Strata CONFERENCE + HADOOP WORLD

Oct. 28–30, 2013
NEW YORK, NY

#strataconf + #hw2013

Co-presented by O'REILLY cloudera

Parquet

Columnar storage for the people

Julien Le Dem @J_ Processing tools lead, analytics infrastructure at Twitter

Nong Li nong@cloudera.com Software engineer, Cloudera Impala

http://parquet.io

Outline

- Context from various companies
- Results in production and benchmarks
- Format deep-dive

Twitter Context

- **Twitter's data**
  - 230M+ monthly active users generating and consuming 500M+ tweets a day.
  - 100TB+ a day of compressed data
  - Scale is huge:
    - Instrumentation, User graph, Derived data, ...

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

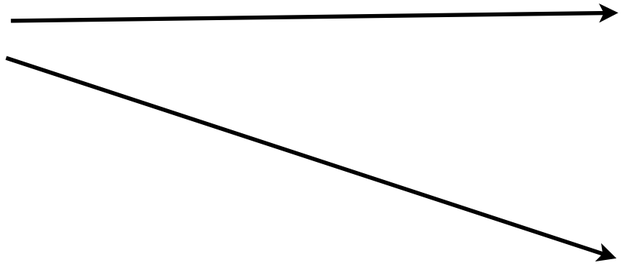
The Parquet Planers
