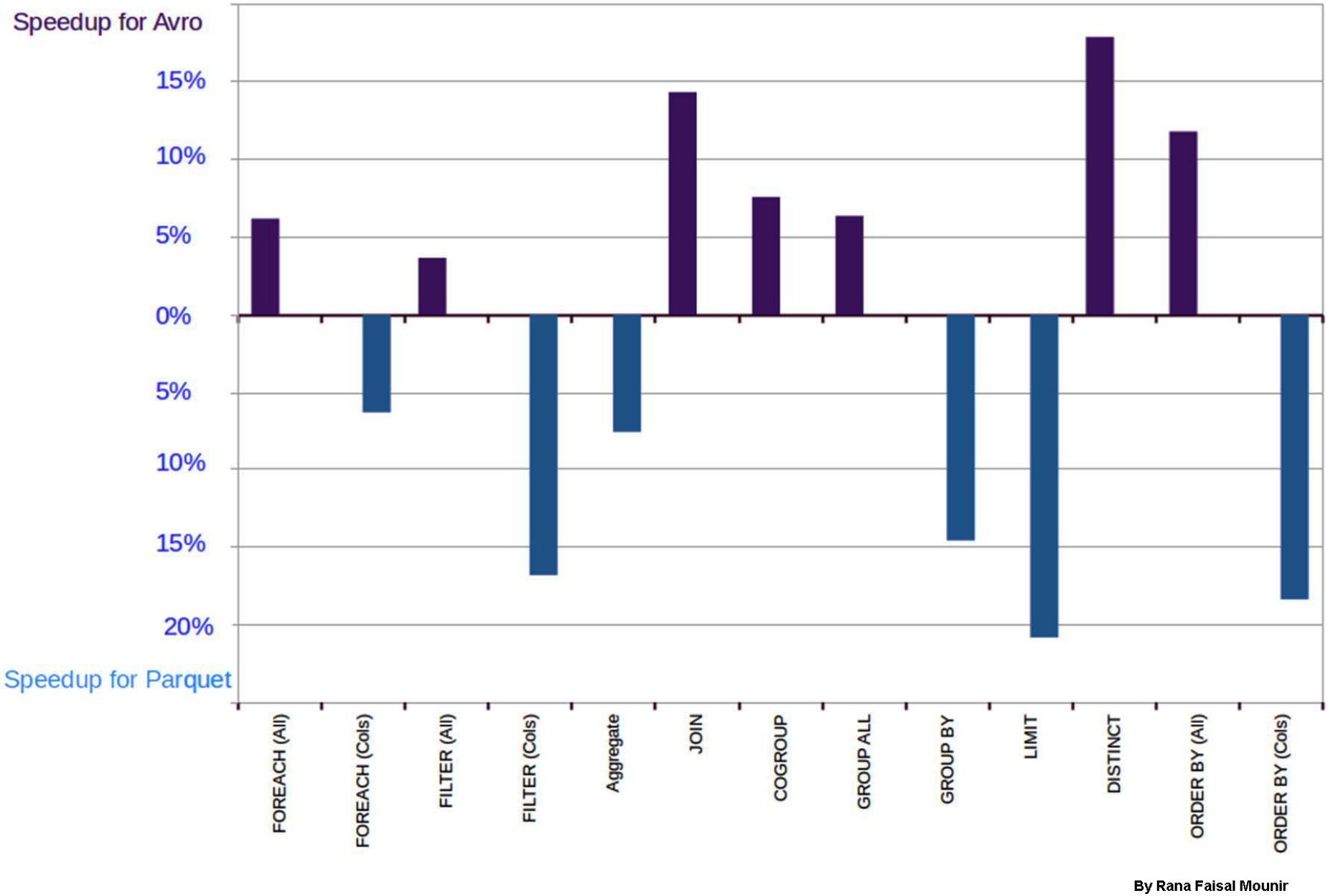
Comparación

Features	Horizontal		Vertical	Hybrid	
	Sequence Files	Avro	Yahoo Zebra	ORC	Parquet
Schema	No	Yes	Yes	Yes	Yes
Column Pruning	No	No	Yes	Yes	Yes
Predicate Pushdown	No	No	No	Yes	Yes
Indexing Information	No	No	No	Yes	Yes
Statistics Information	No	No	No	Yes	Yes
Nested Records	No	No	Yes	Yes	Yes

By Rana Faisal Mounir

Aclaraciones:

- Column Pruning se refiere a la posibilidad de leer las columnas que interesan sin leer todo el fichero
- Predicate Pushdown se refiere a la posibilidad de llevar ciertos filtros de una consulta a los datos, computar el filtrado y devolver el resultado. Esto reduce el tráfico por la red de los datos.
- Neested recordes se refiere a la posibilidad de tener una estructura de datos más compleja donde un registro puede ser embebido en otro. Estructuras anidadas.


