Fuentes de datos

- Spark soporta distintos formatos de ficheros y BDs
  - RDDs
  - Tablas Hive
  - BD externas usando JDBC
  - Ficheros JSON
  - Ficheros Parquet
  - Ficheros ORC

JSON

- Es el formato que se usa para desarrollo web como alternativa a XML
- Spark infiere automáticamente el esquema de un fichero JSON
  - => útil para enviar y recibir datos a sistema externo
- No es un formato eficiente para almacenamiento permanente
- Sencillo, fácil de usar y legible

ORC (Optimized Row Columnar)

- Fue diseñado para almacenar datos Hive de forma mas eficiente (reemplazó a RCFile)
- Basado en columnas
  - Los datos de una columna se almacenan de forma contigua
- Usa mecanismos de compresión (Zlib, Snappy, …)

Parquet

- BD orientada a columnas del ecosistema Hadoop
- Proyecto conjunto de Twitter y Cloudera
- Los valores de cada columna se almacenan en posiciones de memoria contiguas
- Se comprime por columnas => ahorro de espacio
- Técnicas de compresión específicas para cada tipo
- No es necesario leer toda la fila para resolver consultas
- Independiente de Hive y de ningún entorno específico de desarrollo => más popular que ORC

BD orientadas a columnas

- SGBD que almacena tablas por columnas en vez de por filas
- Tanto BD orientados a filas como orientadas a columnas usan SQL
- Pero se aumenta el rendimiento de las consultas sobre todo en grandes conjuntos de datos

Tabla ejemplo

RowId	EmpId	Lastname	Firstname	Salary
001	10	Smith	Joe	40000
002	12	Jones	Mary	50000
003	11	Johnson	Cathy	44000
004	22	Jones	Bob	55000

Sistemas orientados a filas

```
001:10,Smith,Joe,40000;
002:12,Jones,Mary,50000;
003:11,Johnson,Cathy,44000;
004:22,Jones,Bob,55000;
```

- Encontrar todos los registros con salarios entre 40000 y 50000 => el SGBD debe recorrer toda la tabla buscando los registros que cumplen la condición
- Si la tabla tiene unos cientos de registros no cabrá en un único bloque de disco y habrá que hacer varias operaciones de disco para recuperarla y examinarla

Sistemas orientados a filas (II)

Índice sobre la columna salario

```
001:40000;
003:44000;
002:50000;
004:55000;
```

- Para mejorar el rendimiento se usan índices que almacenan un conjunto de columnas junto con punteros a la tabla original
- Son más pequeños que las tablas y reducen el número de operaciones de disco
- Pero mantenerlos es costoso: cuando se actualiza la tabla hay que actualizar el índice

Sistemas orientados a filas (III)

- Existen las BD en-memoria (MMDB) que almacenan todos los datos en memoria RAM
- No requieren índices
- Solo pueden manejar BDs que caben en memoria

Sistemas orientados a columnas

- Serializan todos los valores de cada columna

```
10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
40000:001,50000:002,44000:003,55000:004;
```

- Cualquiera de las columnas parece tener la estructura de un índice

Sistemas orientados a columnas (II)

```
10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
40000:001,50000:002,44000:003,55000:004;
```

- Diferencia
  - Orientada a filas: la clave primaria es el id de fila
  - Orientada a columnas: la clave primaria es el dato

```
…;Smith:001;Jones:002,004;Johnson:003;…
```

- Consultas eficientes
  - Encuentra todas las personas con apellido Jones
  - Contar número de registros que cumplen cierta condición

Sistemas orientados a columnas (III)