Gustave Caillebotte

http://parquet.io

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,
}
```

Goal

To have a state of the art columnar storage available across the Hadoop platform

- Hadoop is very reliable for big long running queries but also IO heavy.
- Incrementally take advantage of column based storage in existing framework.
- Not tied to any framework in particular

Columnar Storage

- Limits IO to data actually needed:
  - loads only the columns that need to be accessed.
- Saves space:
  - Columnar layout compresses better
  - Type specific encodings.
- Enables vectorized execution engines.

@EmrgencyKittens

http://parquet.io

Collaboration between Twitter and Cloudera:

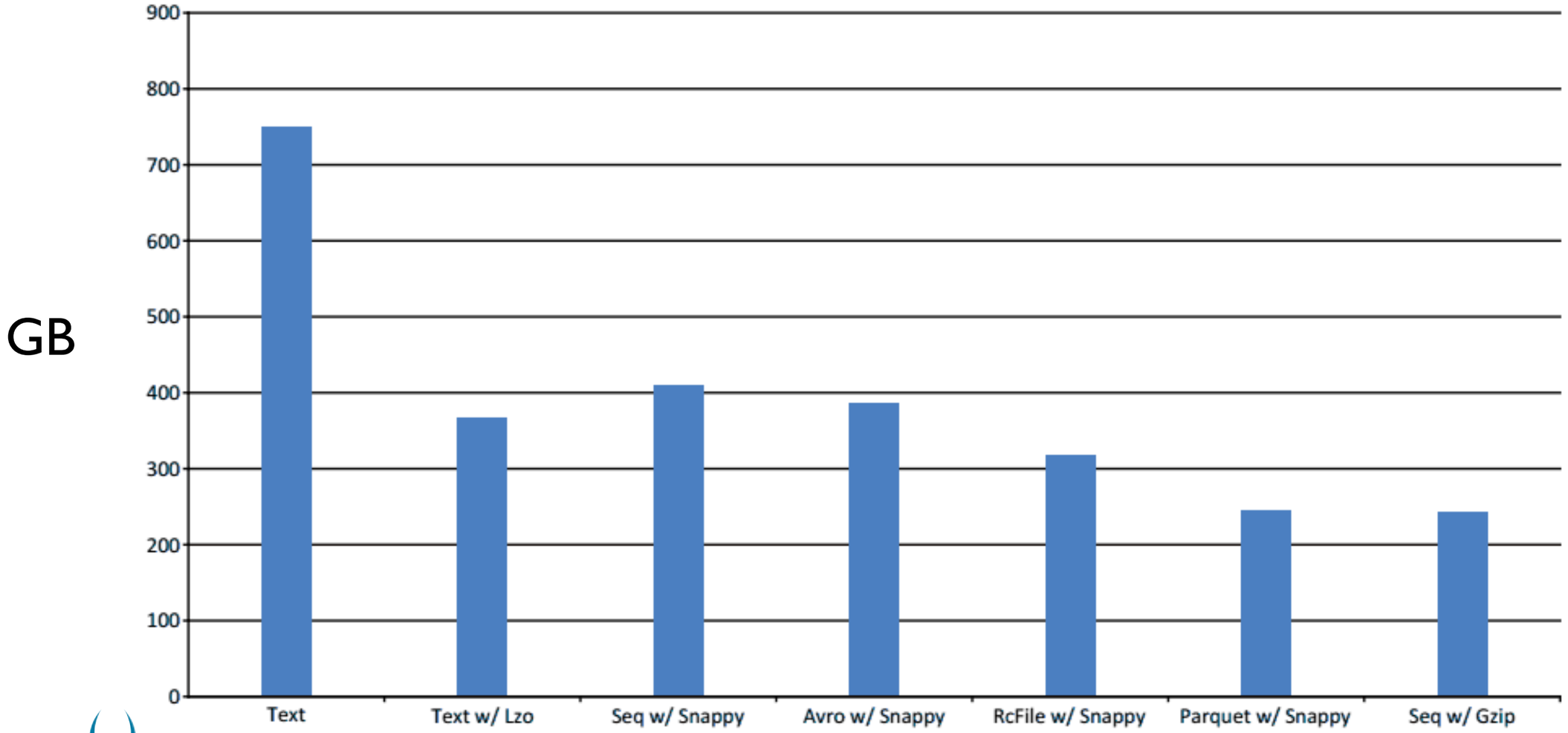
- Common file format definition:
