An Empirical Evaluation of Columnar Storage Formats

ACM Digital Library, 2023

Authors: Xinyu Zeng, Yulong Hui, Jiahong Shen, Andrew Pavlo, Wes McKinney Voltron, Huanchen Zhang

Akshay Sadhu, David Teng, Isha Perry, Gabriel Sanson, Sebastian Jankowski

Georgia Tech

Introduction

Introduction to Columnar Storage Formats

- **Why Columnar Storage?**
  - Optimized for data analytics via:
    1. **Irrelevant attribute skipping**
    2. **Efficient data compression**
    3. **Vectorized query processing**
- **Popular Formats**: Parquet and ORC (adopted in data lakes and warehouses)
  - **Challenges**:
    1. Developed a decade ago, not optimized for modern hardware.
    2. Current hardware supports high-bandwidth storage but with high latency in cloud environments (e.g., AWS S3, Azure Blob).
    3. New lightweight compression, indexing, and filtering methods have emerged, yet existing formats rely on outdated DBMS techniques.

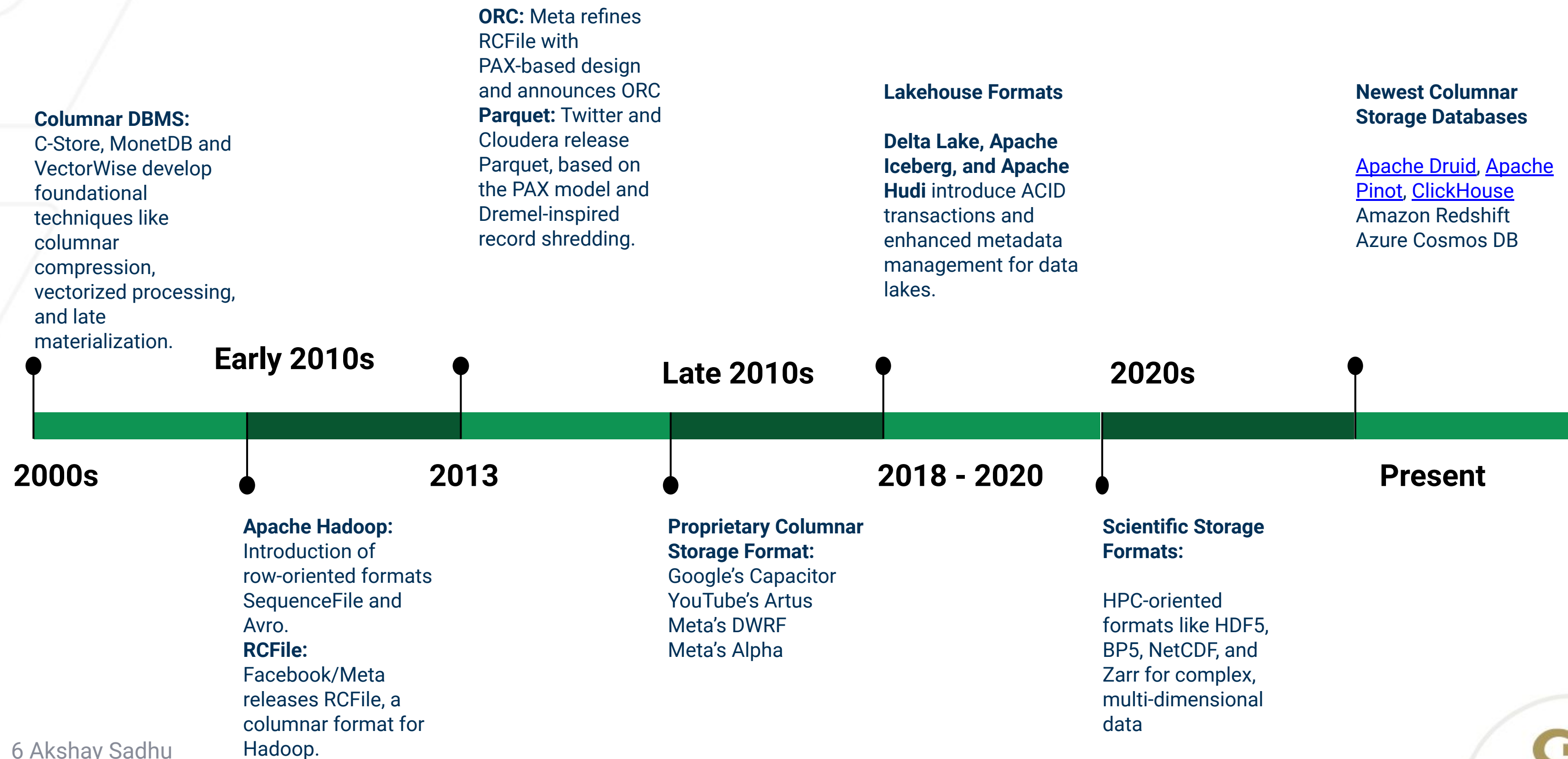
Georgia Tech

Research Goals And Key Findings

- **Study Objective**: Evaluate and benchmark Parquet and ORC to guide the design of next-generation columnar formats.
  a. **Benchmark Creation**: Uses realistic workloads based on real-world data to test components like encoding, compression, indexing, and nested data modeling.
  b. **Machine Learning Focus**: Examines efficiency for ML workloads and GPU processing.
- **Key Findings**:
  a. **File Size & Decoding**: Parquet has smaller file size due to aggressive encoding; ORC excels in selection pruning.
  b. **Compression Trade-offs**: Faster decoding preferred over high compression as storage becomes cheaper.
  c. **ML Workloads & GPU**: Current formats lack parallelism and support for large ML projections; more aggressive compression needed for GPU utilization.