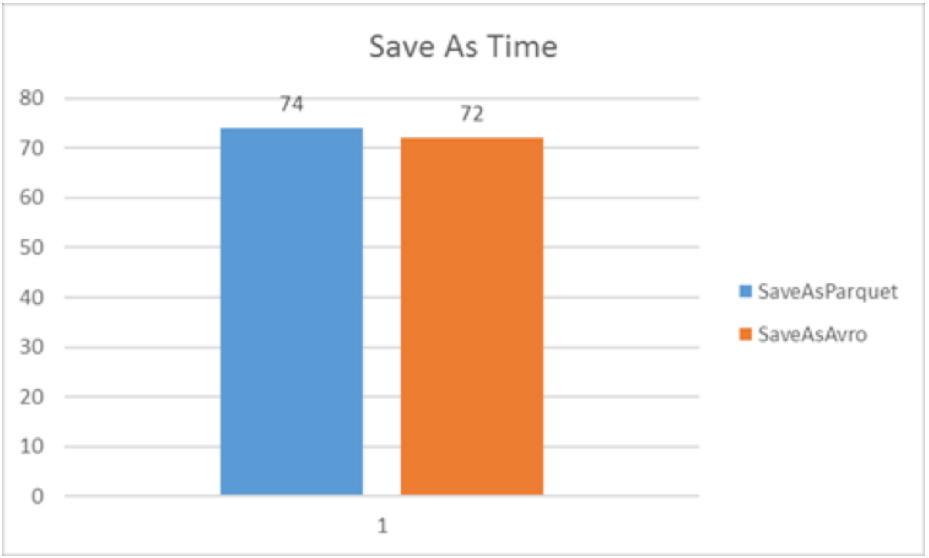
- Para operaciones que devuelven todos los datos de un objeto (fila completa) son más lentas
- Pero estas operaciones son menos frecuentes
- En Big Data, un cuello de botella son los accesos a disco => las BDs orientadas a columna aumentan rendimiento reduciendo el número de accesos a disco
  - Comprimiendo datos
  - Leyendo solo los datos necesarios para resolver la consulta

Comparación AVRO <-> parquet

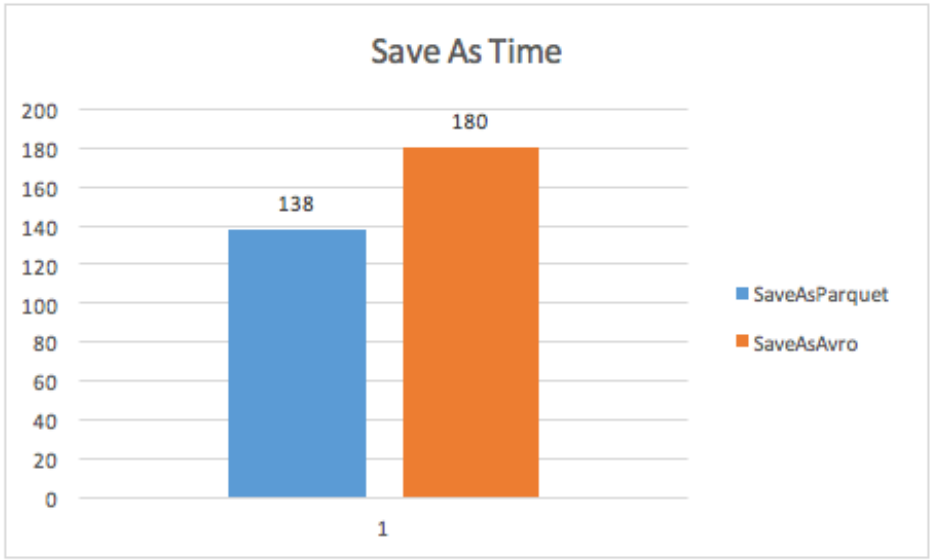
- AVRO
  - Formato de fichero utilizado en Hadoop
  - Orientado a filas
- Test datasets
  - Narrow dataset: 3 columnas, 82.8 millones de filas 3.9GB (CSV)
  - Wide dataset: 103 columnas, 694 millones de filas 194GB (CSV)
- Cluster con 100+ data nodes
- Tiempo en segundos
- Spark 1.6

