Bitmap Indexing of Big Data

EBISS 2019 - Berlin (Germany) - July 5, 2019

Lawan Subba

Supervisors: Christian Thomsen and Torben Bach Pedersen
Aalborg University, Denmark

External Supervisor: Alberto Abello
Polytechnic University of Catalonia, Spain

Outline

1. **Introduction**
   A. Bitmap Index
   B. Distributed Bitmap Indexing Frameworks
2. **Papers**
   P1 - Efficient Indexing of Hashtags using Bitmap Indices - Conference Paper **(Published)**
   P2 - Bitmap indexing with Storage Structure Considerations - Conference Paper **(In Progress)**
   P3 - An Adaptive Bitmap Indexing Scheme for Distributed Environments - Conference Paper
   P4 - Multidimensional Online Analytical Processing on Cell Stores - Conference Paper
   P5 - Bitmap Indexing on Distributed Environments - Journal Paper
   P6 - DBIF: A demonstration of DBIF on Big Data - Demo Paper
3. **Other activities**
   A. PhD Courses
   B. Knowledge Dissemination

1(A): Bitmap Index - Background

RowId	Name	Married	Age
1	Alice	Y	20-30
2	Bob	N	20-30
3	Carol	N	30-40
4	Dave	N	30-40
5	Ed	Y	40-50

Bitmap Index

Married		Age		
Y	N	20-30	30-40	40-50
1	0	1	0	0
0	1	1	0	0
0	1	0	1	0
0	1	0	1	0
1	0	0	0	1

Run Length Encoding

Married		Age		
Y	N	20-30	30-40	40-50
1***1**	1*0	2***1**	2*0	4*0
3*0	1***3**	3*0	2***1**	1***1**
1***1**	1*0		1*0	

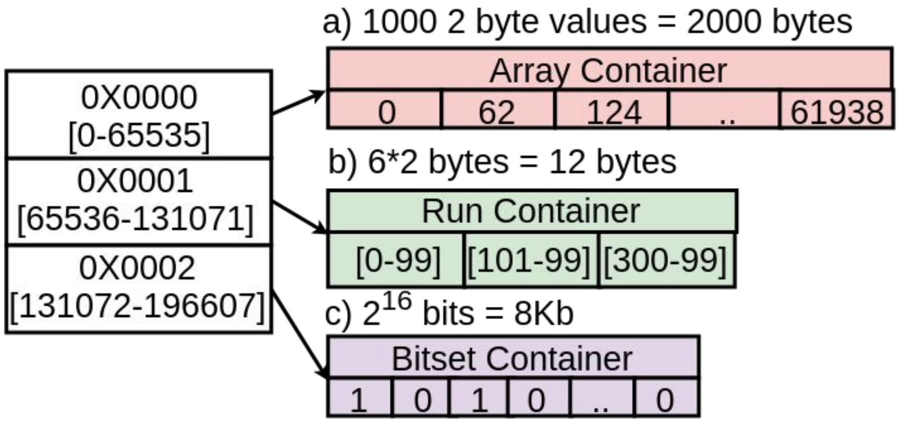
Married="Y"	Married="Y" AND Age="40-50"				
1	1		0		0
0	0	AND	0	=	0
0	0		0		0
0	0		0		0
1	1		1		1

Bitmap Index Example

1. Logical operations (AND/OR) are fast
2. Bitmaps are compressible

1(A): Bitmap Index - Roaring Bitmap

1. Divides data into chunks of 2^{16} [65,536]
2. Each chunk can be stored as one of 3 containers
   a. Array container
   b. Bitset container
   c. Run container
3. Wasteful to store [1, 50000, 90000] as Bitset
4. Fast random access, RLE must begin from the start always
5. Cache friendly



Roaring Bitmap

1(B): Distributed Bitmap Indexing Frameworks

1. Bitmap Index for Database Service (BIDS)
    a. *An efficient and compact indexing scheme for large-scale data store.* ICDE(2013) [3]
       - Peng Lu, Sai Wu, Lidan Shou, and Kian-Lee Tan
    b. Uses RLE based compression, bit-sliced encoding or partial indexing depending on the data characteristics.
    c. The compute nodes are organized according to the Chord protocol, and the indexes are distributed across the nodes.
2. Pilosa
    a. Open source (https://www.pilosa.com/)
    b. Slightly modified version of Roaring bitmap for compression.
    c. Bitmaps are sharded using their own data model and distributed
    d. Aggregate values are stored (Min, Max, Count)

1(B): Distributed Bitmap Indexing Frameworks

1. Bitmap Index for Database Service (BIDS)
   a. *An efficient and compact indexing scheme for large-scale data store.* ICDE(2013) [3]
      - Peng Lu, Sai Wu, Lidan Shou, and Kian-Lee Tan
   b. Uses RLE based compression, bit-sliced encoding or partial indexing depending on the data characteristics.
   c. The compute nodes are organized according to the Chord protocol, and the indexes are distributed across the nodes.
2. Pilosa
   a. Open source (https://www.pilosa.com/)
   b. Slightly modified version of Roaring bitmap for compression.
   c. Bitmaps are sharded using their own data model and distributed
   d. Aggregate values are stored (Min, Max, Count)

