Parquet

An open columnar file format for Hadoop

Julien Le Dem @J_

Processing tools lead, analytics infrastructure

Twitter

http://parquet.io

Context

- **Twitter's data**
  - 200M+ monthly active users generating and consuming 400M+ tweets a day.
  - Scale is huge: Instrumentation, User graph, Derived data, ...

- **Analytics infrastructure**:
  - Several 1K+ nodes Hadoop clusters
  - Log collection pipeline
  - Processing tools

- **Role of Twitter's analytics infrastructure team**
  - Platform for the whole company.
  - Manages the data and enables analysis.
  - Optimizes the cluster's workload as a whole.

Twitter’s use case

- Logs available on HDFS
- Thrift to store logs
- example: one schema has 87 columns, up to 7 levels of nesting.

```
struct LogEvent {
  1: optional logbase.LogBase log_base
  2: optional i64 event_value
  3: optional string context
  4: optional string referring_event
...
  18: optional EventNamespace event_namespace
  19: optional list<Item> items
  20: optional map<AssociationType,Association> associations
  21: optional MobileDetails mobile_details
  22: optional WidgetDetails widget_details
  23: optional map<ExternalService,string> external_ids
}
```

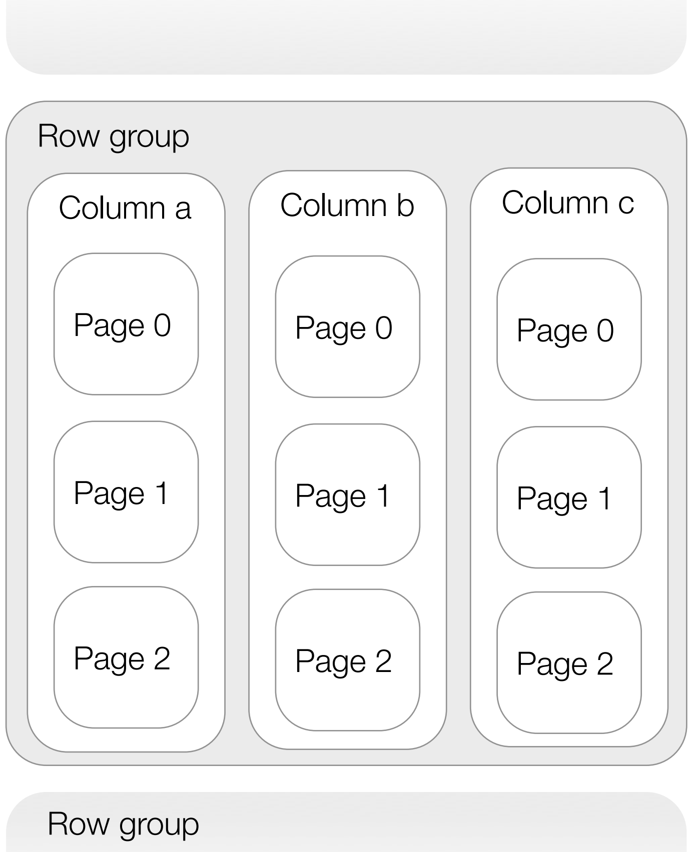
```
struct LogBase {
  1: string transaction_id,
  2: string ip_address,
  ...
  15: optional string country,
  16: optional string pid,
}
```

Parquet

- **Columnar Storage**
  - Saves space: columnar layout compresses better
  - Enables better scans: load only the columns that need to be accessed
  - Enables Dremel-like execution engines
- **Collaboration with Cloudera:**
  - Common file format definition: Language independent, formally specified.
  - Implementation in Java for Map/Reduce: https://github.com/Parquet/parquet-mr
  - C++ code generation in Cloudera Impala: https://github.com/cloudera/impala