Primer test

- Tiempo de creación de los ficheros Avro y Parquet después de leer el fichero en un DataFrame



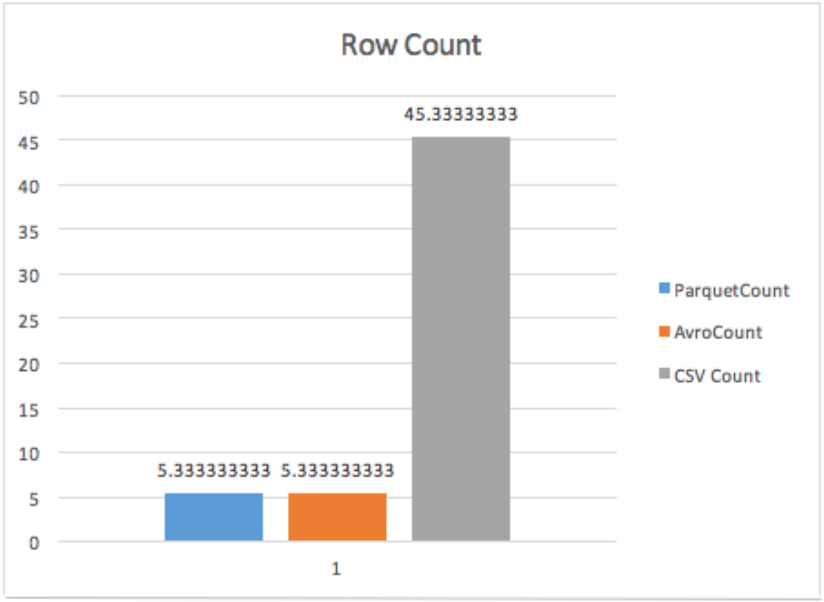
Narrow DS



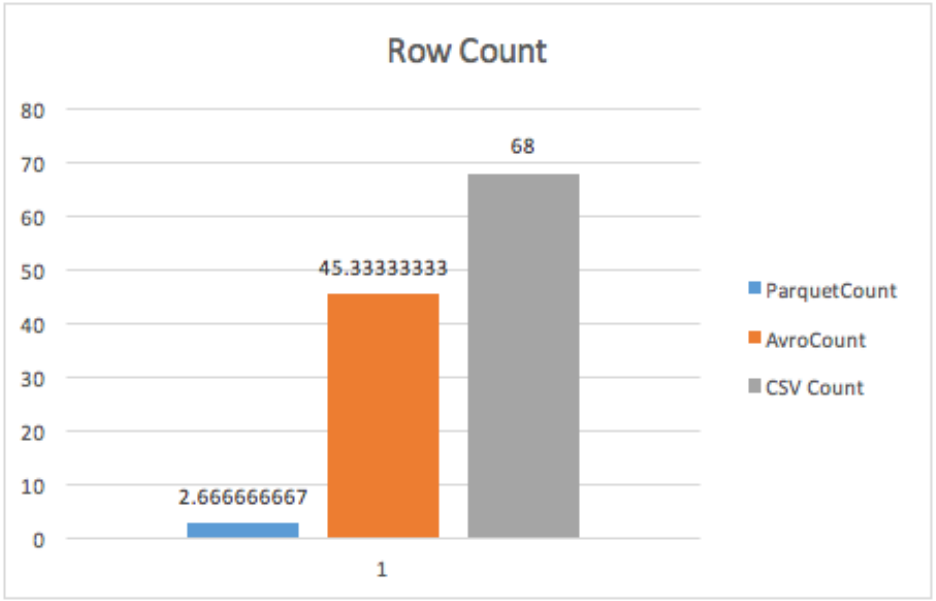
Wide DS

Segundo test

- Tiempo en contar filas



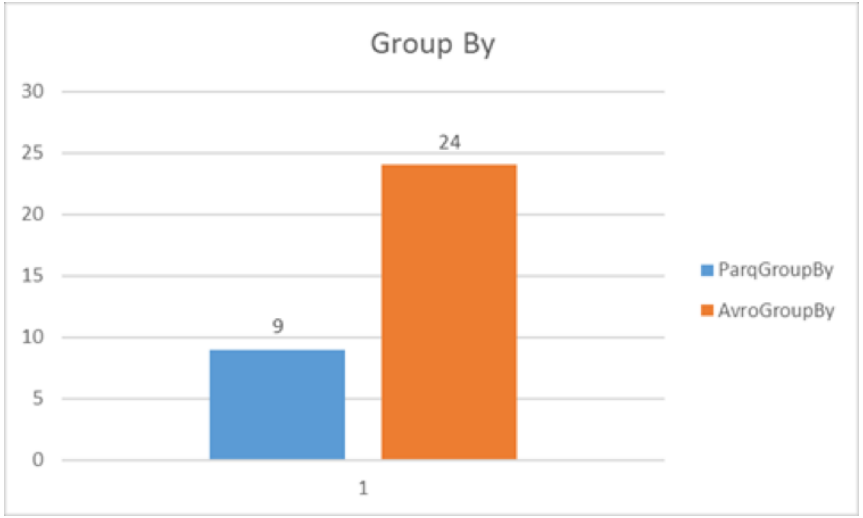
Narrow DS

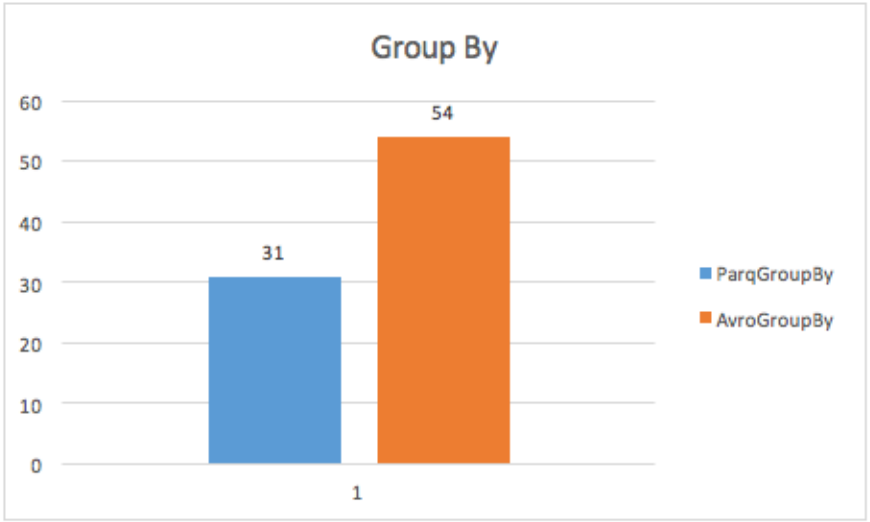


Wide DS

Tercer test

- Group by



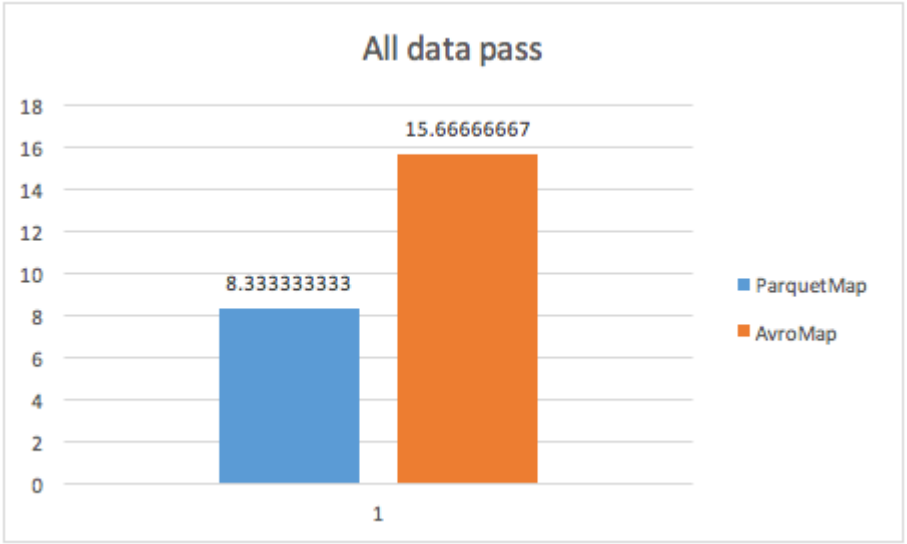


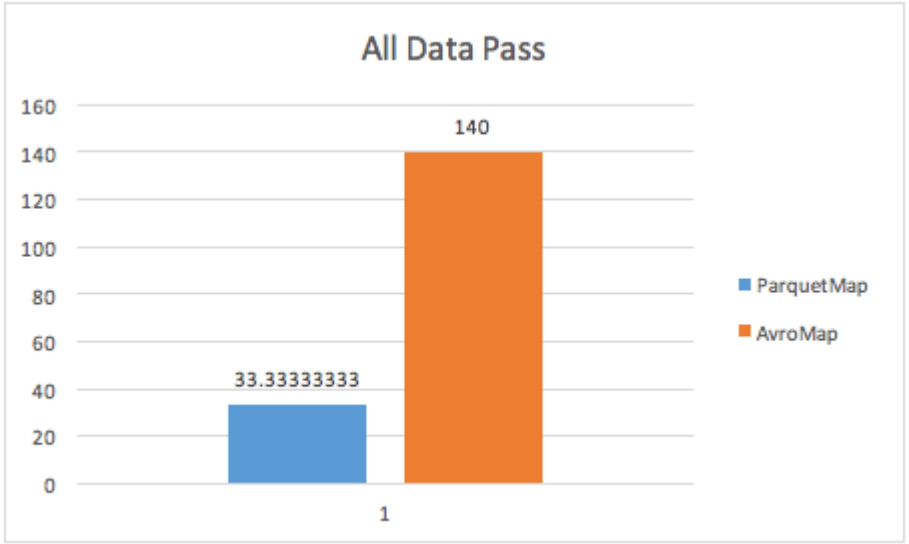
Narrow DS

Wide DS

Cuarto test

- Procesar todos los datos para contar el número de columnas de cada fila



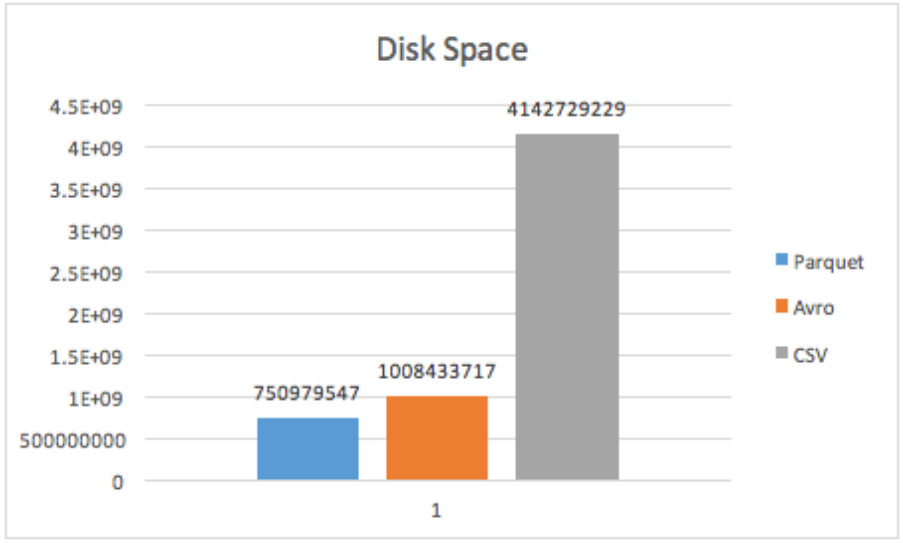


Narrow DS