Background & Related Work

Timeline



Feature Taxonomy

Feature Taxonomy Overview

- Authors developed feature taxonomy for Parquet and ORC

- Convenient way to identify commonalities and differences
  - Primarily discuss rationale behind different implementations of features

Table 1: Feature Taxonomy – An overview of the features of columnar storage formats.

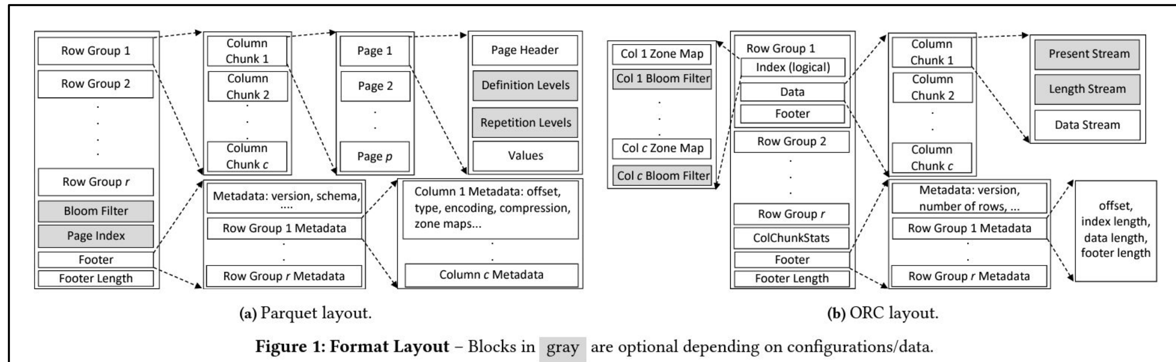
		Parquet	ORC
FEATURES	Internal Layout (§3.1)	PAX	PAX
	Encoding Variants (§3.2)	plain, RLE_DICT, RLE, Delta, Bitpacking	plain, RLE_DICT, RLE, Delta, Bitpacking, FOR
	Compression (§3.3)	Snappy, gzip, LZO, zstd, LZ4, Brotli	Snappy, zlib, LZO, zstd, LZ4
	Type System (§3.4)	Separate logical and physical type system	One unified type system
	Zone Map / Index (§3.5)	Min-max per smallest zone map/row group/file	Min-max per smallest zone map/row group/file
	Bloom Filter (§3.5)	Supported per column chunk	Supported per smallest zone map
	Nested Data Encoding (§3.6)	Dremel Model	Length and presence

Georgia Tech

Format Layout

	Parquet	ORC
Row Group Size	Defined by row count (1M rows)	Defined by physical size (64 MB)
Block Compression	Maps compression unit to smallest zone map	Allows tuning for performance vs. space trade-off

- **Encoding & Compression:** Lightweight encoding per column chunk, followed by block compression.
- **Footer Metadata:** Stores schema, tuple count, row group offsets, and zone maps.
- **Internal Layout:** PAX



(a) Parquet layout.

(b) ORC layout.

Figure 1: Format Layout – Blocks in gray are optional depending on configurations/data.

Georgia Tech

Encoding

Parquet Encoding:

- **Dictionary Encoding:**
  - Effective for large-value integers, compresses by mapping values to codes.
  - Limited to 1 MB dictionary size per column chunk; excess values are stored “plain.”
- **RLE (Run-Length Encoding) + Bitpacking:**
  - Applied to dictionary codes.
  - RLE for runs of 8+ consecutive values; otherwise, Bitpacking.
  - **Limitation:** Fixed RLE threshold (8) lacks flexibility for data with varying patterns.

ORC Encoding:

- **NDV Ratio Threshold:**
  - Determines whether Dictionary Encoding is applied, based on column's NDV / row count.
- **Four Encoding Schemes for Integer Columns:**
  - **RLE:** For repeated values of 3-10 occurrences.
  - **Delta Encoding:** For long runs and increasing/decreasing patterns.
  - **Bitpacking & PFOR:** Based on subsequence characteristics.
  - **Advantage:** Higher compression opportunity by detecting patterns.
  - **Trade-off:** Increased complexity and metadata requirements may slow down decoding.

