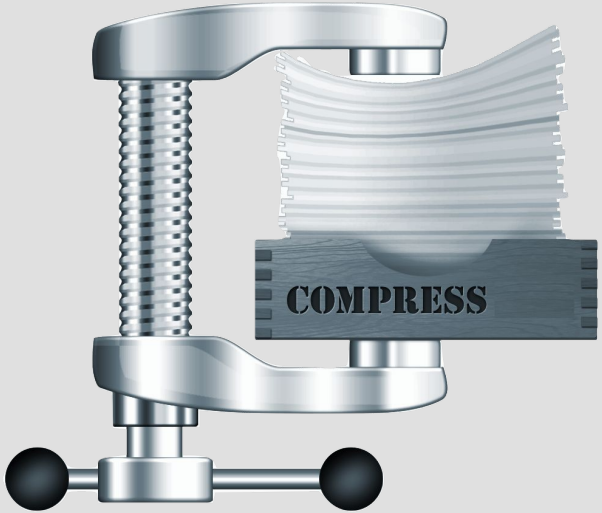


DATABASE COMPRESSION

Pooja Nilangekar [poojan@cmu.edu]
Rohit Agrawal [rohit10@cmu.edu]

PROJECT OBJECTIVE

Compressing the DBMS :-

- Use less space to store cold data
- Process less data per query
- Minimize speed overhead
- Use Delta & Dictionary encoding

GOALS (Revised)

75 %	100 %	125 %
✓ > Compressed Tile Class	✓ > Delta Encoding of Decimals	✓ > Deduplication of variable length field.
✓ > Metadata per tile	✓ > **SELECT** on compressed and uncompressed data	✗ Order Preserving Dictionary Encoding
✓ > Delta Encoding of Integers	★ Checkpointing the Compressed Tile	✓ > Compressing DataTable Offline.
✓ > Insertion of Data, followed by Compression		✗ TileGroup Compaction

QUICK OVERVIEW

Storage in Peloton

TileGroup #1

Name	ID	Year	Age	Class
Messi	9890	2002	40	A
Ronaldo	9891	2003	70	B
Rooney	9892	2004	50	C