http://parquet.io

Format

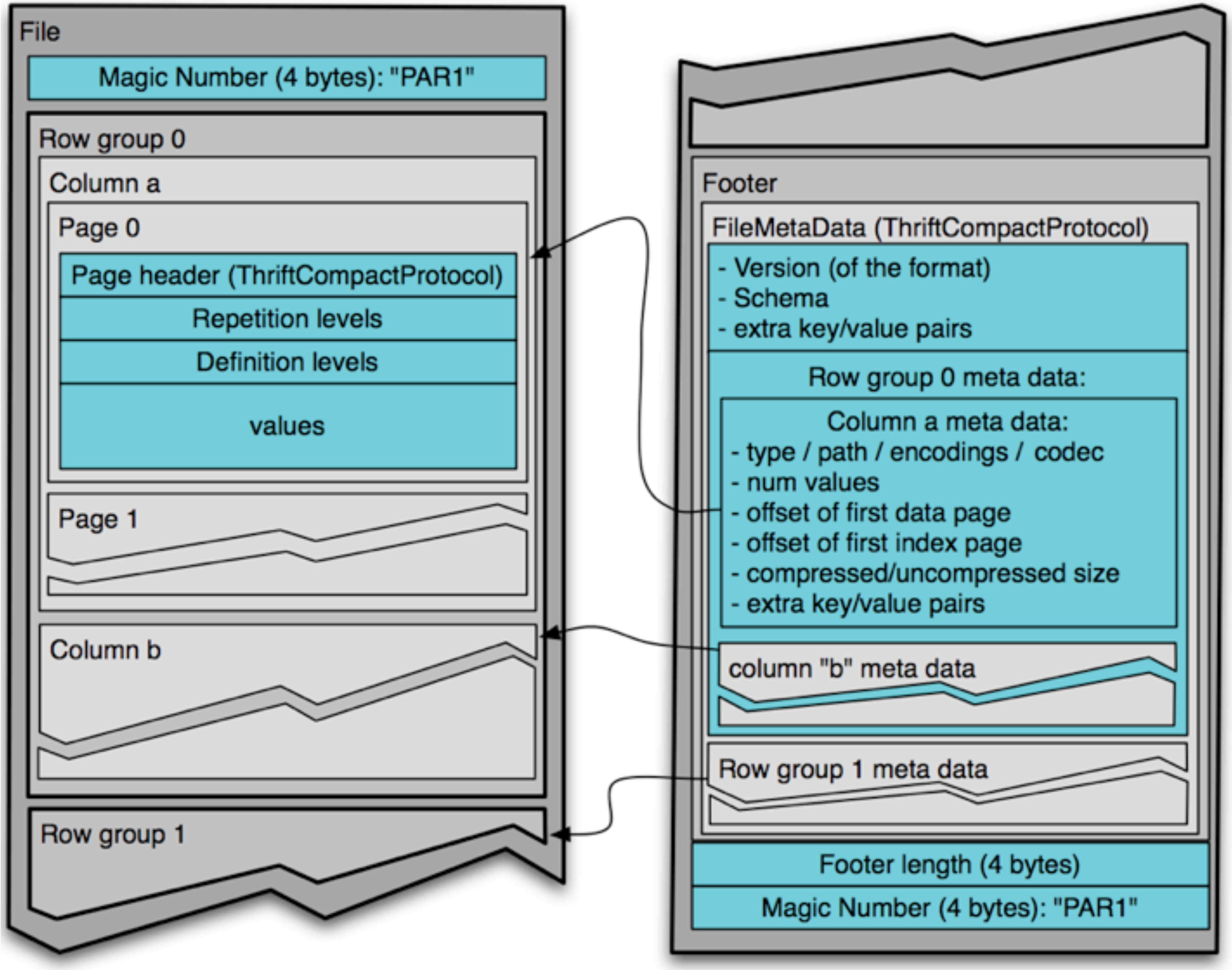


- **Row group:** A group of rows in columnar format.
  - Max size buffered in memory while writing.
  - One (or more) per split while reading.
  - roughly: 10MB < row group < 1 GB

- **Column chunk:** The data for one column in a row group.
  - Column chunks can be read independently for efficient scans.