  - Language independent
  - Formally specified.

- Implementation in Java for Map/Reduce:
  - https://github.com/Parquet/parquet-mr

- C++ and code generation in Cloudera Impala:
  - https://github.com/cloudera/impala

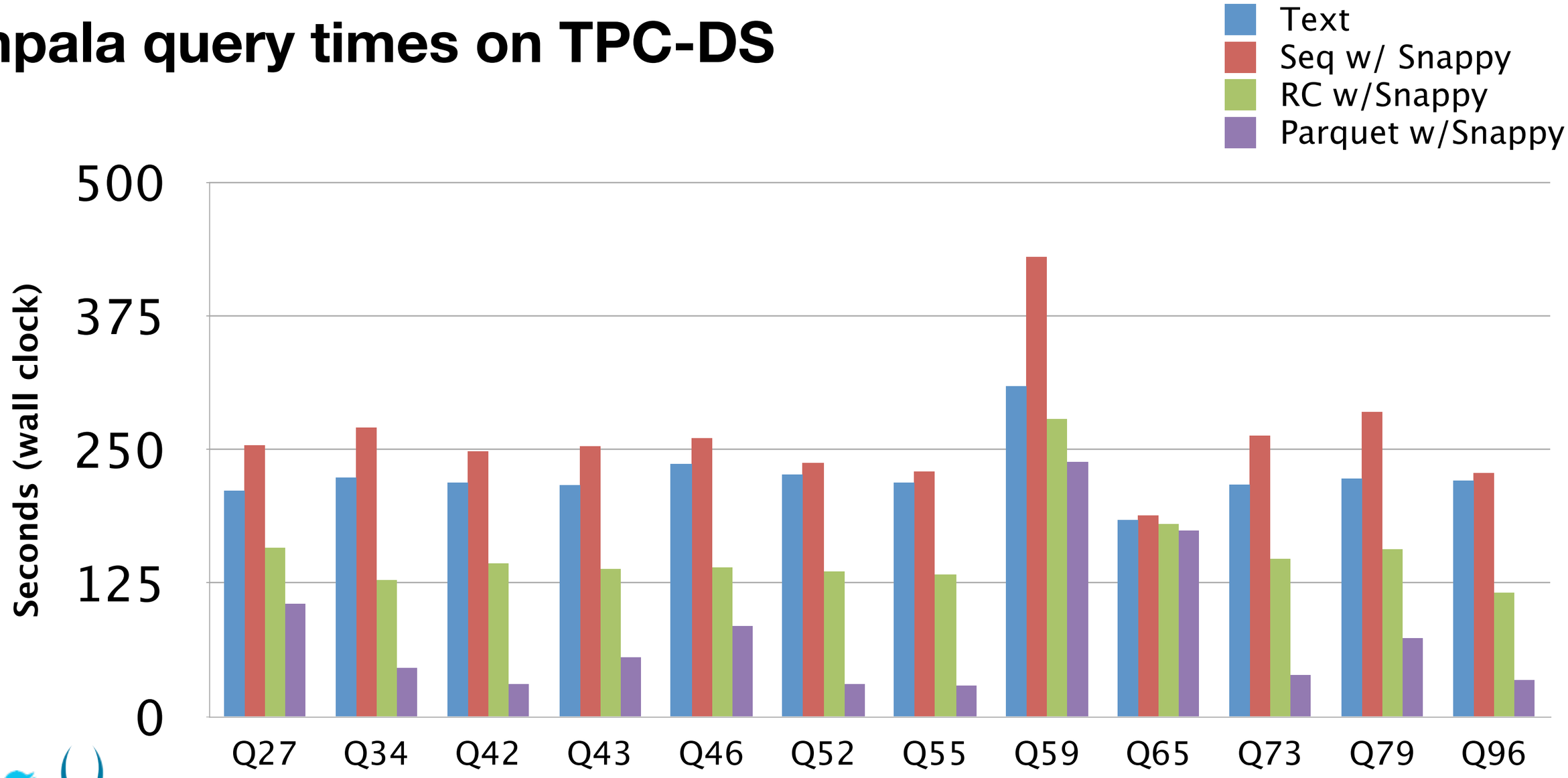
Results in Impala

- TPC-H lineitem table @ 1TB scale factor



http://parquet.io

Impala query times on TPC-DS



http://parquet.io

Criteo: The Context

- Billions of new events per day
- ~60 columns per log
- Heavy analytic workload
- BI analysts using Hive and RCFile
- Frequent schema modifications

- Perfect use case for Parquet + Hive !

Parquet + Hive: Basic Reqs

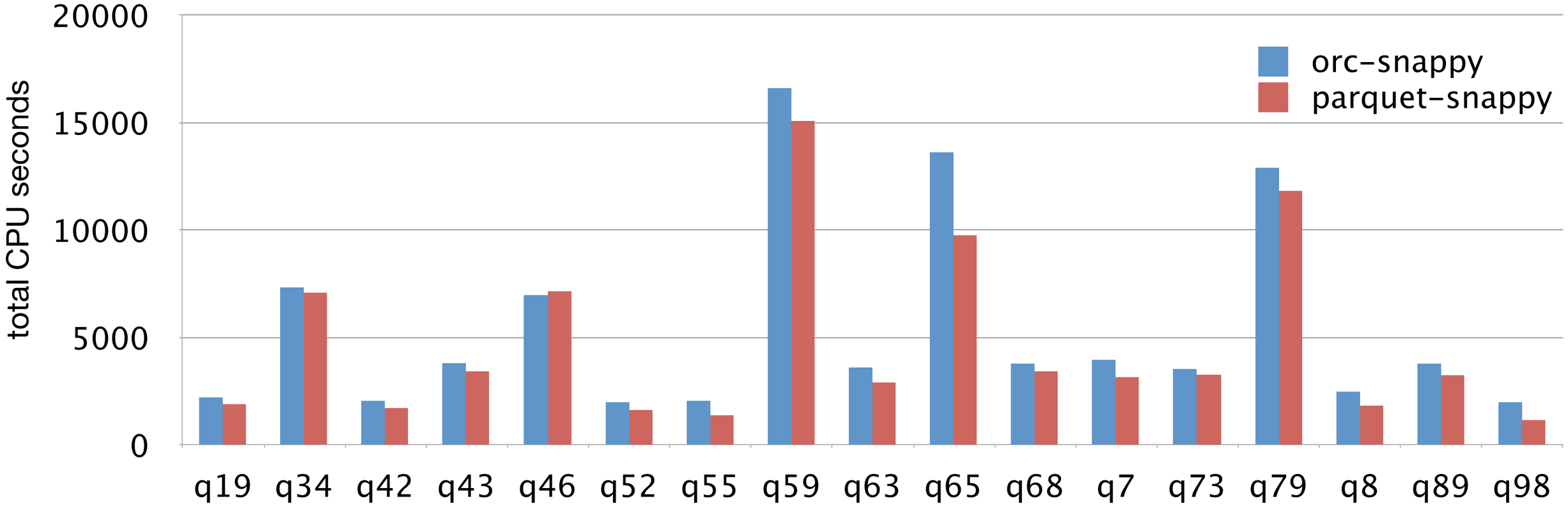
- MapRed compatibility due to Hive.
- Correctly handle evolving schemas across Parquet files.