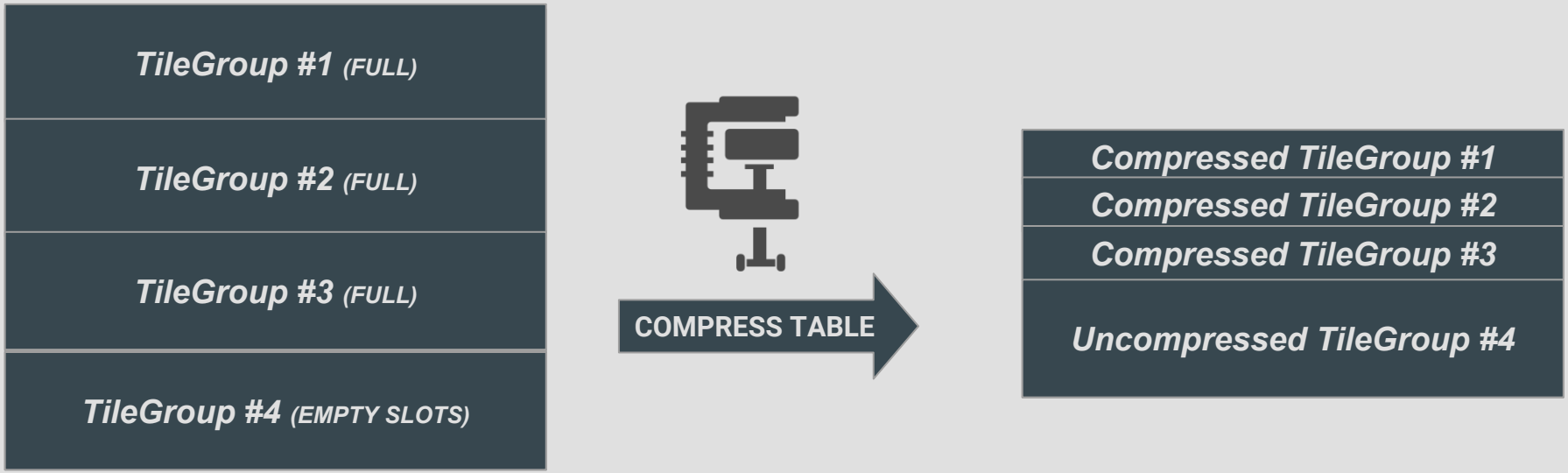
Tile #1 **Tile #2** **Tile #3**

TileGroup #2

Name	ID	Year	Age	Class
Bob	7894	2005	40	A
Sam	4244	2004	70	B
Carl	6746	2008	50	C

Data Table

METHODOLOGY



Uncompressed Data Table

Compressed Data Table

COMPRESSION : INTEGERS

Using Delta Encoding

1. Sort the Data
2. Base Value for Encoding ⇒ MEDIAN
3. Find Compressed Type
    a. Min Offset ⇒ Min - Median & Max Offset⇒ Max - Median
    b. Biggest DataType that can store these offsets ⇒ Compressed Type
4. Calculate and store offsets from base

OVERVIEW OF DELTA ENCODING

ID (BIGINT)	Year (SMALLINT)	ZipCode (INTEGER)
1000856	1996	15213
1000932	1992	15232
1000189	1990	15217
1000456	1994	15230
1000789	2000	15228

ID (BIGINT)	Year (SMALLINT)	ZipCode (INTEGER)
1000189	1990	15213
1000456	1992	15217
1000789	1994	15228
1000856	1996	15230
1000932	2000	15232

OVERVIEW OF DELTA ENCODING

ID
1000189
1000456
1000789
1000856
1000932

Median : 1000789
Minimum Offset : -600
Maximum Offset : 143

SMALLINT

Year
1990
1992
1994
1996
2000

Median : 1994
Minimum Offset : -4
Maximum Offset : 6

TINYINT

ZipCode
15213
15217
15228
15230
15232

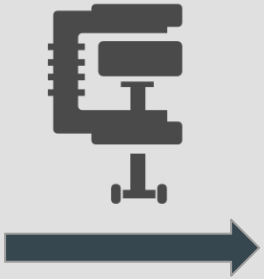
Median : 15228
Minimum Offset : -15
Maximum Offset : 4

TINYINT

OVERVIEW OF DELTA ENCODING

ID (BIGINT)	Year (SMALLINT)	ZipCode (INTEGER)
1000856	1996	15213
1000932	1992	15232
1000189	1990	15217
1000456	1994	15230
1000789	2000	15228

SIZE : 70 bytes



ID (SMALLINT)	Year (TINYINT)	ZipCode (TINYINT)
67	2	-15
143	-2	4
-600	-4	-11
-333	0	2
0	6	0

METADATA :

1000789	1994	1522

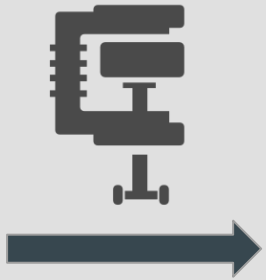
SIZE : 34 bytes

COMPRESSION : DECIMALS

1. Get Max Exponent for column
2. Multiply each value with this exponent ⇒ Integer Column
3. Compress like Integer Column [Delta Encoding]
4. In uncompression, instead of just adding offset to base, add [offset/exponent] to base

COMPRESSION : DECIMALS

TEMPERATURE (DECIMAL)
88.56
88.78
87.12
88.34
88.90



TEMPERATURE (DECIMAL)
0
22
-144
-22
34

METADATA :