- **Page:** Unit of compression in a column chunk.
  - Should be big enough for compression to be efficient.
  - Minimum size to read to access a single record (when index pages are available).
  - roughly: 8KB < page < 100KB

Format


File
Magic Number (4 bytes): "PAR1"
Row group 0
Column a
Page 0
Page header (ThriftCompactProtocol)
Repetition levels
Definition levels
values
Page 1
Column b
Row group 1
Footer
FileMetaData (ThriftCompactProtocol)
- Version (of the format)
- Schema
- extra key/value pairs
Row group 0 meta data:
Column a meta data:
- type / path / encodings / codec
- num values
- offset of first data page
- offset of first index page
- compressed/uncompressed size
- extra key/value pairs
column "b" meta data
Row group 1 meta data
Footer length (4 bytes)
Magic Number (4 bytes): "PAR1"

Layout: Row groups in columnar format. A footer contains column chunks, offset and schema.

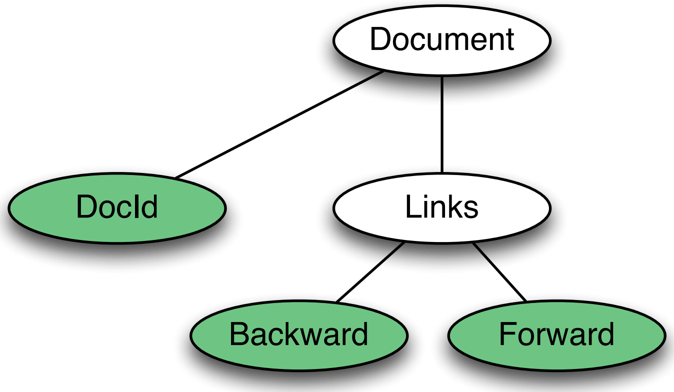
Language independent: Well defined format. Hadoop and Cloudera Impala support.

Dremel’s shredding/assembly

- Each cell is encoded as a triplet: **repetition level, definition level, value**.
- Level values are bound by the depth of the schema: **stored in a compact form.**

Schema:

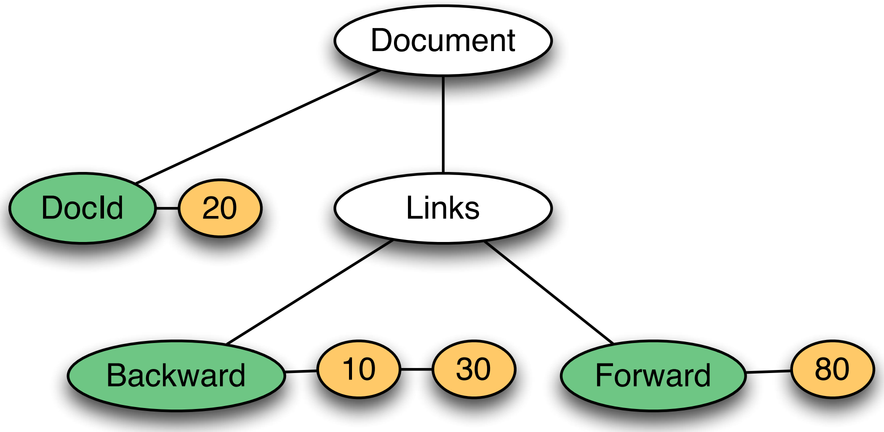
```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward;
  }
}
```



Columns	Max rep. level	Max def. level
`DocId`	0	0
`Links.Backward`	1	2
`Links.Forward`	1	2

Record:

```
DocId: 20
Links
  Backward: 10
  Backward: 30
  Forward:  80
```



Column	value	R	D
`DocId`	20	0	0
`Links.Backward`	10	0	2
`Links.Backward`	30	1	2
`Links.Forward`	80	0	2

Reference: http://research.google.com/pubs/pub36632.html

APIs

- **Iteration on columns:**
  - Iteration on triplets: repetition level, definition level, value.
  - Repetition level = 0 indicates a new record.
  - encoded or decoded values: computing aggregations on integers is faster than strings.