Georgia Tech

Compression

	Parquet	ORC
Compression	Snappy, gzip, LZO, zstd, LZ4, Brotli	Snappy, zlib, LZO, zstd, LZ4

- Both use block compression by default, but differ in configurability
    - Parquet: exposes all configuration to users
    - ORC: provides wrapper with two pre-configured options:
        - (1) Optimize for speed
        - (2) Optimize for compression

- Key observation: block compression is unhelpful/detrimental to query speed when used on columnar storage formats

Type System

	Parquet	**ORC**
Type System	Separate logical and physical type system	One unified type system

- Drastically different type systems
  - Parquet: minimal set of primitive types, all other types encoded as primitive
  - ORC: separate implementation for each type

- Both support complex types (e.g. Struct, List, Map)
  - Parquet does not support Union type

Georgia Tech

Zone Map / Index

	Parquet	**ORC**
Zone Map / Index	Min-max per smallest zone map/row group/file	Min-max per smallest zone map/row group/file

- Metadata which contains (1) min value, (2) max value, (3) row count of a range in the file
  - Improves query performance by skipping unnecessary data

- Both contain zone maps at file and row group level
  - Parquet: Smallest granularity is physical page
  - ORC: Smallest granularity is configurable value

Georgia Tech

Zone Map / Index (cont.)

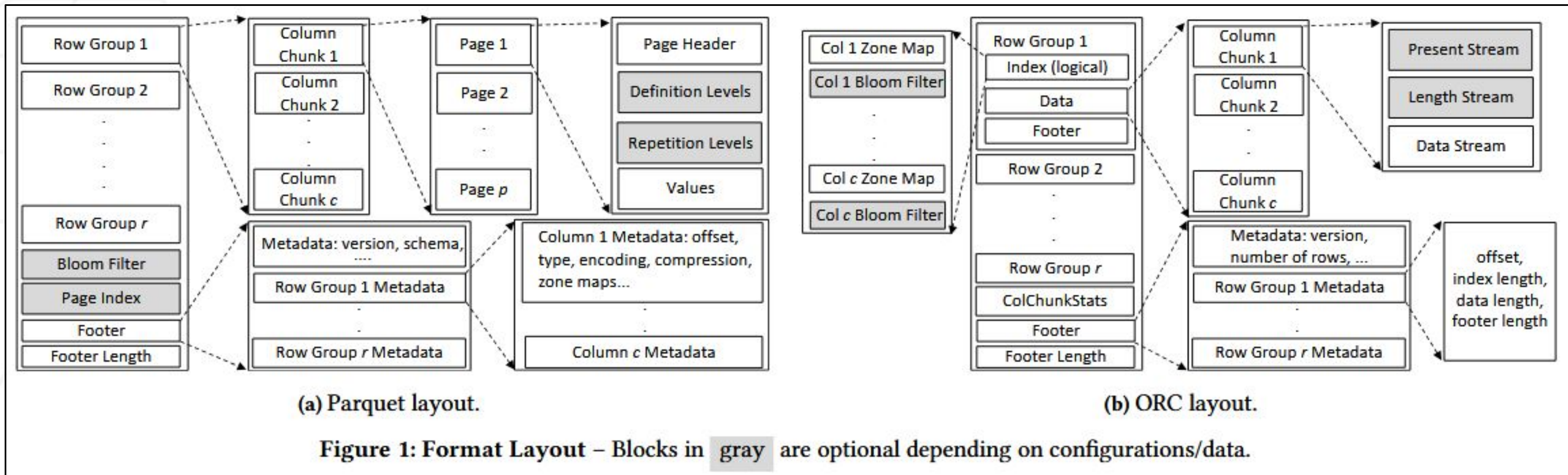


Figure 1: Format Layout – Blocks in gray are optional depending on configurations/data.

- Parquet: originally stored small zone maps in page headers
  - caused expensive random I/O
  - updated to add optional “PageIndex” → Store small zone maps in one place
- ORC: stores smallest zone maps at start of each row group

Georgia Tech

Bloom Filter

	Parquet	**ORC**
Bloom Filter	Supported per column chunk	Supported per smallest zone map

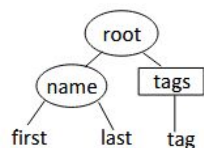
- Cheap data structure which says if a value is likely absent from a column
  - Improves query performance by skipping unnecessary data

- Differing levels of granularity
  - Parquet: Granularity only at column chunk level because PageIndex is optional
  - ORC: Same granularity as the smallest zone maps; co-located with them

Georgia Tech

Nested Data Encoding

	Parquet	**ORC**
Nested Data Encoding	Dremel Model	Length and presence



- Parquet encoding only uses atomics → duplicated non-atomics → file size > ORC’s

(a) Example schema and three sample records.

first

Val	R	D
Mike	0	2
	0	1
Joe	0	2

last

Val	R	D
Lee	0	2
Hill	0	2
	0	1

tag

Val	R	D
a	0	2
b	1	2
	0	1
c	0	2

(b) Parquet’s. R/D=Repetition/Definition Level.

name

P
1
1
1

first

Val	P
Mike	1
	0
Joe	1

last

Val	P
Lee	1
Hill	1
	0

tags

Len	P
2	1
0	1
1	1

tag

Val	P
a	1
b	1
c	1

(c) ORC’s. Len=length, P=presence.

Figure 3: Nested Data Example – Assume all nodes except the root can be null.