Existing Work

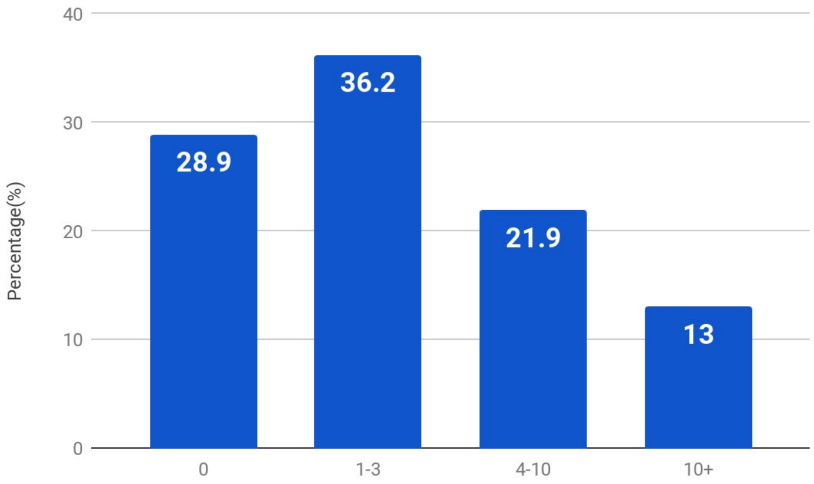
a. Fixed compression algorithm
b. Lock users to their specific implementation to store, distribute and retrieve bitmap indices.

P1: Contributions

1. An open source, lightweight and flexible distributed bitmap indexing framework for big data which integrates with commonly used tools incl. Apache Hive and Orc.
2. The bitmap compression algorithm to use and key-value store to store indices are easily swappable.
3. Demonstrate that search for substrings like hashtags in tweets can be greatly accelerated by using our bitmap indexing framework.
4. Published at DOLAP 2019

P1: Hashtags

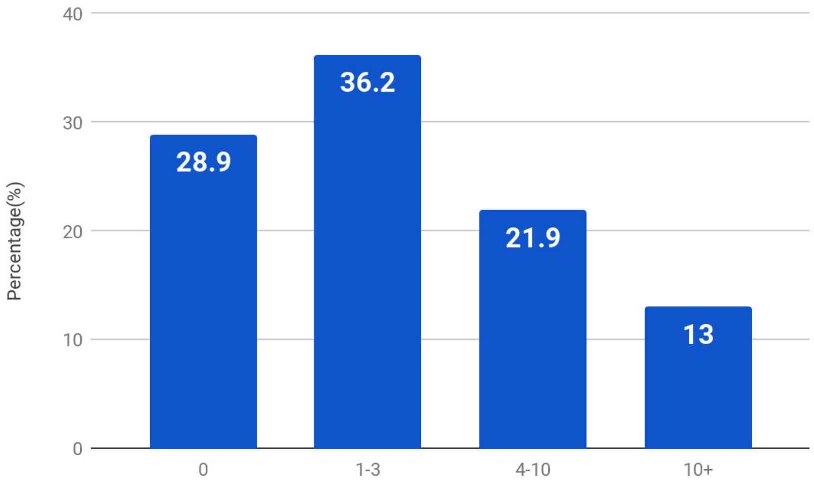
- A keyword containing numbers and letters preceded by a hash sign(#)
- Simplicity and lack of formal syntax



Distribution of Hashtags used in 8.9 million instagram posts in 2018 [1]

P1: Hashtags

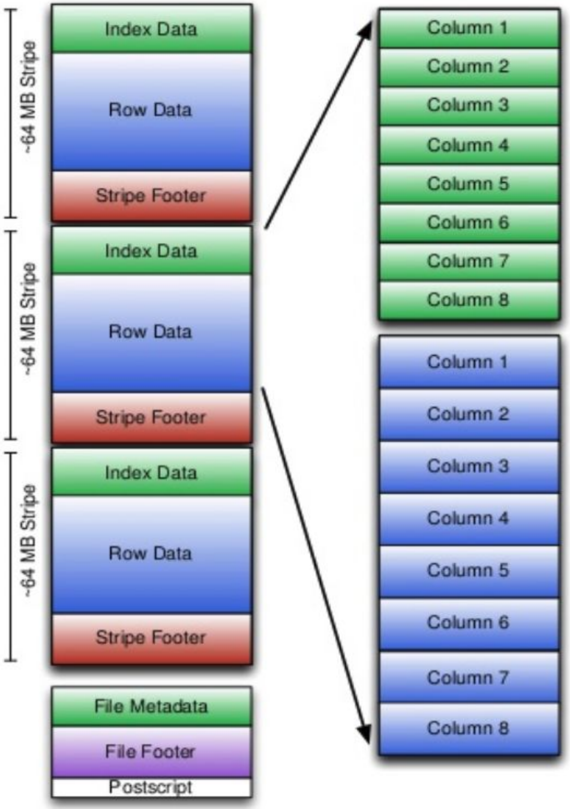
- A keyword containing numbers and letters preceded by a hash sign(#)
- Simplicity and lack of formal syntax
- **Challenge**
    - SELECT COUNT(*) FROM table WHERE (tweet LIKE “%#tag1%”)
    - SELECT COUNT(*) FROM table WHERE (tweet LIKE “%#tag1%” OR …)
    - SELECT COUNT(*) FROM table WHERE (tweet LIKE “%#tag1%” AND …)