- Read only the columns used by query to minimize data read.
- Interoperability with other execution engines (eg Pig, Impala, etc.)



Performance of Hive 0.11 with Parquet vs orc

TPC-DS scale factor 100
All jobs calibrated to run ~50 mappers
Nodes:
2 x 6 cores, 96 GB RAM, 14 x 3TB DISK

Size relative to text:
orc-snappy: 35%
parquet-snappy: 33%



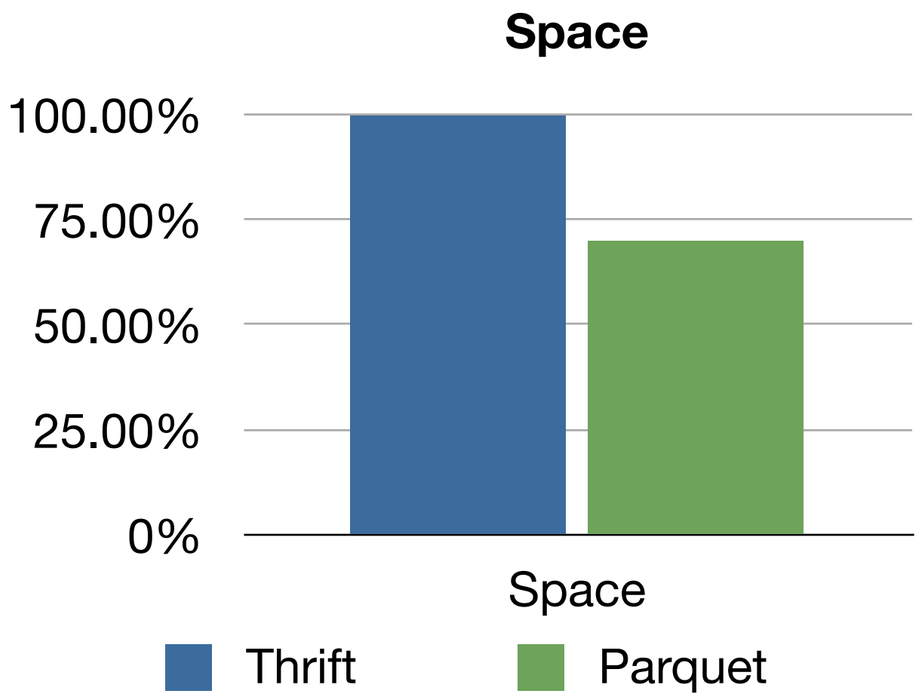
http://parquet.io

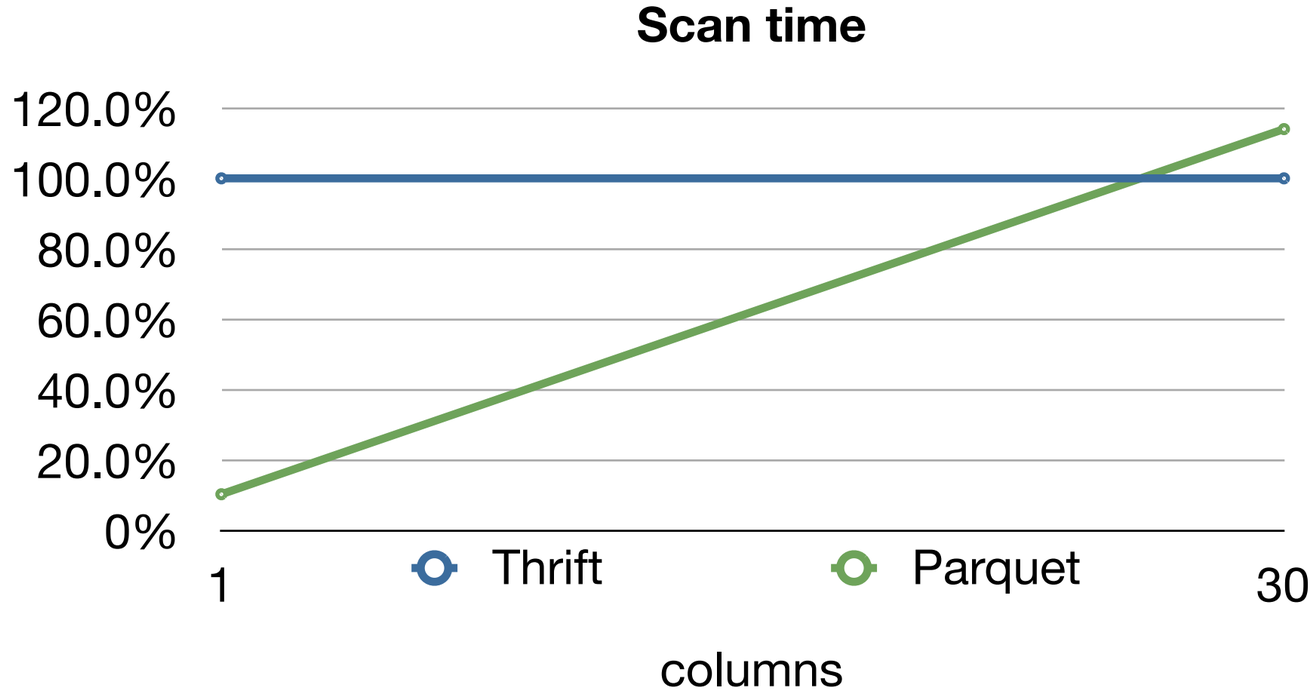
Twitter: production results

Data converted: similar to access logs. 30 columns.

Original format: Thrift binary in block compressed files (LZO)

New format: Parquet (LZO)





- **Space saving:** 30% using the same compression algorithm
- **Scan + assembly time compared to original:**