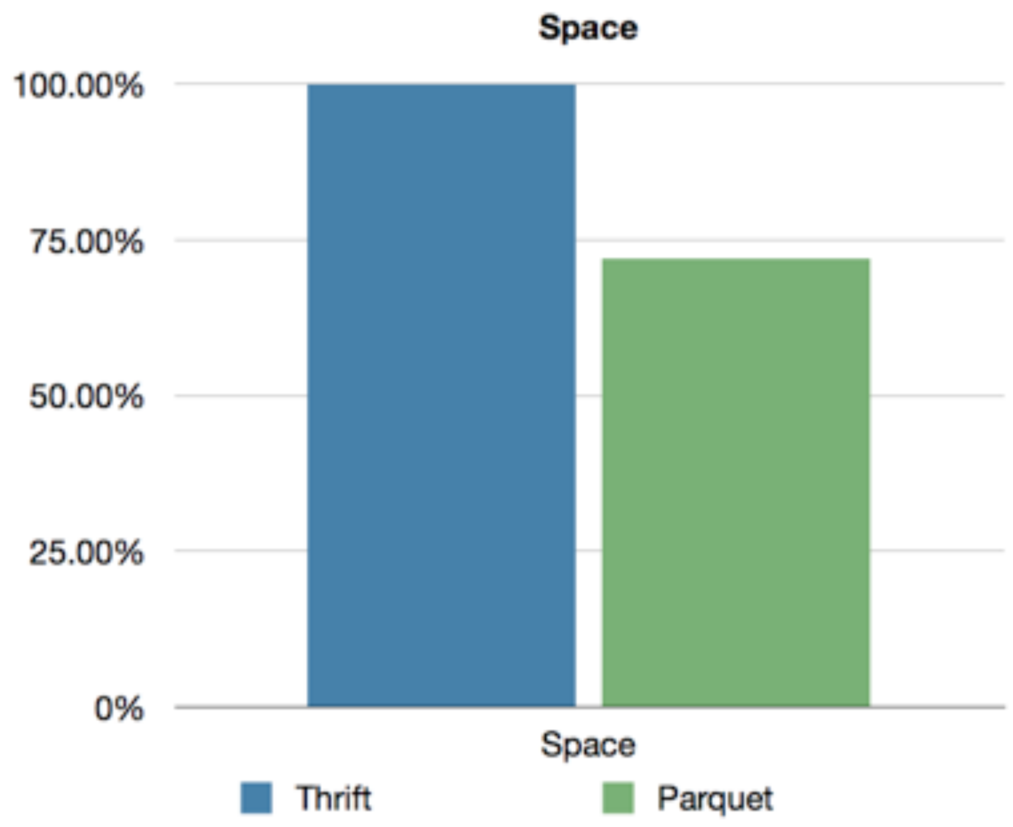
- **Iteration on fully assembled records:**
  - Assembles projection for any subset of the columns: only those are loaded from disc.