Distribution of Hashtags used in 8.9 million instagram posts in 2018 [1]

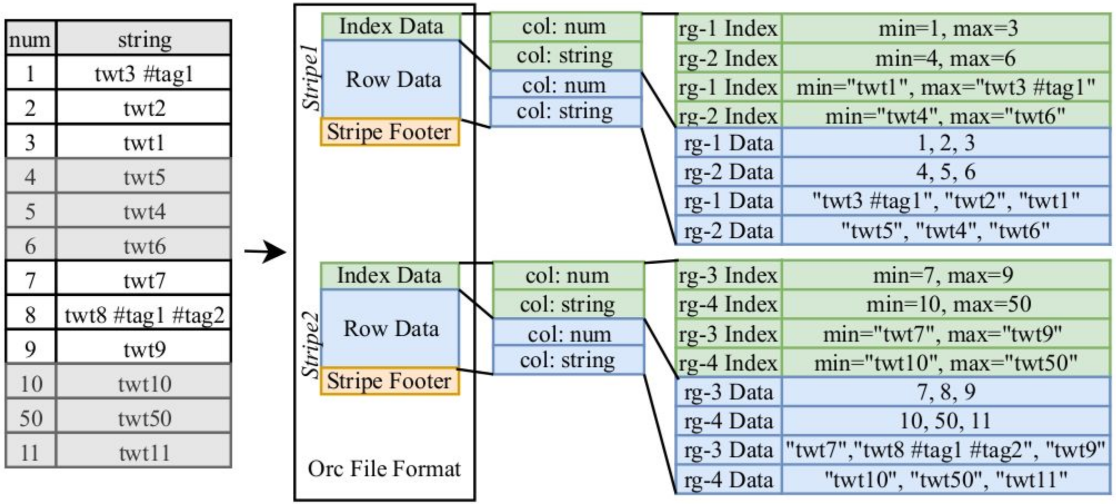
P1: Apache Orc

1. Storing data in a columnar format lets the reader read, decompress, and process only the values that are required by the current query.
2. Stripes=64MB and rowgroups = 10,000 rows
3. Min-max based Indices are created at rowgroup, stripe and file level.



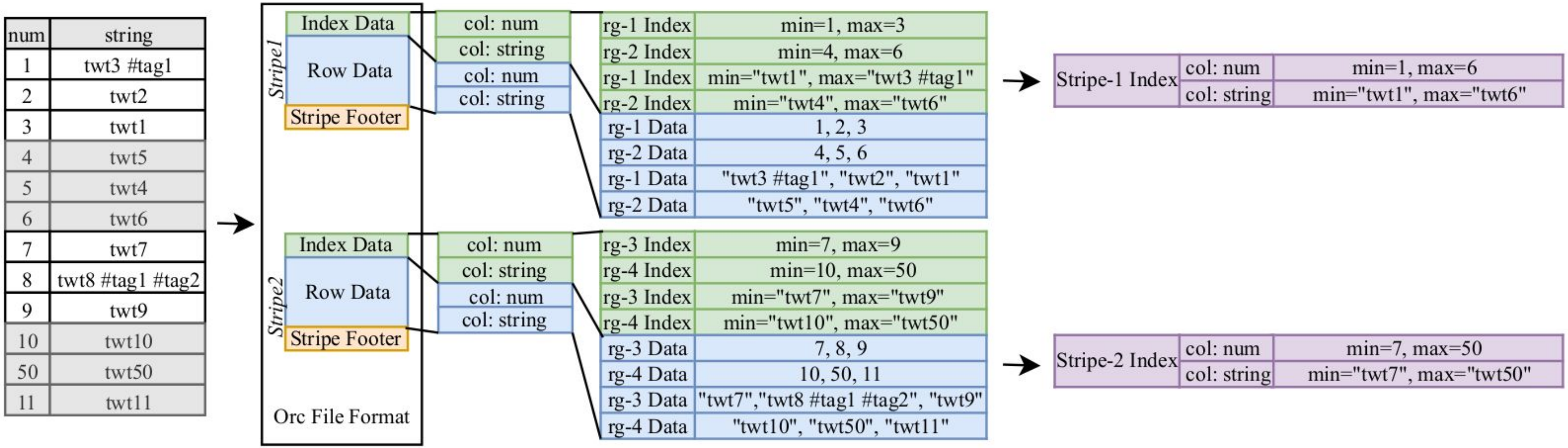
Orc File Format [2]