88.56	100

SIZE : 40 bytes

SIZE : 21 bytes

COMPRESSION : VARCHAR

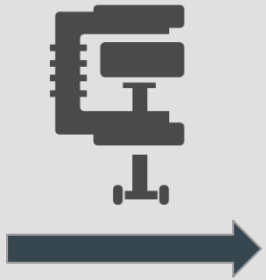
Using Deduplication

1. Cuckoo Hash ⇒ Dictionary
2. Store dictionary key instead of actual VARCHAR.
3. Compression Type : TINYINT / SMALLINT

COMPRESSION : VARCHAR

CITY (VARCHAR)
Pittsburgh
New York
New York
Pittsburgh
New York
Pittsburgh

SIZE : 99 bytes



CITY (TINYINT)
0
1
1
0
1
0

SIZE : 41 bytes

METADATA :

Pittsburgh	0
New York	1

EVALUATION

Workload:

- Temperature Dataset

Metrics Evaluated

- Compression Ratio
- Sequential Scan Timing.

EVALUATION : Temperature Dataset

Temperature Dataset

Characteristics :

- Size : 123.4 MB
- Number of Tuples ~ 2.5 million
- Tuple :

SITE (VARCHAR)	MONTH (INTEGER)	DAY (INTEGER)	YEAR (INTEGER)	TEMPERATURE (DECIMAL)

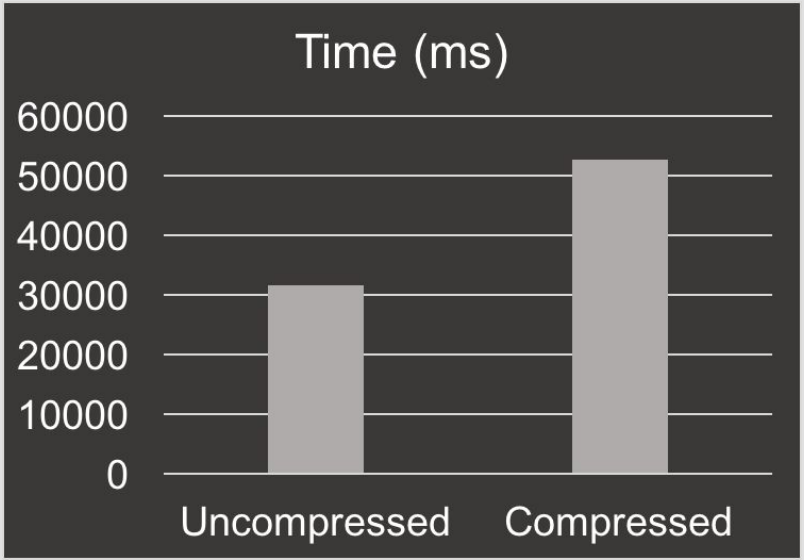
Source : http://academic.udayton.edu/kissock/http/Weather/

EVALUATION : COMPRESSION RATIO

Data	Size
Uncompressed Storage	18.124 MB
Compressed Storage	3.799 MB
Metadata Storage	0.259 MB