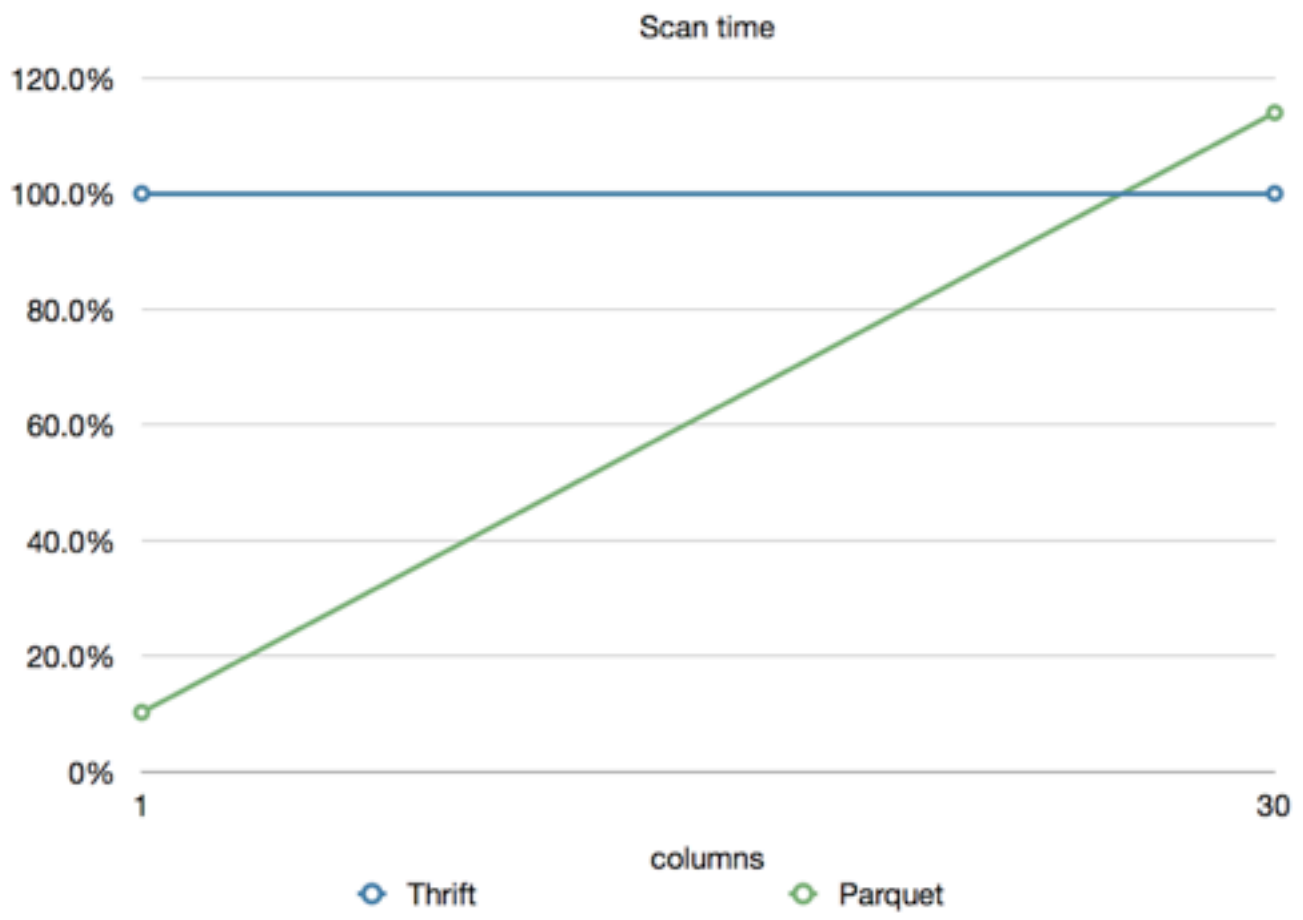
- **Schema definition and record materialization:**
  - Hadoop does not have a notion of schema, however Pig, Hive, Thrift, Avro, ProtoBufs do.
  - Event-based SAX-style record materialization layer. No double conversion.

Initial results

Data converted: similar to access logs

Original format: Thrift binary in block compressed files





Space saving: 28% using the same compression algorithm

Scan + assembly time compared to original:

One column: 10%

All columns: 114%

http://parquet.io

Where do we go from here?

- **Bring the techniques from parallel DBMSs to Hadoop:**
    - Hadoop is very reliable for big long running queries but also IO heavy.
    - Enable Dremel-style execution engines.
    - Incrementally take advantage of column based storage in our stack.

- **Future:**
    - Indices.
    - Better encodings.
    - Better execution engines.

http://parquet.io

Where do ***you*** go from here?

Questions? Ideas?

Contribute at: github.com/Parquet

@JoinTheFlock

http://parquet.io