P1: Apache Orc



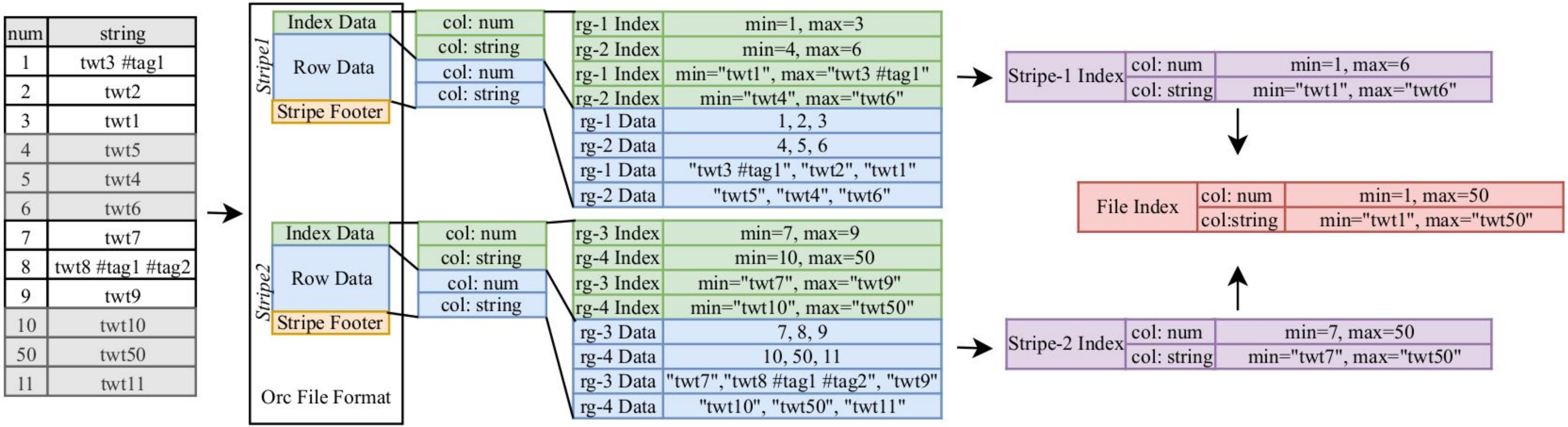
1. Min-max based indices

P1: Apache Orc



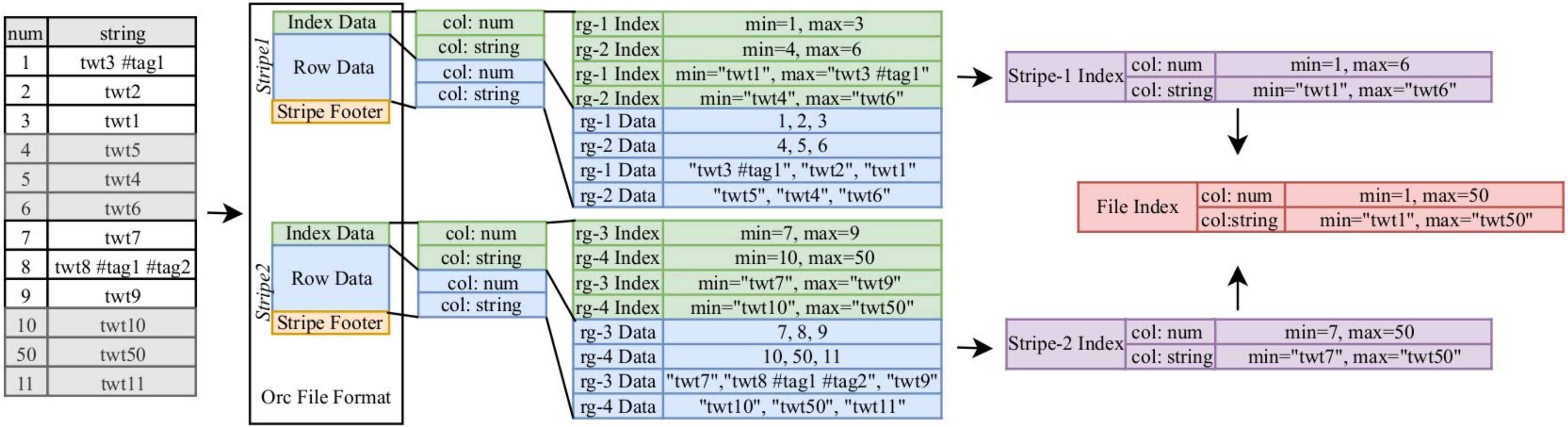
1. Min-max based indices

P1: Apache Orc



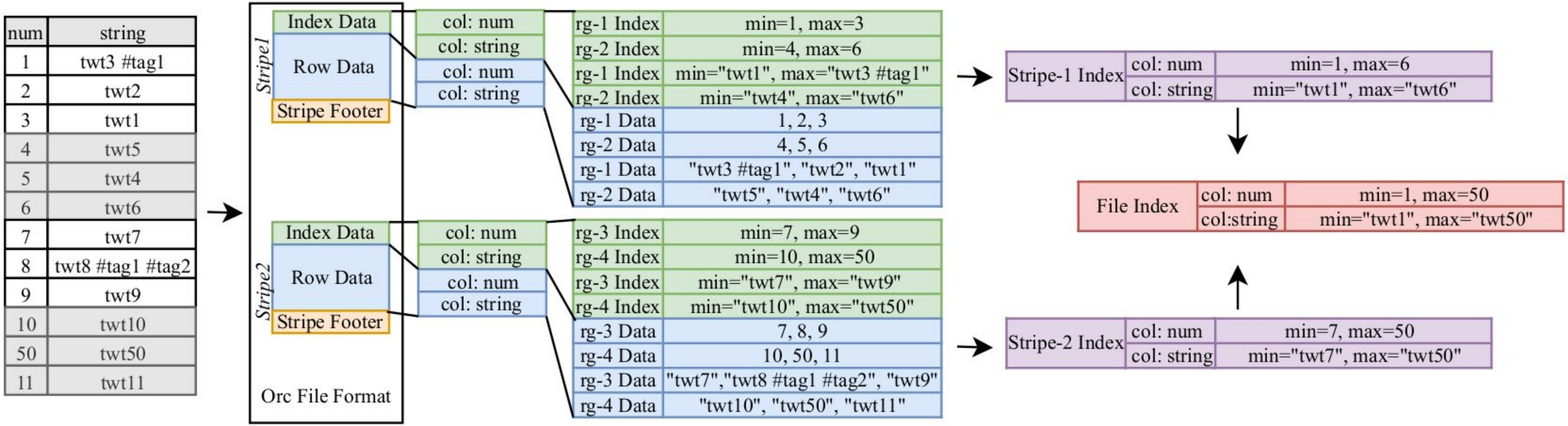
1. Min-max based indices

P1: Apache Orc



1. Min-max based indices
   a. Possibility of false positives
   b. No way to index substrings

P1: Apache Orc



1. Min-max based indices
   a. Possibility of false positives