Georgia Tech

Columnar Storage Benchmark

Column Properties

- NDV Ratio: $f_{cr} = N / DV$
- Null Ratio: $| \{i | a_i \text{ is null}\} | / N$
- Value Range
- Sortedness:

$$asc = |\{i | 1 \le i < n \text{ and } a_i < a_{i+1}\}|; \; desc = |\{i | 1 \le i < n \text{ and } a_i > a_{i+1}\}|$$

$$eq = |\{i | 1 \le i < n \text{ and } a_i = a_{i+1}\}|; \; f_{sortness} = \frac{\max(asc, desc) + eq - \lfloor \frac{N}{2} \rfloor}{\lceil \frac{N}{2} \rceil - 1}$$

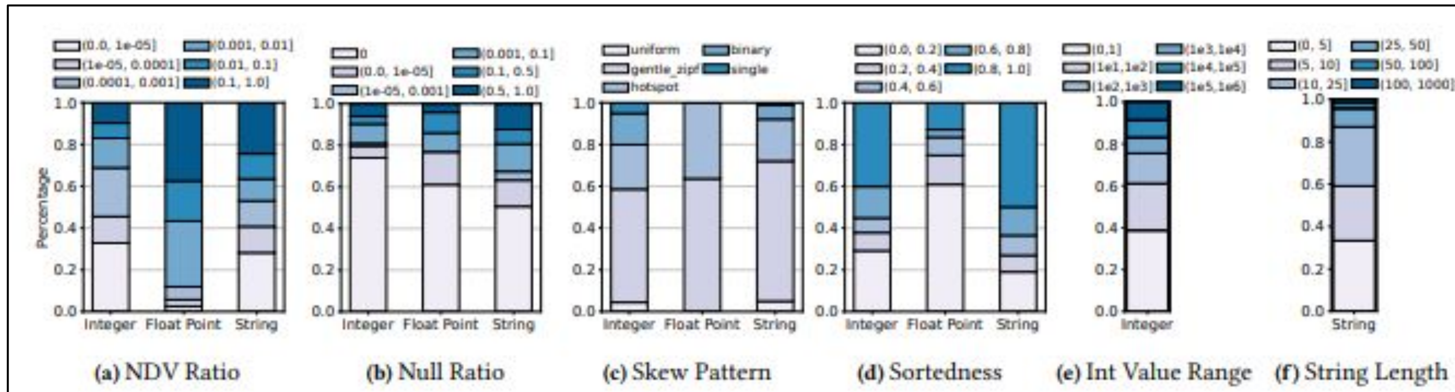
- Skewness:

$$p(k) = \frac{1}{k^s} / \left(\sum_{n=1}^{C} \frac{1}{n^s}\right)$$

Georgia Tech

Parameter Distribution in Real-World Data

- Public BI Benchmark: real-world data and queries from Tableau with 206 tables
- UCI-ML: 622 data sets for ML training
- Yelp: Yelp's businesses, reviews, and user information
- LOG: log information of internet search traffic for EDGAR filings through SEC.gov
- Geonames: geographical information of all countries
- IMDb: data sets that with basic information, ratings, and reviews for movies in a collection



Georgia Tech

Predefined Workloads

- Split into five workloads of bi, classic, Geo, LOG, and ML
- Core workload was created as a default for evaluation
- Selectivity specified based on the window of data collection
  - For example Geo and LOG with smaller windows utilized lower selectivity

Table 3: Data type distribution of different workloads – Number in the table indicating the proportion of columns.

Type	core	bi	classic	geo	log	ml
Integer	0.37	0.46	0.33	0.31	0.22	0.24
Float	0.21	0.20	0.06	0.08	0.46	0.39
String	0.41	0.34	0.61	0.61	0.32	0.37
Bool	0.003	0.002	0.00	0.00	0.00	0.01

Table 4: Summarized Workload Properties – We categorize each property into three levels. The darker the color the higher the number.

Properties	core	bi	classic	geo	log	ml
NDV Ratio	0.12	0.08	0.25	0.18	0.08	0.12
Null Ratio	0.09	0.11	0.09	0.13	0.02	0.00
Value Range	medium	small	large	small	small	large
Sortedness	0.54	0.57	0.49	0.45	0.75	0.30
Zipf s	1.12	1.10	1.42	0.89	1.26	1.00
Pred. Selectivity	mid	high	high	low	low	mid

Georgia Tech

Experimental Evaluation

Experiment Setup

Environment: AWS i3.2x large instance

- 8 Intel Xeon vCPUs, 61GB memory , 1.7TB NVMe SSD storage
- OS: Ubuntu 20.04 LTS

Data Generation and File Format: Arrow v9.0.0 for test file generation

- Parquet: 1 million rows, dictionary page size limit of 1 MB
- ORC: 64MB row group size, default NDV-ratio threshold of 0.8
- 20-column table with 1m rows for each workload

Methodology

- Take the average measurement of 3 runs per test
- Focus on raw scan performance of Parquet and ORC
- Perform a sequential scan, report the execution time
- Clear the buffer cache and perform 30 select queries
- Report the average latency of the select queries

