Data Management in Large-Scale Distributed Systems

File formats

Thomas Ropars

thomas.ropars@univ-grenoble-alpes.fr

http://tropars.github.io/

2020

References

- Designing Data-Intensive Applications by Martin Kleppmann
    - Chapters 2: *Column-oriented storage*
    - Chapter 4: *Formats for Encoding Data*

In this lecture

- Representation of large data on disks
  - Data that will be queried for analysis
- Column-oriented file formats

Agenda

Storing data on disks

- The representation of data on disks is in general not the same as in memory
  - Storing a pointer on disk would be meaningless
  - Random accesses on disk can be very slow

- Many file formats exist in the context of *Big Data*
  - CSV
  - JSON
  - Avro
  - Parquet
  - ORC
  - etc.

What are the properties of each file format? Which one to choose?

Challenges

- Try to have a compact representation of the data
  - And organize the data so that they can be efficiently compressed
- Allow modifying the schema and ensure forward/backward compatibility
  - Not covered in this lecture
- Optimize the performance of read operations
  - Write once, read many

Agenda

Textual formats

Examples of such formats

- CSV
- JSON
- XML

Advantages

- Readable by humans

Drawbacks

- High storage footprint
- Very low read performance

Textual formats

CSV

- Comma Separated Values
- Good for storing data organized as a single table
  - The name of the columns is given by the first row (not verbose)
- No hierarchical structure

JSON – XML

- Support for hierarchical structures
- Very verbose (large footprint)

Binary encoding formats

Examples

- Avro (Hadoop)
- Thrift (Facebook)
- Protocol Buffers (Google)

Idea

- Describe the data using a schema
- Pack all fields describing an item (a row) in a binary format

Advantages

- Can lead to huge space reduction

Example

By M. Kleppmann

```
{
  "userName": "Martin",
  "favoriteNumber": 1337,
  "interests": ["daydreaming", "hacking"]
}
```

Figure: A JSON document

Storage space

- If stored as JSON text file: 81 bytes
- If stored as simple binary JSON encoding (not using a schema): 66 bytes
  - Space saved on the representation of numbers and on structure information

With Avro

```
{
  "type": "record",
  "name": "Person",
  "fields": [
    {"name": "userName",
      "type": "string"},
    {"name": "favoriteNumber", "type": ["null", "long"], "default": null},
    {"name": "interests",
      "type": {"type": "array", "items": "string"}}
  ]
}
```

Figure: Definition of the schema

With Avro

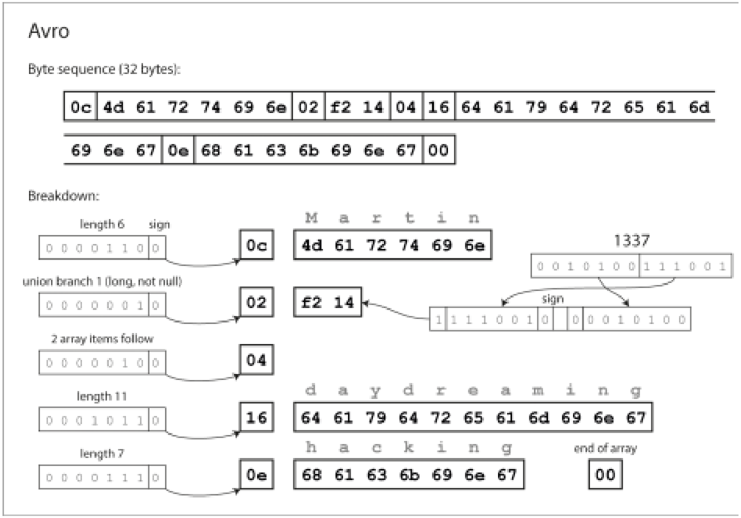

Figure: Binary representation of the item (32 bytes)

Agenda

Row-oriented formats

All formats described until now are row-oriented

- All the values from one row of a table are stored next to each other

Limitations

Row-oriented formats

All formats described until now are row-oriented

- All the values from one row of a table are stored next to each other

Limitations

- Inefficient data compression
  - Data of different types are next to each other
- Inefficient read operations
  - We are often only interested in a few entries in a row
    - But we have to read the full row
  - We may want to filter elements based on a condition on one entry
    - But we have to read all the rows