   b. No way to index substrings
2. Queries that are optimized
   a. SELECT tweet FROM table WHERE col like “%#tag1%”
   b. SELECT tweet FROM table WHERE col like “%#tag1%” AND/OR “%#tag2%”

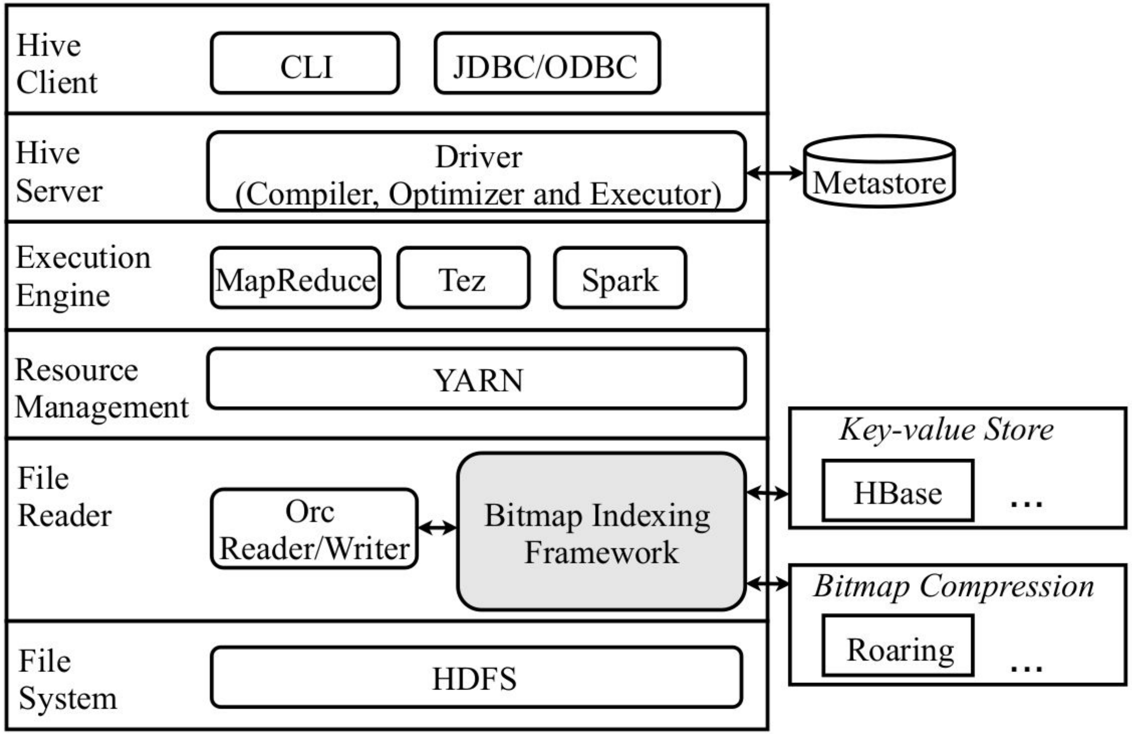
P1: Background
Apache Hive

1. Data warehouse solution running on Hadoop.
2. Allows users to use the query language HiveQL to write, read and manage datasets in distributed storage structures.
3. Allows creation of Orc based tables.