Georgia Tech

Benchmark Result Overview

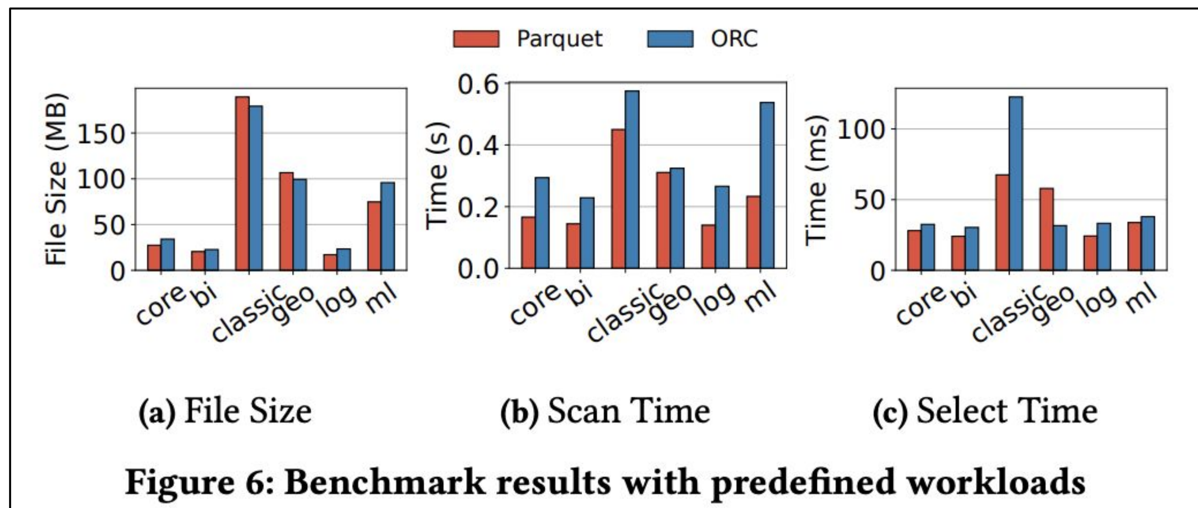


Figure 6: Benchmark results with predefined workloads

File Size: Neither format was consistently better

Scan Time: Parquet performed faster scans due to lightweight integer encoding

Select Time: Parquet had lower average latencies for the select queries, except on the `geo` workload

Georgia Tech

Encoding Analysis

Compression Ratio

- **Parquet:** better for integers with low/medium NDV ratio, floats, integer value ranges
- **ORC:** better for large NDV ratio and highly sorted integer columns

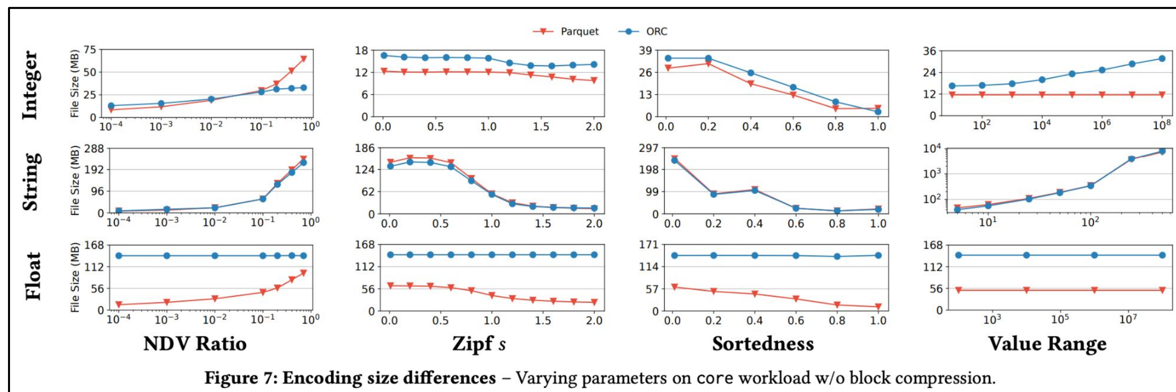
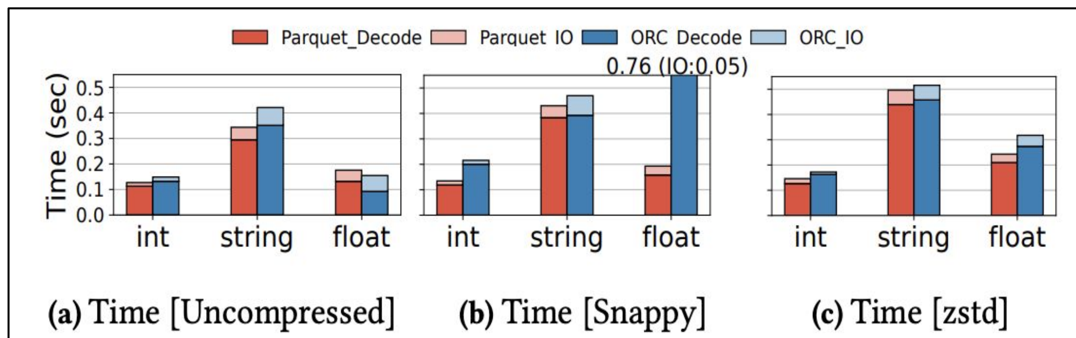


Figure 7: Encoding size differences – Varying parameters on core workload w/o block compression.