Speedup for Avro
15%
10%
5%
0%
5%
10%
15%
20%
Speedup for Parquet
FOREACH (All)
FOREACH (Cols)
FILTER (All)
FILTER (Cols)
Aggregate
JOIN
COGROUP
GROUP ALL
GROUP BY
LIMIT
DISTINCT
ORDER BY (All)
ORDER BY (Cols)
By Rana Faisal Mounir

Compresión de datos en Hadoop

- La compresión permite reducir tamaño de ficheros y también cantidad de datos que se mueven por la red en el Cluster.
- En cambio la compresion/decompresión añade cómputo, con lo cual añade uso de CPU
- Algunos ejemplos de codec y algoritmos:

Codec	File Extension	Splittable?*	Degree of Compression	Compression Speed
Gzip	.gz	No	Medium	Medium
Bzip2	.bz2	Yes	High	Slow
Snappy	.snappy	No	Medium	Fast
LZO	.lzo	No, unless indexed	Medium	Fast
LZ4	.lz4	No, unless using 4mc	Medium	Fast

*Splittable: hace referencia a la posibilidad de que una partición de un fichero comprimido se pueda descomprimir sin necesidad de considerar el total del fichero. Es importante, en caso de querer comprimir ficheros

SequenceFile

- Bzip2, Gzip
- 3 niveles de compresión[3]
  - NONE: sin compresión
  - RECORD: Comprime únicamente los value, cada value por separado
  - BLOCK: comprime a nivel de record, varios record comprimidos juntos en bloques.
- Permite splitting

Avro

- Snappy, Gzip
- Compresión a nivel de bloque
- Permite splitting

Parquet

- Compresion a nivel de pages
- LZO, LZ4, Snappy, Gzip[4]
- Permite splitting

[3] https://hadoop.apache.org/docs/r1.0.4/api/org/apache/hadoop/io/SequenceFile.CompressionType.html

[4] https://www.javadoc.io/doc/org.apache.parquet/parquet-common/1.10.0/org/apache/parquet/hadoop/metadata/CompressionCodecName.html

Cómo escoger *file format* y tipo de compresión

En Big Data el lema es: **distinto problema distinta solución**!

Aplicado a este contexto, no hay un formato o algoritmo de compresión válido para todos los casos y con mejores resultados tanto en performance como reducción almacenamiento. Dependerá de diversos factores de los cuales algunos son:

File Format

- Cuál es la estructura de mi fichero?, es mas *Row-oriented*(SequentialFile, Avro) o *column-oriented*(Parquet).
- Querré comprimir los datos?, el algoritmo de compresión es admitido por qué formatos.
- Mi fichero tiene estructuras complejas? arrays, neested records,...(Parquet)

Compresión

- Más rapidez en la compresión y descompresión vs más ratio de compresión?
- Mi fichero se consulta con poca frecuencia (*Cold Data*) o con más frecuencia (*Hot Data*)?, opciones para *Cold Data* Gzip o Bzip2, para *Hot Data* Snappy o LZO serían opciones a considerar.
- Tengo jobs de MapReduce y busco paralelismo?, necesito entonces que el algoritmo sea *splittable*, LZO o LZ4

Para practicar aspectos de *Block Size y Balanceamiento* no se olvide de realizar la práctica "Management_sesion03_Hands-on02"

Para practicar aspectos de File Formats y Compresión no se olvide de realizar la práctica "Management_sesion03_Hands-on03"

En la próxima sesión

Sesion 04: **Big Data Procesing.**

- Escalabilidad y Elasticidad
- Fases de una consulta distribuida
- Data Shipping y Query Shipping
- ...

referencias

R.F., Munir y col. (2016). "ResilientStore: A Heuristic-Based Data Format Selector for Intermediate Results". En: *Model and Data Engineering. MEDI 2016. Lecture Notes in Computer Science* 9893.1, págs. 42-56.

White, Tom (2015). *Hadoop: The Definitive Guide, Fourth Edition*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: OReilly Media, Inc. ISBN: 978-1-491-90163-2.