Storage Format and Compression

Big Data File Format

What is Storage Format?

- File encoding into bytes store on disk
- HDFS allows you to store any type of file formats
  - ANY binary file
  - ANT text file

Why Storage Format important for Big Data

- Efficiency gain in storage or processing is significant factor at Big Data scale
- Considerations for choosing the file format
  - Speed of writing
  - Speed of reading
  - Compression support
  - Schema support
  - Splitability
  - Size of file
  - Support for nested data structures such as map, list, struct
  - *Will your application support?*

Storage Formats

- Traditional formats
  - Unstructured text - body of email, tweet messages
  - Semi formatted text - csv, tsv, xml, json, regex parsable formats like sys logs
- Advanced formats
  - Sequence file
  - Avro
  - ORC (columnar format)
  - Parquet (columnar format)

Example

Format	Size on Disk
Uncompressed CSV	1.8 GB
Avro	1.5 GB
Avro w/ Snappy Compression	750 MB
Parquet w/ Snappy Compression	300 MB

CSV/TSV

- Human readable format
- Splittable but does not support block compression
- Easy to work with while exchanging data with systems outside Hadoop
- Fast writing

Json/XML

- Avoid storing data in json or xml as they increase data size due to tags
- Json/xml are non splittable, so avoid them for large files
- But if individual records are fully contained json/xml doc, the file remains splittable, so you can use it.

Avro

- Binary format - splittable, and support block compression
- Stores schema within file
- Schema can be extracted, supplied and used for validation
  - Schema manipulation can be extracted using avro-tools
- Separate schema file allows flexible schema evolution
  - Rename, add, delete fields, change data type
- Useful as message exchange format
  - Commonly used format for messages in Apache Kafka

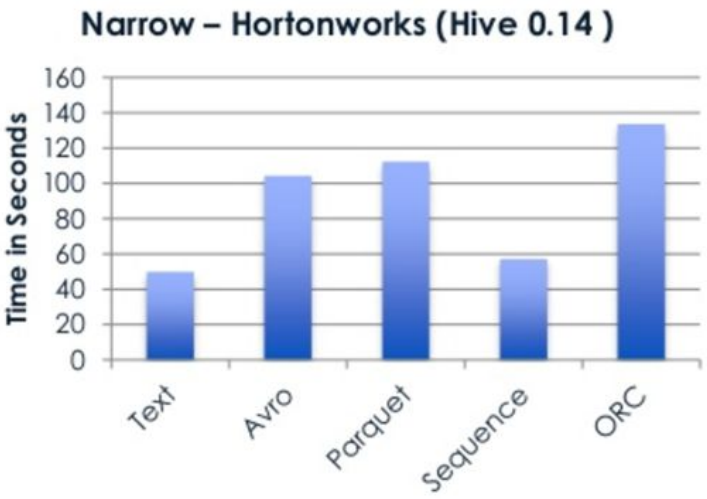
Sequence File

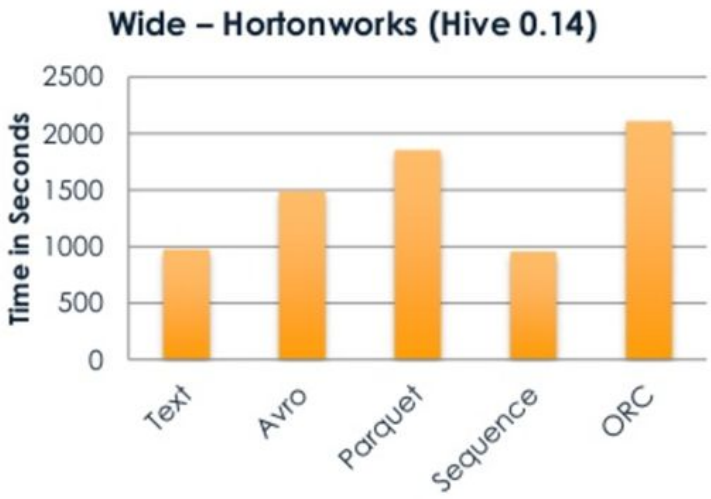
- Binary format, splittable and support block compression
- Stores data in typed key - value format
- Schema evolution is limited
- Native format for Hadoop map reduce framework
- Commonly used to save intermediate data between two MR jobs