Decoding Speed (Fig 8a)

- **Parquet:** Faster decoding for integer and string columns
- **ORC:** Faster decoding for floats, ORC does not apply float encoding algorithms



(a) Time [Uncompressed] (b) Time [Snappy] (c) Time [zstd]

Georgia Tech

Block Compression

Compression Ratio

- Ztsd achieves a better compression ratio than Snappy for all data types

Scan Time

- Applying Zstd to Parquet only speeds up scans on slow storage tiers
- Decompression overhead of Ztsd hinders scan performance for faster storage devices

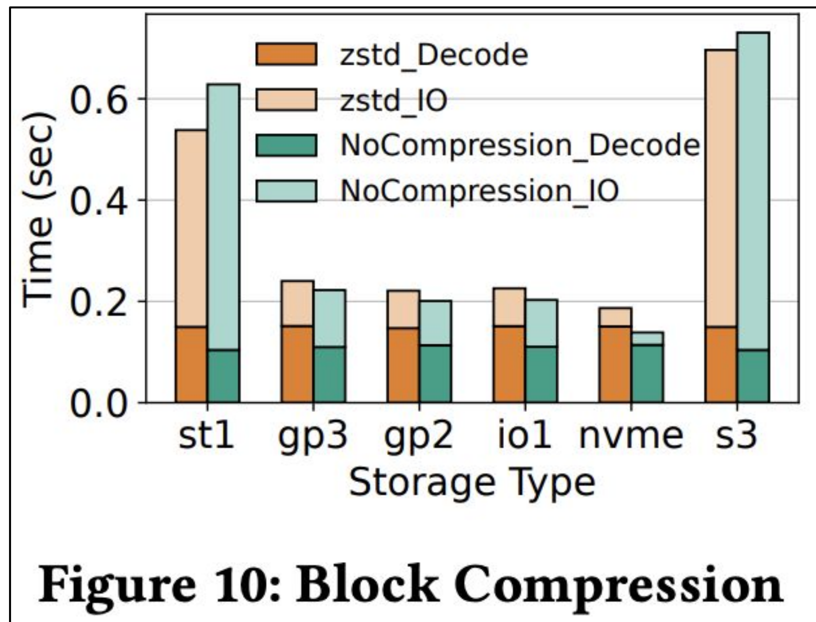


Figure 10: Block Compression

Georgia Tech

Wide-Table Projection

Experiment Data:

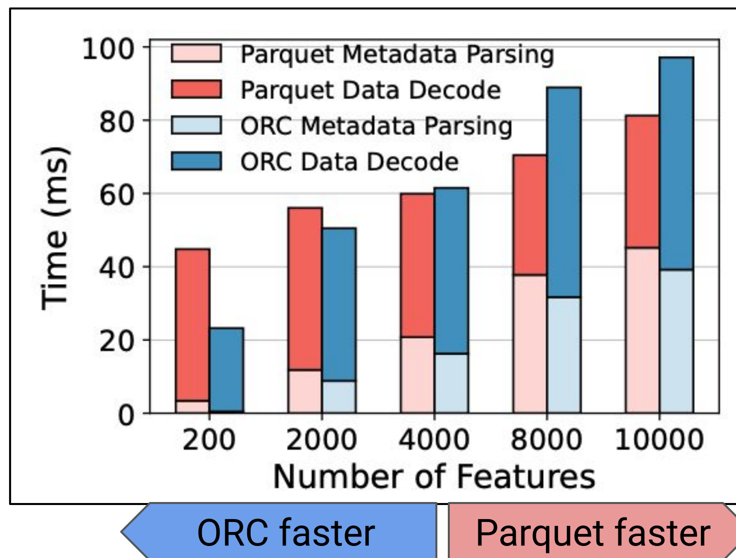
- Table of 10K rows with a varying number of float attributes stored in Parquet and ORC.

Task:

- Randomly selecting 10 attributes to project.

Results:

- **Metadata Parsing:** Both Parquet and ORC show increasing metadata parsing time as the number of features grows.
- **Decoding:** ORC's data decode time grows more rapidly compared to Parquet for wide tables.



Parquet generally handles wide-tables (>4000 features) projections more efficiently than ORC.