Apache HBase

1. Column oriented key-value store.
2. The major operations that define a key-value database are **put**(key, value), **get**(key) and **delete**(key).
3. High throughput and low input/output latency

P1: Lightweight Bitmap Indexing Framework



- The Orc reader/writer are modified to use our indexing framework.
- The key-value store and bitmap compression algorithm to use are easily replaceable.

P1: Framework Interface

Listing 1: Interface for Indexing framework

```
public interface IBitmapIndexingFramework {
    /* find indexable keys in column fields */
    String[] findKeys(String column);

    /* determine if search predicate is usable by framework */
    boolean isProcessable (String ast);

    /* create bitmap index from rownumber and column */
    boolean createBitmap(int rowNr, String column);

    /* store all key-bitmap pairs in key-value store */
    boolean storeKeyBitmap(String[] args);

    /* get bitmap index for a single key */
    byte[] getKeyBitmap(String[] args);
}
```

- Current implementation uses function to find hashtags, HBase for storage and Roaring bitmap for compression, users are free to use their own implementations
  - bitmap compression method
  - key-value store
  - method to find keys

P1: Framework Use in Hive

Listing 2: HiveQL for Bitmap Index creation/use

```
/* bitmap index creation */
CREATE TABLE tblOrc(id INT, tweet VARCHAR) STORED AS ORC;
SET hive.optimize.bitmapindex=true;
SET hive.optimize.bitmapindex.format=tblOrc/tweet/;
SET hive.optimize.bitmapindex.framework='com.BIFramework';
INSERT INTO tblOrc SELECT id, tweet FROM tblCSV;
/* bitmap index usage */
SET hive.optimize.ppd=true;
SET hive.optimize.index.filter=true;
SET hive.optimize.bitmapindex=true;
SELECT * FROM tblOrc WHERE tweet LIKE '%#tag%';
```

P1: Index Creation

1. Orc File
   a. Stripe (64 MB)
   b. Rowgroup (10,000 rows)
   c. Row (Rownumber)
2. To determine stripe number and rowgroup number from row number the number of rowgroups must be made consistent across stripes in a file.
3. Ghost rowgroups are added to stripes than contain less rowgroups than the maximum rowgroups per stripe.

P1: Index Creation

rownr	tweet
0	twt0 #tag1
1	twt1
2	twt2
3	twt3
4	twt4
5	twt5
6	twt6
7	twt7 #tag2
8	twt8
9	twt9
10	twt10
11	twt11
12	twt12
13	twt13
14	twt14
15	twt15
16	twt16 #tag1#tag2
17	twt17

(a) Sample dataset

str0	rg0	0	twt0 #tag1
		1	twt1
	rg1	2	twt2
		3	twt3
str1	rg0	4	twt4
		5	twt5
	rg1	6	twt6
		7	twt7 #tag2
str2	rg0	8	twt8
		9	twt9
	rg1	10	twt10
		11	twt11
	rg2	12	twt12
		13	twt13
str3	rg0	14	twt14
		15	twt15
	rg1	16	twt16 #tag1 #tag2
		17	twt17

(b) Sample dataset stored in Orc

P1: Index Creation

rownr	tweet
0	twt0 #tag1
1	twt1
2	twt2
3	twt3
4	twt4
5	twt5
6	twt6
7	twt7 #tag2
8	twt8
9	twt9
10	twt10
11	twt11
12	twt12
13	twt13
14	twt14
15	twt15
16	twt16 #tag1#tag2
17	twt17

(a) Sample dataset

str0	rg0	0	twt0 #tag1
		1	twt1
	rg1	2	twt2
		3	twt3
str1	rg0	4	twt4
		5	twt5
	rg1	6	twt6
		7	twt7 #tag2
str2	rg0	8	twt8
		9	twt9
	rg1	10	twt10
		11	twt11
	rg2	12	twt12
		13	twt13
str3	rg0	14	twt14
		15	twt15
	rg1	16	twt16 #tag1 #tag2
		17	twt17

(b) Sample dataset stored in Orc

str0	rg0	0	twt0 #tag1
		1	twt1
	rg1	2	twt2
		3	twt3
	grg2	4	
		5	
str1	rg0	6	twt4
		7	twt5
	rg1	8	twt6
		9	twt7 #tag2
	grg2	10	
		11	
str2	rg0	12	twt8
		13	twt9
	rg1	14	twt10
		15	twt11
	rg2	16	twt12
		17	twt13
str3	rg0	18	twt14
		19	twt15
	rg1	20	twt16 #tag1 #tag2
		21	twt17
	grg2	22	
		23	