ORC/Parquet

- Binary, columnar format
- Splittable, support block compression
- Compress files up to 88%
- Support partial reading (aka query pushdown) thus processing is fast
- Latest version support schema evolution
- Parquet is the default file format for Apache Spark
- Slower writing speed compared to text, avro formats
- Commonly used for data storage for interactive/iterative data processing

Benchmark - Write (Hortonworks)

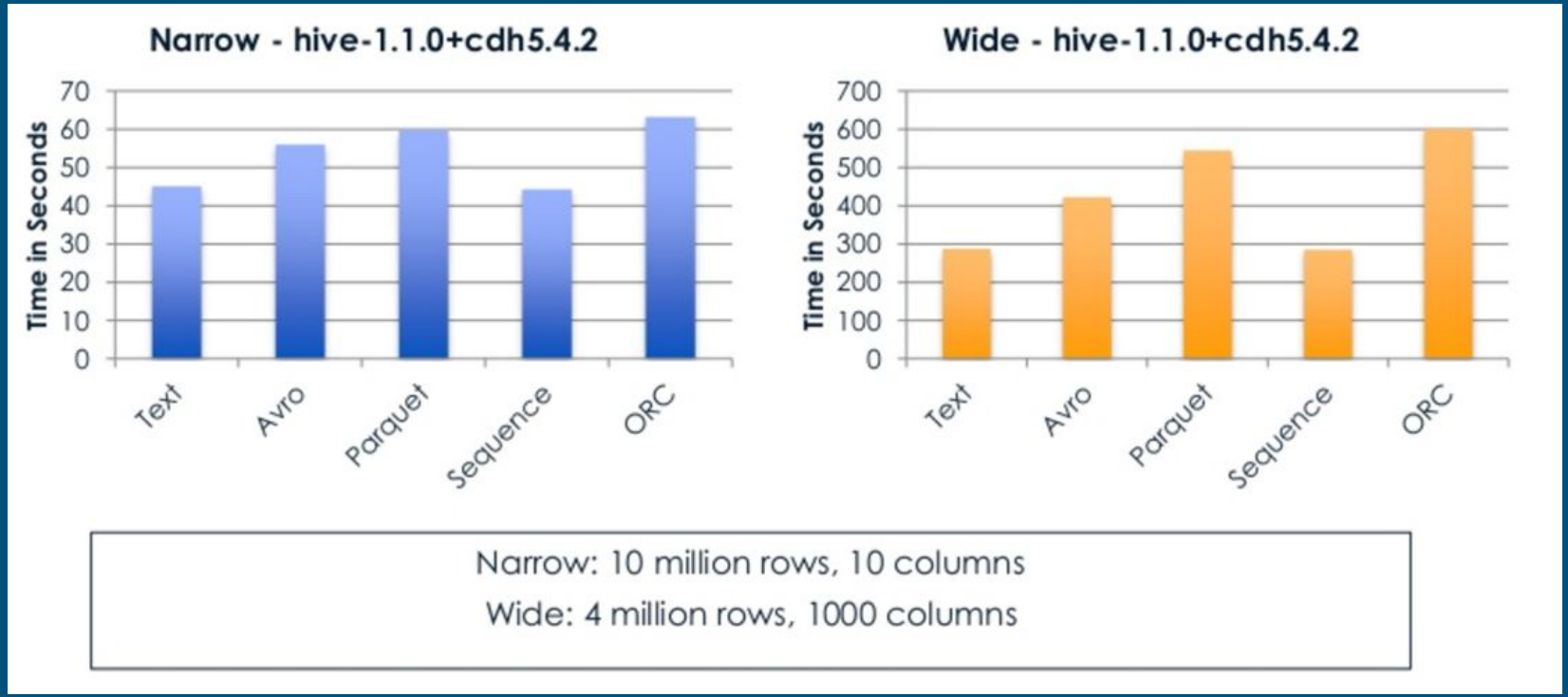




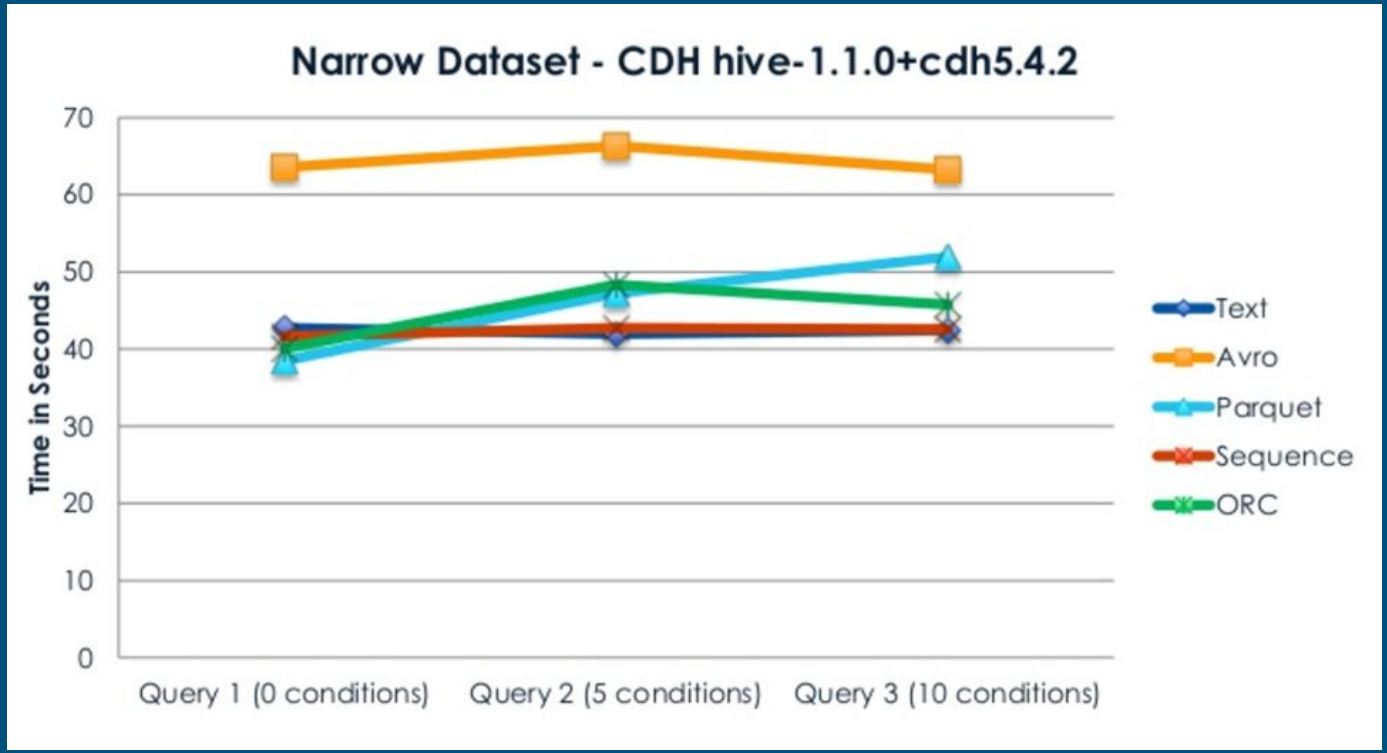
Narrow: 10 million rows, 10 columns

Wide: 4 million rows, 1000 columns

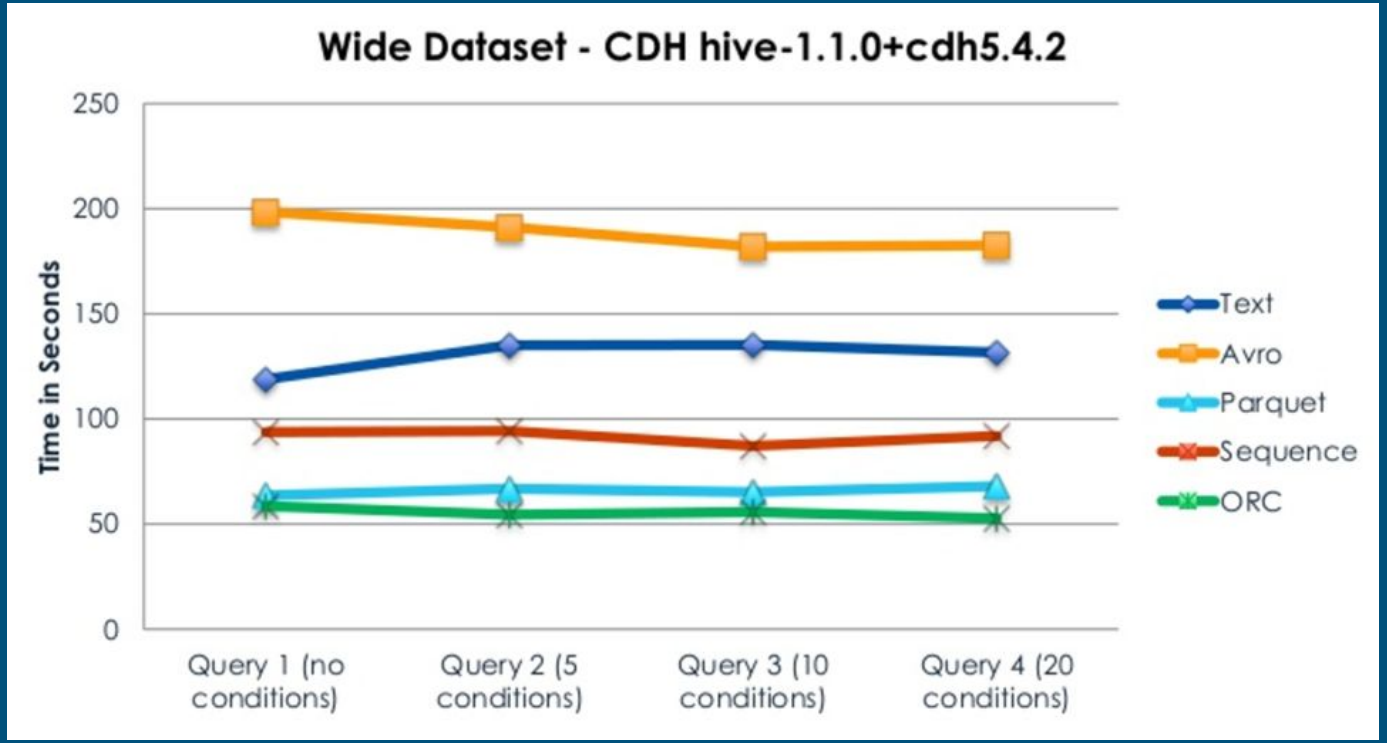
Benchmark - Write (Cloudera)



Benchmark - Read (Narrow Dataset)



Benchmark - Read (Wide Dataset)



Schema Evolution

- What is schema evolution
  - Add, delete columns, change data type
- Schema evolution can be (very) expensive
- Formats that support schema evolution
  - Avro - best!
  - Parquet
  - ORC