Georgia Tech

Machine Learning Workloads

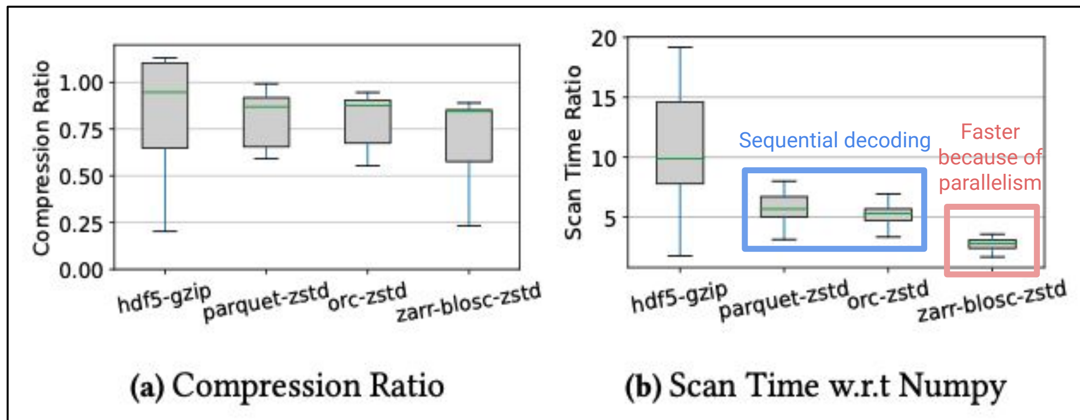
Compression Ratio and Deserialization Performance

Experiment Data:

- 30 data sets with vector embeddings from the top downloaded and top trending lists on Hugging Face

Task:

- Measure compression ratio and scan time with NumPy arrays for embeddings.



(a) Compression Ratio (b) Scan Time w.r.t Numpy

Results:

- None of the four formats achieves good compression with vector embeddings
- Zarr outperformed in scan time, because of it supports parallel reads. Parquet and ORC struggled due to their sequential decoding processes.

Georgia Tech

Machine Learning Workloads

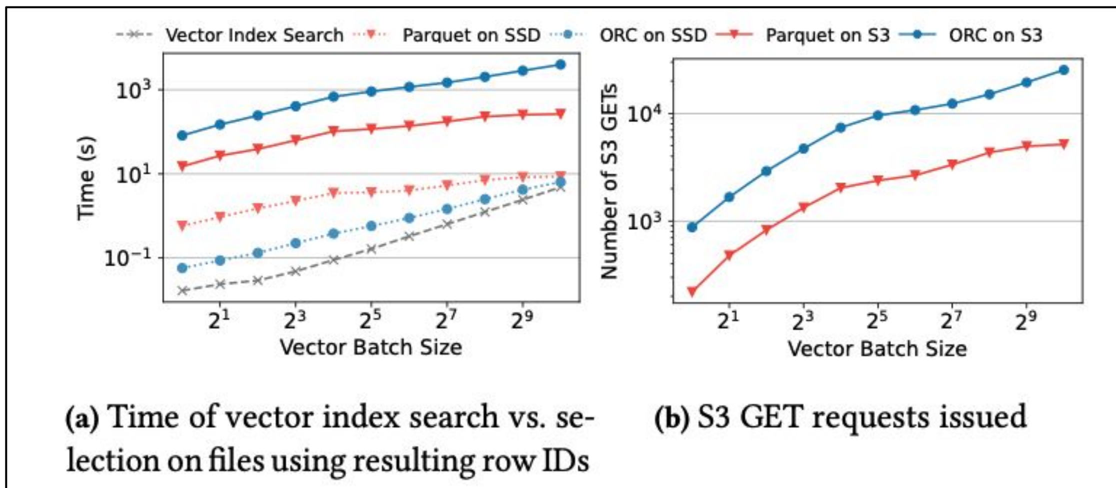
Top-k Similarity Search Performance

Experiment Data:

- LAION-5B dataset - open-source dataset containing 5 billion image-text pairs

Task:

- Perform top-k similarity search queries using 100M embeddings for nearest neighbors (k=10)



(a) Time of vector index search vs. selection on files using resulting row IDs

(b) S3 GET requests issued

Results:

- **ORC** is better on **SSD**: because of **fine-grained zone maps**.
- **Parquet** is better on **S3**: because of **faster GET requests**

Georgia Tech

Machine Learning Workloads

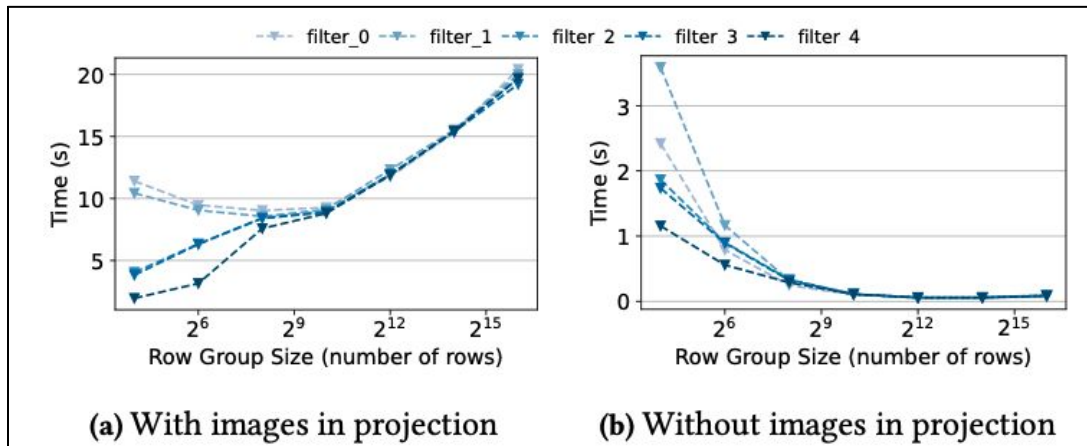
Storage of Unstructured Data

Experiment Data:

- LAION-5B dataset with Parquet storing URLs and metadata. 13 GB with 219K rows and is stored on NVMe SSD

Task:

- Evaluate filter scan queries on Parquet data using varying filter selectivities (1, 0.1, 0.01, 0.001, and 0.0001, respectively).