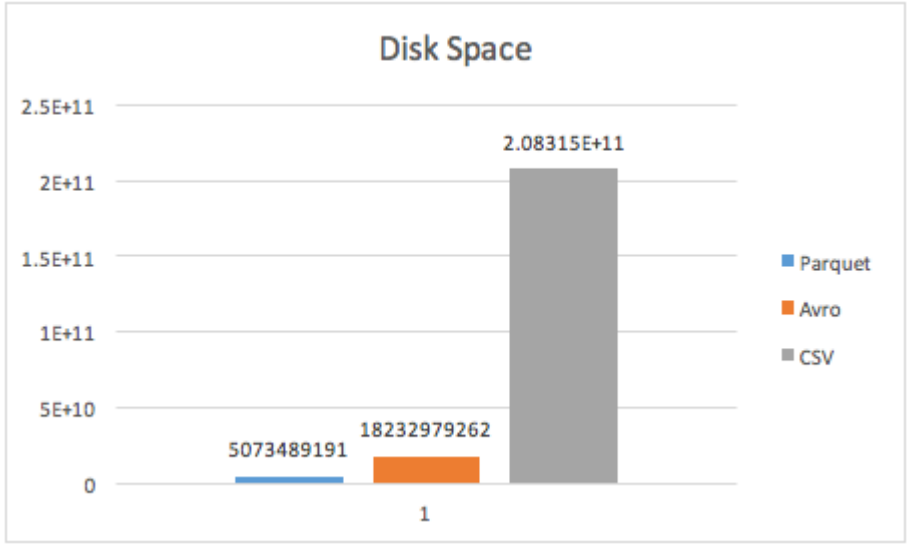
Wide DS

Quinto test

- Espacio en disco



Narrow DS



Wide DS

¿Cómo cargar y guardar datos? => load y save

- Para cargar datos en un DataFrame necesitamos un objeto *DataFrameReader* accesible a través del campo **read** de una sesión Spark
- El método load (del objeto *DataFrameReader)* carga datos de una fuente de datos a un DataFrame

```
df = spark.read.load("users.parquet")
df.select("name", "favorite_color").write.save("namesAndFavColors.parquet")
```

- Para guardar datos de un DataFrame necesitamos un objeto *DataFrameWriter* accesible a través del campo **write** de una sesión Spark
- El método save (del objeto *DataFrameWriter)* guarda el contenido de un DataFrame a la fuente de datos que se indique

¿Cómo cargar y guardar datos? => load y save (II)

- Si no se especifica formato => parquet por defecto
- Se puede especificar el formato manualmente (json, parquet, orc)

```
df = spark.read.load("gente.json", format="json")
df.select("name", "age").write.save("nombresyedades.parquet", format="parquet")
```

- En vez de load (save) pueden usarse los métodos json, orc y parquet para no tener que especificar formato

Ficheros parquet: EJEMPLO

```
peopleDF = spark.read.json("gente.json")

# DataFrames can be saved as Parquet files, maintaining the schema information.
peopleDF.write.parquet("people.parquet")

# Read in the Parquet file created above.
# Parquet files are self-describing so the schema is preserved.
# The result of loading a parquet file is also a DataFrame.
parquetFile = spark.read.parquet("people.parquet")

# Parquet files can also be used to create a temporary view and then used in SQL statements.
parquetFile.createOrReplaceTempView("parquetFile")
teenagers = spark.sql("SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19")
teenagers.show()
```

```
+------+
|  name|
+------+
|Justin|
+------+
```