Compression

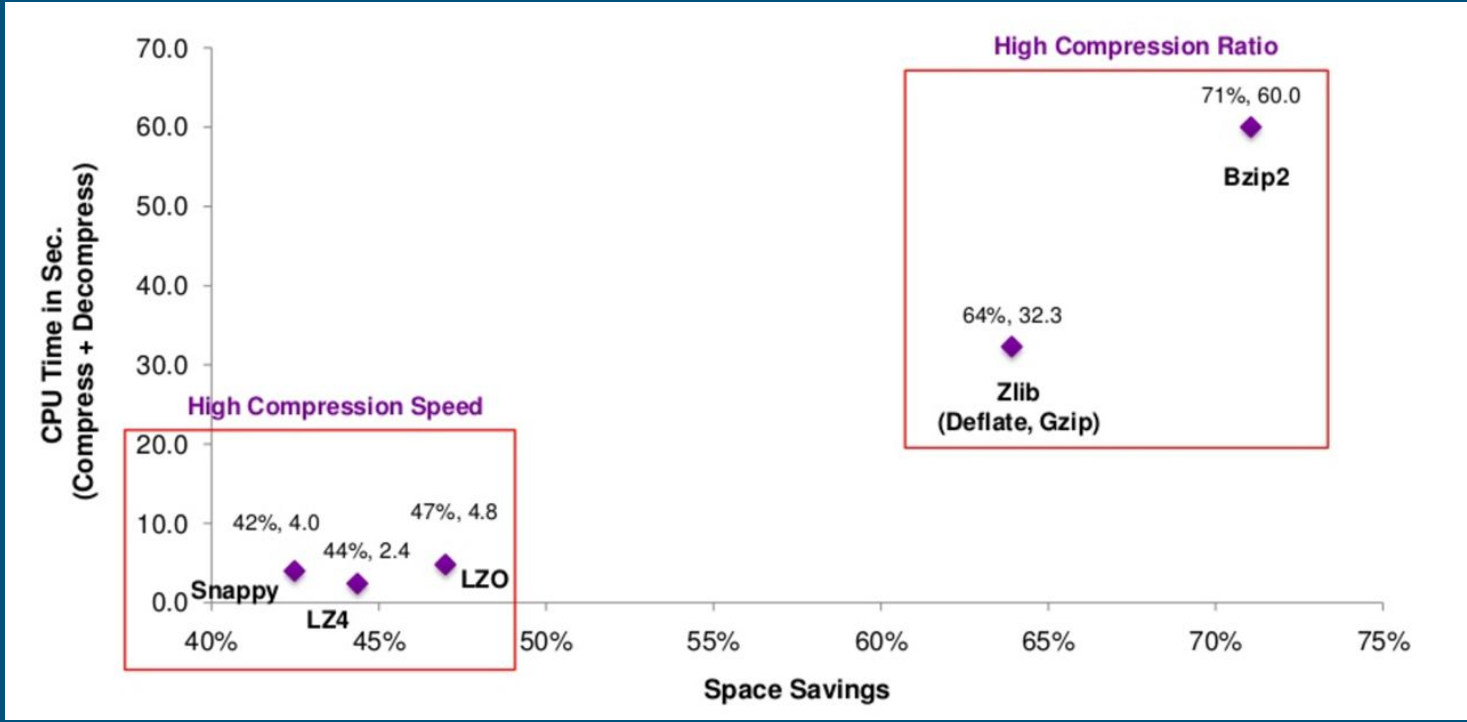
Compression

- Compression save DISK and IO space at expense of CPU
- Consideration for choosing the right compression codec
  - Compression Ratio
  - Speed of compression
  - Speed of decompression
  - Splitability - record or block level or file level. Splitability increases parallelism
  - Interoperability among programming languages and OS

Compression Options

Format	Algorithm	Strategy	Emphasis	Comments
zlib	Uses DEFLATE (LZ77 and Huffman coding)	Dictionary-based, API	Compression ratio	Default codec
gzip	Wrapper around zlib	Dictionary-based, standard compression utility	Same as zlib, codec operates on and produces standard gzip files	For data interchange on and off Hadoop
bzip2	Burrows-Wheeler transform	Transform-based, block-oriented	Higher compression ratios than zlib	Common for Pig
LZO	Variant of LZ77	Dictionary-based, block-oriented, API	High compression speeds	Common for intermediate compression, HBase tables
LZ4	Simplified variant of LZ77	Fast scan, API	Very high compression speeds	Available in newer Hadoop distributions
Snappy	LZ77	Block-oriented, API	Very high compression speeds	Came out of Google, previously known as Zippy

Benchmark - compression codecs



Compression Guidelines

- Compression is generally a good idea since Big Data processing are usually IO bound not CPU bound
- For large files use bzip2 since it supports splitability
- For small files use gzip
- For iterative, interactive workload use fast codec such as snappy
- Choose avro, parquet, orc - container based file formats that supports block level compression

Compression: Hive Example

```
hive> hive.exec.compress.intermediate=true;

hive> mapred.map.output.compression.codec =
org.apache.hadoop.io.compress.SnappyCodec;

hive> set hive.exec.compress.output = true;

hive> set mapred.output.compression.codec =
org.apache.hadoop.io.compress.BZip2Codec;
```

Compression: MR Job

```
$ hadoop jar <.jar file> <class name> \
"-Dmapreduce.output.fileoutputformat.compress=true" \
"-Dmapreduce.output.fileoutputformat.compress.codec=org.apache.hadoop.io.compress
.GzipCodec" \
"-Dmapreduce.map.output.compress=true" \
"-Dmapreduce.map.output.compress.codec=org.apache.hadoop.io.compress.GzipCodec" \
<input path> <output path>
```

Read Compressed Data

```
$ hadoop fs -text /user/hive/warehouse/final_comp_on/* >
data.txt
```

Reference

- Data format benchmarks http://www.svds.com/dataformats/
- Compression Options in Hadoop - A Tale of Tradeoffs http://www.slideshare.net/Hadoop_Summit/kamat-singh-june27425pmroom210cv2
- Data Architecture on Hadoop https://www.safaribooksonline.com/library/view/hadoop-application-architectures/9781491910313/ch01.html