(a) With images in projection (b) Without images in projection

Results:

- **Smaller row groups** help with **unstructured data** (images) but hurt tabular queries.
- **Larger row groups** help with **structured data** but slow down image retrieval.

Georgia Tech

GPU Decoding

Experiment Data:

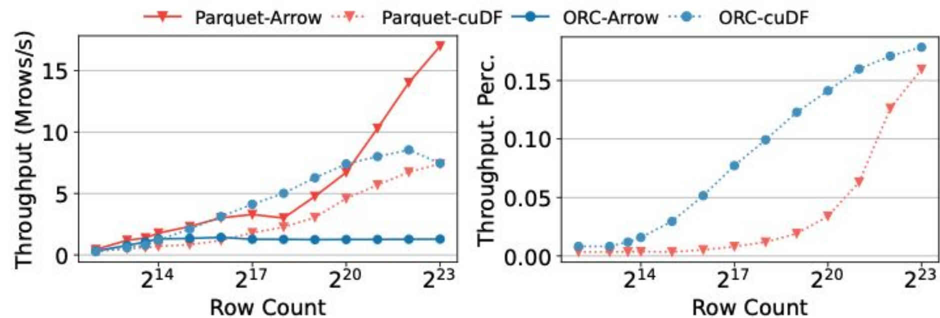
- Dataset stored in Parquet and ORC formats, 32 columns and varying rows

System Used:

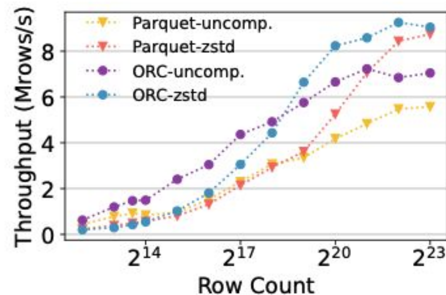
- NVIDIA RTX 3090, AMD EPYC 7H12 (128 cores), 512GB RAM, Intel P5530 NVMe SSD.

Task:

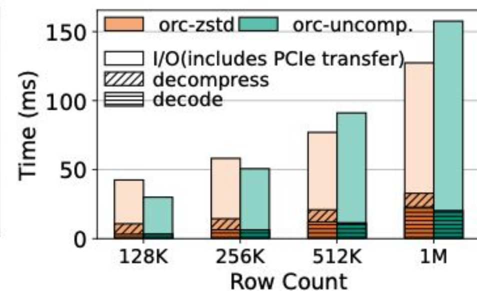
- Evaluate GPU decoding efficiency using Parquet-cuDF and ORC-cuDF, focusing on throughput and compression performance.



(a) core workload (b) Peak GPU Throughput Percentage



(c) cuDF varying compression



(d) Time breakdown of ORC in (c)

Results:

- **Throughput:** ORC-cuDF achieves higher throughput than Parquet-cuDF, leveraging GPU parallelism effectively.
- **Compression Impact:** ORC performs better only for smaller row count.

Georgia Tech

Lesson and Future Directions

Lessons Learned:

1. **Dictionary Encoding:** Effective for most data types; future formats should use it aggressively.
2. **Simple Encoding:** Simpler schemes boost decoding performance; avoid complex codec selection overhead.
3. **Query Bottleneck:** Shifting to CPU; use block compression sparingly unless highly beneficial.
4. **Metadata Layout:** Centralized and optimized for random access to support wide ML tables.
5. **Sophisticated Indexing:** Cheaper storage allows for better indexing/filtering to speed up queries.
6. **Nested Models:** Design for modern in-memory formats to reduce overhead.
7. **ML Workload Support:** Prioritize wide-table projections, low-selectivity efficiency, and float-specific compression.
8. **GPU Decoding:** Parallelizable encoding algorithms needed to maximize GPU utilization.

Future Focus:

Enhance parallelism, metadata organization, and GPU compatibility to meet evolving data needs.

Conclusion

Conclusion

- Evaluation of Columnar formats
- Tradeoffs tested with Real World data sets to demonstrate differences between encoding algorithms
- Experiments were conducted on the metrics of the format to show design differences and considerations important for ML workloads and modern hardware

Exam Questions

1. What compression techniques do both ORC and Parquet use? Which techniques are distinct to either ORC or Parquet?

2. Briefly explain the key differences between Parquet and ORC.

3. What are the benefits of using multiple compression techniques? What are the drawbacks?

4. How does the Number of Distinct Values (NDV) in a dataset affect the choice and efficiency of compression techniques in data storage formats like Parquet and ORC?

Thank You