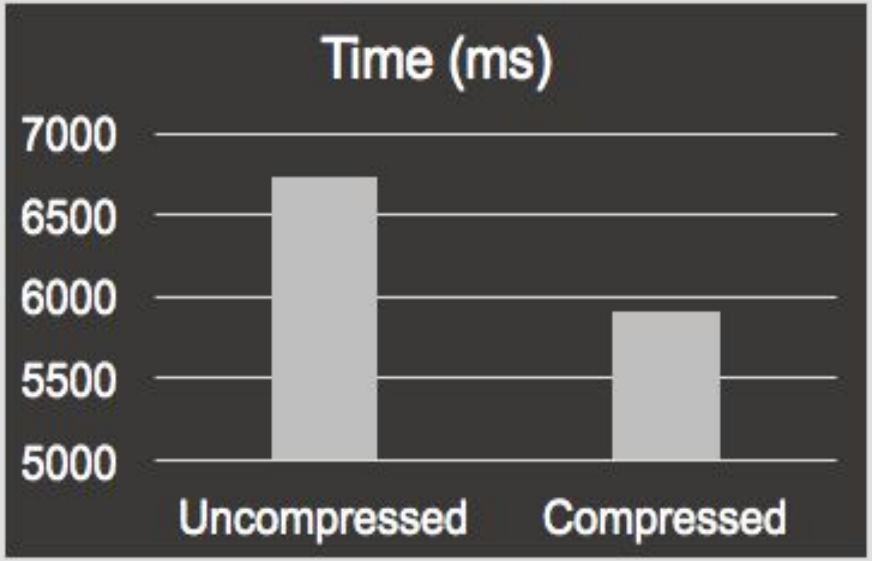
EVALUATION : TIMING

Time taken to compress the tuples : 46.12 seconds



Sequential Scan Time Comparison

`SELECT * FROM TABLE`

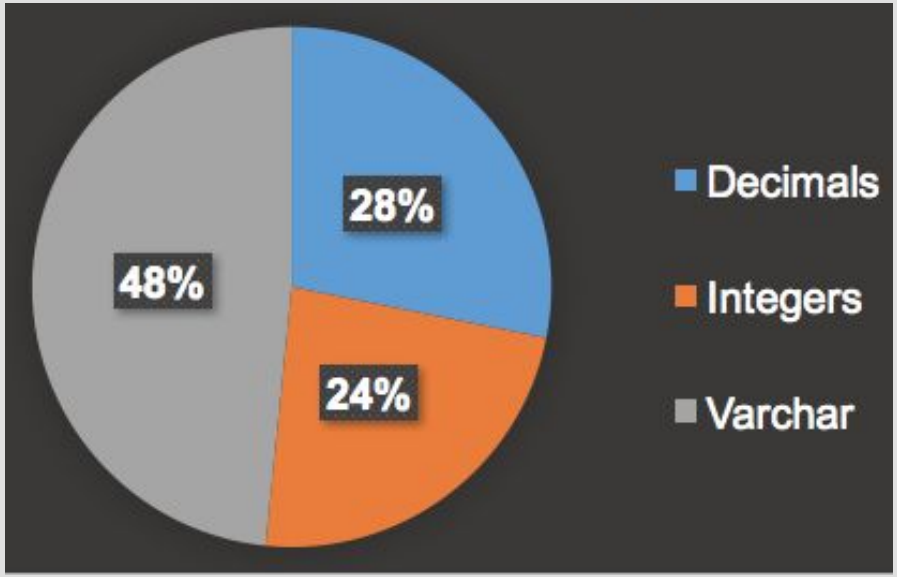


Sequential Scan Time Comparison

`SELECT COUNT(*) FROM TABLE`

EVALUATION : TIMING BREAKDOWN

Slowdown Analysis per column



ANALYSIS

Compression Ratio Analysis: Compression Ratios match Hyper, SAP HANA

Timing Analysis : Significant Performance slowdown while accessing compressed data

Reasons :

- Cuckoo Hash
- Lack of support for
    - Vectorized Instructions.
    - Range scan and point queries on Compressed Data
    - Integration with LLVM

Source Code Contributions - PELOTON

SOURCE

Files Added :

- **`src/include/storage/compressed_tile.h`**
- **`src/storage/compressed_tile.cpp`**

Files Modified :

- **`src/ include/storage/data_table.h`**
- **`src/storage/data_table.cpp`**
- **`src/include/storage/tile_group.h`**
- **`src/storage/tile_group.cpp`**
- **`src/include/storage/tile.h`**
- **`src/storage/tile.cpp`**

TEST

- Correctness Test *[INTEGERS, DECIMALS & VARCHAR]*

  >> **`INSERT -> COMPRESS -> INSERT (UNCOMPRESSED) -> SELECT.`**

- Compression Size Test *[INTEGER, DECIMALS & VARCHAR]*

FUTURE WORK

- Vectorization using SIMD
- Run Query on Compressed Data
- Dictionary Encoding
- Compress Tile Groups Parallely

SPEED, SPEED and MORE SPEED!

Thank You