  - One column: 10%
  - All columns: 110%

http://parquet.io

Production savings at Twitter

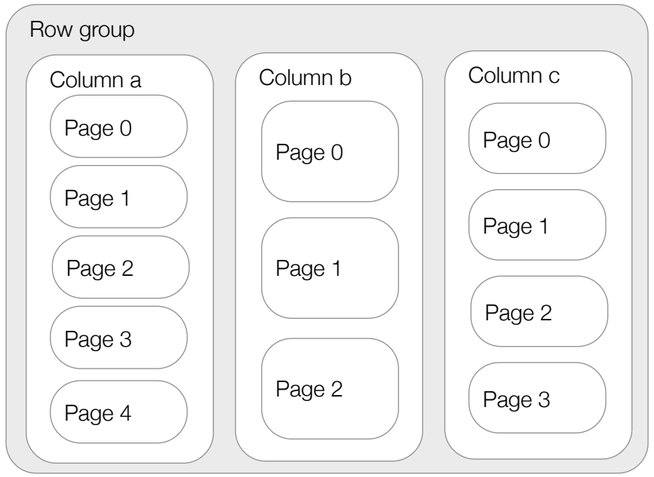
- Petabytes of storage saved.
- Example jobs taking advantage of projection push down:
  - Job 1 (Pig): reading 32% less data => 20% task time saving.
  - Job 2 (Scalding): reading 14 out of 35 columns. reading 80% less data => 66% task time saving.
  - Terabytes of scanning saved every day.

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: 50MB < row group < 1 GB

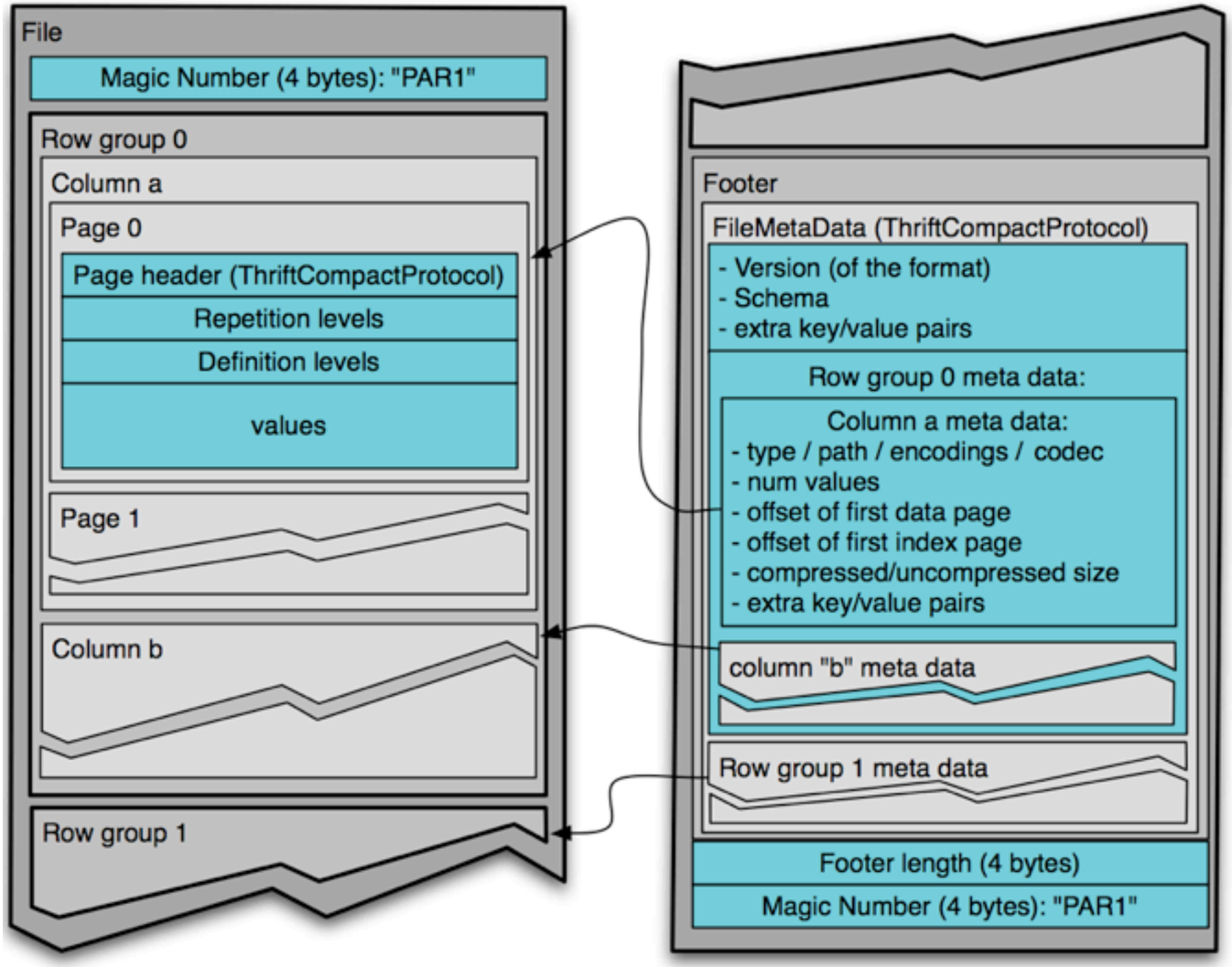
- **Column chunk:** The data for one column in a row group.
  - Column chunks can be read independently for efficient scans.

- **Page:** Unit of access 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 < 1MB