Column-oriented formats

Examples

- Parquet (Twitter + Cloudera)
- ORC (Hadoop)

Description

- Stores all the values from each column together
- Efficient compression
  - Close values are of the same type (e.g., integers)
  - The number of distinct values in a column is often small (not the case for rows)
- Optimizations on read:
  - Projection push-down
  - Predicate push-down

Optimizations on read

Projection push-down

- We are interested in a subset of columns
- We can read only the files corresponding to these columns
    - Or the file chunks storing these columns

Predicate push-down

- We are interested in items corresponding to a condition
    - `SELECT * FROM Customers WHERE Country='Mexico'`
- Check the condition by reading only the corresponding column

Parquet data layout

- Data are stored in files
- A file consists of one or more row groups
  - A set of rows
- A row group contains exactly one column chunk per column
  - A column chunk is contiguous in the file
- Metadata are stored at the end of the file
  - Position of each column chunk
  - Statistics about each chunk
    - Min/Max statistics for numbers
    - Dictionary filtering for other columns (as long as less than 40k different values)
- About sorting
  - Sorting rows based on the filtering criteria that is used the more often for filtering can improve performance

Example

source: https://blog.usejournal.com/sorting-and-parquet-3a382893cde5

Description of the data

- Customer table with one column being the country
- `SELECT * FROM Customers WHERE Country='Mexico'`
- Some numbers:
  - 10M rows
  - 10k rows per row group
  - 1% of the customers are from Mexico

Amount of data read to answer the query

- With a row-based format:
- With an unsorted parquet file:
- With a sorted parquet file:

Example

source: `https://blog.usejournal.com/sorting-and-parquet-3a382893cde5`

Description of the data

- Customer table with one column being the country
- `SELECT * FROM Customers WHERE Country='Mexico'`
- Some numbers:
  - 10M rows
  - 10k rows per row group
  - 1% of the customers are from Mexico

Amount of data read to answer the query

- With a row-based format: All data
- With an unsorted parquet file:
- With a sorted parquet file:

Example

source: `https://blog.usejournal.com/sorting-and-parquet-3a382893cde5`

Description of the data

- Customer table with one column being the country
- `SELECT * FROM Customers WHERE Country='Mexico'`
- Some numbers:
  - 10M rows
  - 10k rows per row group
  - 1% of the customers are from Mexico

Amount of data read to answer the query

- With a row-based format: All data
- With an unsorted parquet file:
  - Probability of a row group with no customer from Mexico: $0.99^{10000} = 2.25 \times 10^{-44}$
  - All row groups
- With a sorted parquet file:

Example

source: `https://blog.usejournal.com/sorting-and-parquet-3a382893cde5`

Description of the data

- Customer table with one column being the country
- `SELECT * FROM Customers WHERE Country='Mexico'`
- Some numbers:
  - 10M rows
  - 10k rows per row group
  - 1% of the customers are from Mexico

Amount of data read to answer the query

- With a row-based format: All data
- With an unsorted parquet file:
  - Probability of a row group with no customer from Mexico: $0.99^{10000} = 2.25 \times 10^{-44}$
  - All row groups
- With a sorted parquet file: 1% of the row groups (10)

Another example

source: `https://www.linkedin.com/pulse/we-taking-only-half-advantage-columnar-file-format-eric-sun/`

Description of the data

- A website log dataset
  - Information in one entry: `timestamp, user_id, cookie, page_id, http_header, ...`
- Queries filters against `page_id`

Some results

- Avro:
  - Compressed data footprint: 1.4 TB
  - Amount of data read on query: 1.4 TB
- ORC[1] unsorted/sorted:
  - Compressed data footprint: 0.9 TB / 0.5 TB
  - Amount of data read on query: 300 GB / 200 MB

[1] similar to parquet

Additional references

Suggested reading

- *Dremel: Interactive Analysis of Web-Scale Datasets.*, S. Melnik et al., VLDB, 2010.