(c) Sample dataset stored in Orc with ghost rowgroups

P1: Index Creation

rownr	tweet
0	twt0 #tag1
1	twt1
2	twt2
3	twt3
4	twt4
5	twt5
6	twt6
7	twt7 #tag2
8	twt8
9	twt9
10	twt10
11	twt11
12	twt12
13	twt13
14	twt14
15	twt15
16	twt16 #tag1#tag2
17	twt17

(a) Sample dataset

str0	rg0	0	twt0 #tag1
		1	twt1
	rg1	2	twt2
		3	twt3
str1	rg0	4	twt4
		5	twt5
	rg1	6	twt6
		7	twt7 #tag2
str2	rg0	8	twt8
		9	twt9
	rg1	10	twt10
		11	twt11
	rg2	12	twt12
		13	twt13
str3	rg0	14	twt14
		15	twt15
	rg1	16	twt16 #tag1 #tag2
		17	twt17

(b) Sample dataset stored in Orc

str0	rg0	0	twt0 #tag1
		1	twt1
	rg1	2	twt2
		3	twt3
	grg2	4	
		5	
str1	rg0	6	twt4
		7	twt5
	rg1	8	twt6
		9	twt7 #tag2
	grg2	10	
		11	
str2	rg0	12	twt8
		13	twt9
	rg1	14	twt10
		15	twt11
	rg2	16	twt12
		17	twt13
str3	rg0	18	twt14
		19	twt15
	rg1	20	twt16 #tag1 #tag2
		21	twt17
	grg2	22	
		23	

(c) Sample dataset stored in Orc with ghost rowgroups

stripe	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3
rowgroup	0	0	1	1	2	2	0	0	1	1	2	2	0	0	1	1	2	2	0	0	1	1	2	2
rownr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
#tag1	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
#tag2	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0

(d) Bitmap representation

P1: Index Creation

rownr	tweet
0	twt0 #tag1
1	twt1
2	twt2
3	twt3
4	twt4
5	twt5
6	twt6
7	twt7 #tag2
8	twt8
9	twt9
10	twt10
11	twt11
12	twt12
13	twt13
14	twt14
15	twt15
16	twt16 #tag1#tag2
17	twt17

(a) Sample dataset

str0	rg0	0	twt0 #tag1
		1	twt1
	rg1	2	twt2
		3	twt3
str1	rg0	4	twt4
		5	twt5
	rg1	6	twt6
		7	twt7 #tag2
str2	rg0	8	twt8
		9	twt9
	rg1	10	twt10
		11	twt11
	rg2	12	twt12
		13	twt13
str3	rg0	14	twt14
		15	twt15
	rg1	16	twt16 #tag1 #tag2
		17	twt17

(b) Sample dataset stored in Orc

str0	rg0	0	twt0 #tag1
		1	twt1
	rg1	2	twt2
		3	twt3
	grg2	4	
		5	
str1	rg0	6	twt4
		7	twt5
	rg1	8	twt6
		9	twt7 #tag2
	grg2	10	
		11	
str2	rg0	12	twt8
		13	twt9
	rg1	14	twt10
		15	twt11
	rg2	16	twt12
		17	twt13
str3	rg0	18	twt14
		19	twt15
	rg1	20	twt16 #tag1 #tag2
		21	twt17
	grg2	22	
		23	

(c) Sample dataset stored in Orc including ghost rowgroups

stripe	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3
rowgroup	0	0	1	1	2	2	0	0	1	1	2	2	0	0	1	1	2	2	0	0	1	1	2	2
rownr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
#tag1	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
#tag2	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0

(d) Sample dataset stored in Orc with ghost rowgroups

Key	Value	
	WorkerNode-OrcFilename	..
mrgps	3	..
#tag1	Roaring(**1**,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**1**,0,0,0)	..
#tag2	Roaring(0,0,0,0,0,0,0,0,0,**1**,0,0,0,0,0,0,0,0,0,0,**1**,0,0,0)	..

(e) Key and bitmaps

P1: Query processing using Bitmap Indices

- SELECT tweet FROM tweets WHERE tweet like “%#tag1%” OR tweet like “%#tag2%”
  - **Predicate:** tweet like “%#tag1%” OR tweet like “%#tag2%”
    - #tag1 = RoaringBitmap(Stripe0, Stripe1,...,StripeN)
    - #tag2 = RoaringBitmap(Stripe0, Stripe1,...,StripeN)
    - maxRowgroupsPerStripe = value
    - rowsPerRowGroup = 10000
  - **Stripes:** (Stripe0, Stripe1,...)
  - **Slice:** Slice(bitmap, StartStripe, EndStripe)
    - Slice(#tag1, 0, 1) and Slice(#tag2, 0, 1)
    - #tag1 = RoaringBitmap(Stripe0, Stripe1)
    - #tag2 = RoaringBitmap(Stripe0, Stripe1)
    - resultBitmap = #tag1 OR #tag2
  - Calculate Stripes and Rowgroups
    - Calc(resultBitmap, maxRowgroupsPerStripe, rowsPerRowgroup)