http://parquet.io

Format



- **Layout:**

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

- **Language independent:**

  Well defined format. Hadoop and Cloudera Impala support.

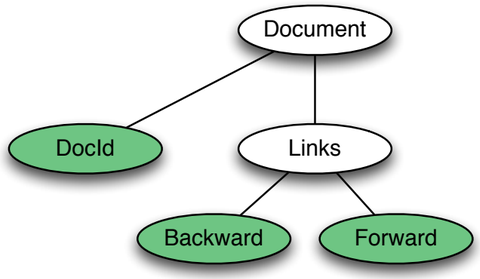
http://parquet.io

Nested record shredding/assembly

- Algorithm borrowed from Google Dremel's column IO
- 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:

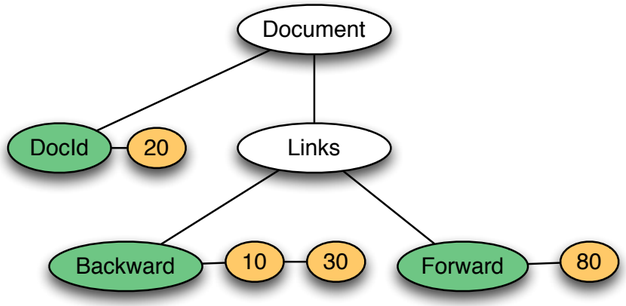
```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward;
  }
}
```



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

http://parquet.io

Repetition level

Schema:

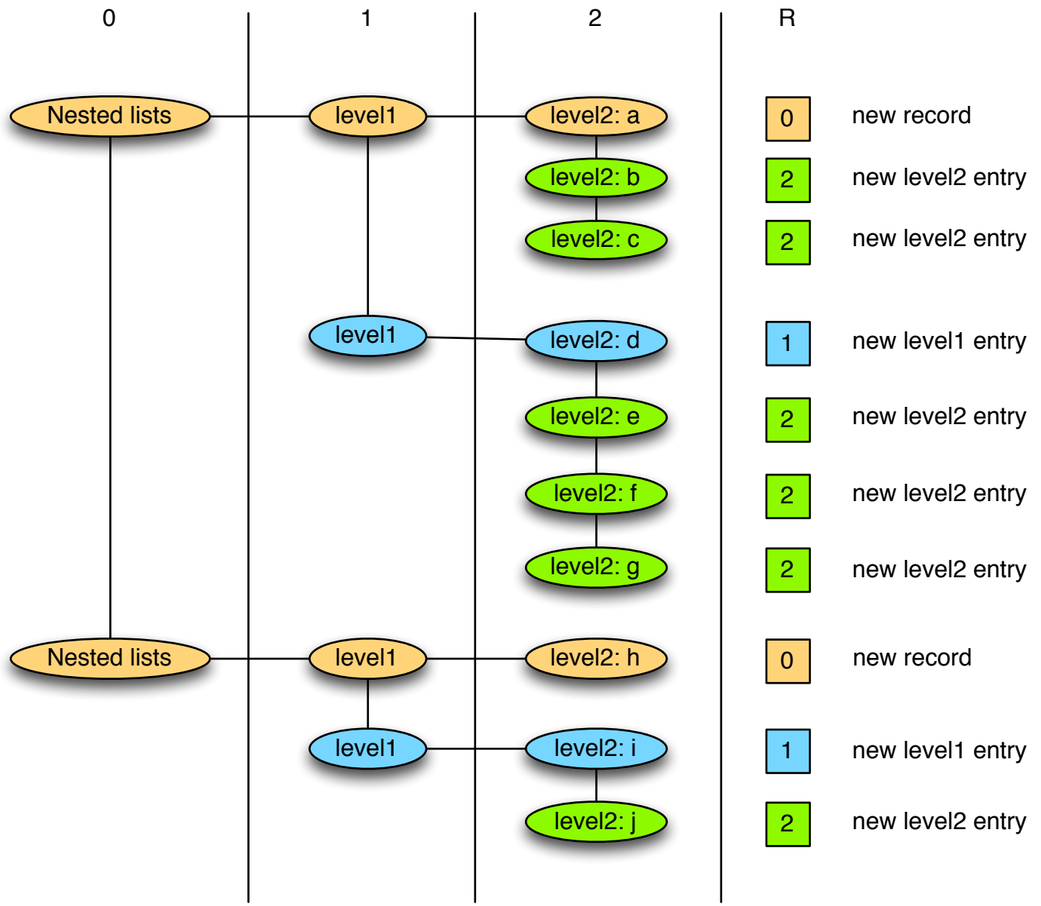
```
message nestedLists {
    repeated group level1 {
        repeated string level2;
    }
}
```

Records:

```
[[a, b, c], [d, e, f, g]]
[[h], [i, j]]
```

Columns:

```
Level: 0,2,2,1,2,2,2,0,1,2
Data:  a,b,c,d,e,f,g,h,i,j
```



more details: https://blog.twitter.com/2013/dremel-made-simple-with-parquet

http://parquet.io

Differences of Parquet and ORC Nesting support

- Parquet:

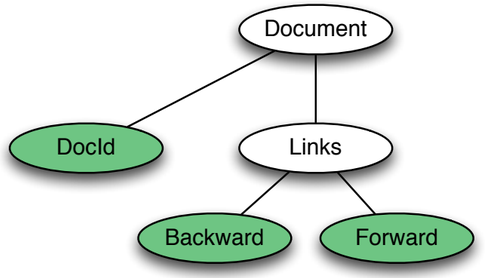
  Repetition/Definition levels capture the structure.

  => one column per *Leaf* in the schema.

  Array<int> is one column.

  Nullity/repetition of an inner node is stored in each of its children

  => One column independently of nesting with some redundancy.



- ORC:

  An extra column for each Map or List to record their size.

  => one column per *Node* in the schema.

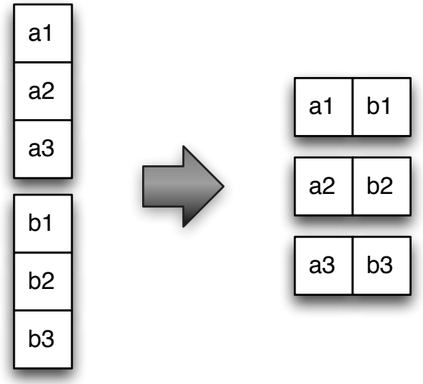
  Array<int> is two columns: array size and content.

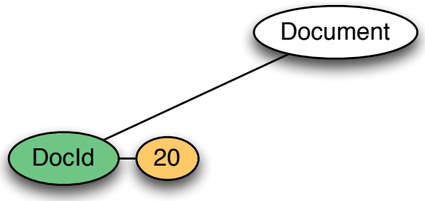
  => An extra column per nesting level.

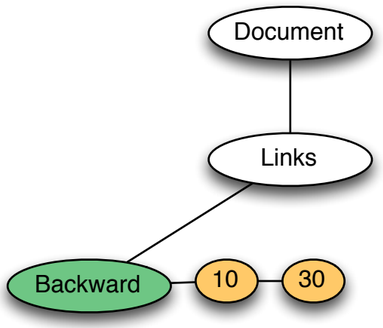
http://parquet.io

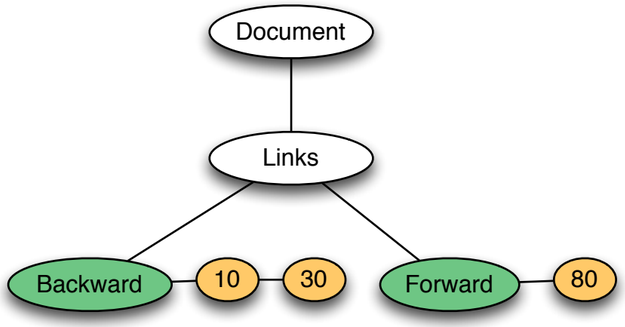
Reading assembled records

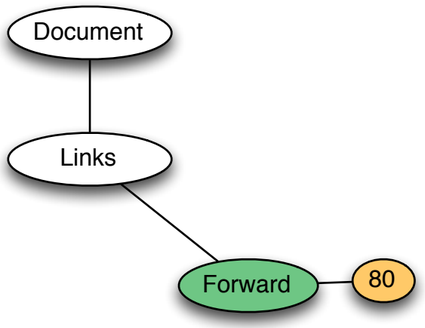
- Record level API to integrate with existing row based engines (Hive, Pig, M/R).
- Aware of dictionary encoding: enable optimizations.
- Assembles projection for any subset of the columns: only those are loaded from disc.











http://parquet.io

Projection push down

- **Automated in Pig and Hive:**

Based on the query being executed only the columns for the fields accessed will be fetched.

- **Explicit in MapReduce, Scalding and Cascading using globing syntax.**

Example: `field1;field2/**;field4/{subfield1,subfield2}`

Will return:

`field1`

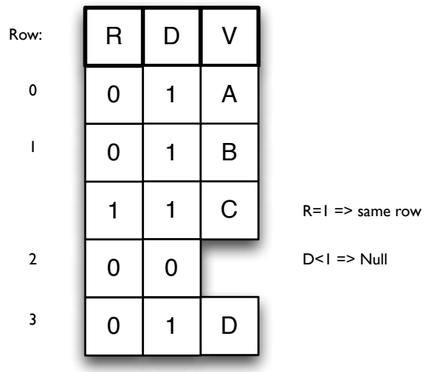
all the columns under `field2`

`subfield1` and 2 under `field4`

but not `field3`

Reading columns

- To implement column based execution engine
- 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 on strings.



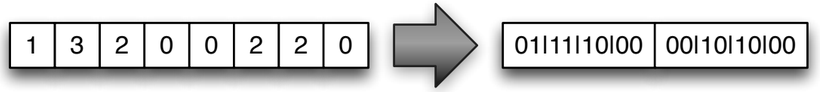
Integration APIs

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

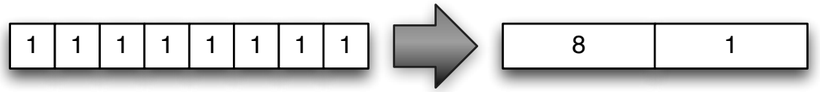
- Integration with existing type systems and processing frameworks:
  - Impala
  - Pig
  - Thrift and Scrooge for M/R, Cascading and Scalding
  - Cascading tuples
  - Avro
  - Hive
  - Spark

Encodings

- **Bit packing:**
  - Small integers encoded in the minimum bits required
  - Useful for repetition level, definition levels and dictionary keys



- **Run Length Encoding:**
  - Used in combination with bit packing
  - Cheap compression
  - Works well for definition level of sparse columns.



- **Dictionary encoding:**
  - Useful for columns with few (< 50,000) distinct values
  - When applicable, compresses better and faster than heavyweight algorithms (gzip, lzo, snappy)
- **Extensible:** Defining new encodings is supported by the format

Parquet 2.0

- **More encodings:** compact storage without heavyweight compression
  - Delta encodings: for integers, strings and sorted dictionaries.
  - Improved encoding for strings and boolean.

- **Statistics:** to be used by query planners and predicate pushdown.

- **New page format:** to facilitate skipping ahead at a more granular level.

Main contributors

Julien Le Dem (Twitter): Format, Core, Pig, Thrift integration, Encodings

Nong Li, Marcel Kornacker, Todd Lipcon (Cloudera): Format, Impala

Jonathan Coveney, Alex Levenson, Aniket Mokashi, Tianshuo Deng (Twitter): Encodings, projection push down

Mickaël Lacour, Rémy Pecqueur (Criteo): Hive integration

Dmitriy Ryaboy (Twitter): Format, Thrift and Scrooge Cascading integration

Tom White (Cloudera): Avro integration

Avi Bryant, Colin Marc (Stripe): Cascading tuples integration

Matt Massie (Berkeley AMP lab): predicate and projection push down

David Chen (Linkedin): Avro integration improvements

How to contribute

Questions? Ideas? Want to contribute?

Contribute at: github.com/Parquet

Come talk to us.

Cloudera

Criteo

Twitter