P1: Experiments

1. Distributed cluster on Microsoft Azure
   a. 1 master and 7 nodes as slaves.
   b. Ubuntu OS with 4 VCPUS, 8 GB memory, 192 GB SSD
   c. Hive 2.2.0, HDFS 2.7.4 and HBase 1.3.1
2. Datasets
   a. Three datasets: 55GB, 110GB and 220GB. Pattern in results were similar
   b. Schema for the datasets contains 13 attributes
      [tweetYear, tweetNr, userIdNr, username, userId, latitude, longitude, tweetSource, reTweetUserIdNr, reTweetUserId, reTweetNr, tweetTimeStamp, **tweet**]

Dataset	Tuples	Total HashTags	Unique Hastags	Orc Files	Stripes	Rowgroups
Tweets55	192,665,259	32,534,370	5,363,727	66	285	19,360
Tweets110	381,478,160	62,281,496	9,063,962	128	624	38,351
Tweets220	765,196,395	126,603,736	16,149,621	224	1342	76,918

P1: Queries Used

LIKE:

```
SELECT tweetSource, COUNT(*) as Cnt
FROM TableName
WHERE tweet LIKE '%hashtag1%'
GROUP BY tweetSource;
```

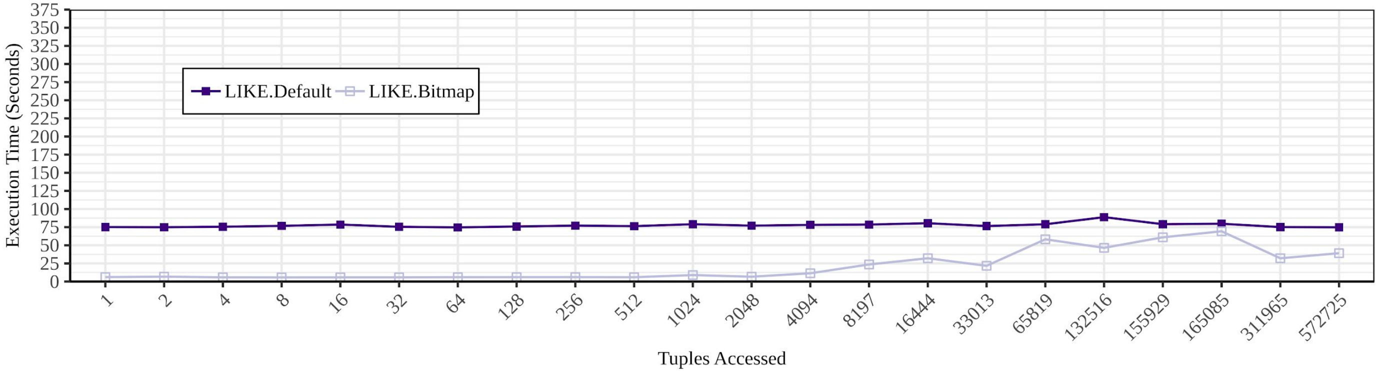
OR-LIKE:

```
SELECT tweetSource, COUNT(*) as Cnt
FROM TableName
WHERE (tweet LIKE '%hashtag1%' OR tweet LIKE '%hashtag2%',...)
GROUP BY tweetSource;
```

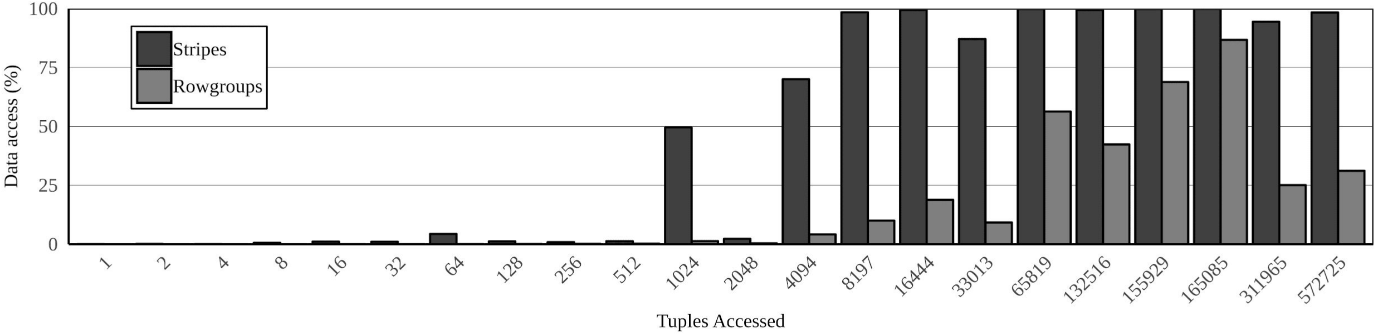
JOIN:

```
SELECT t1.tweetSource, COUNT(*) as Cnt
FROM TableName AS t1 JOIN TableName AS t2  JOIN (t1.tweetNr = t2.reTweetNr)
WHERE t1.tweetNr != -1
AND (t1.tweet LIKE '%hashtag1%')
AND (t2.tweet LIKE '%hashtag1%')
GROUP BY t1.tweetSource;
```

P1: LIKE Queries

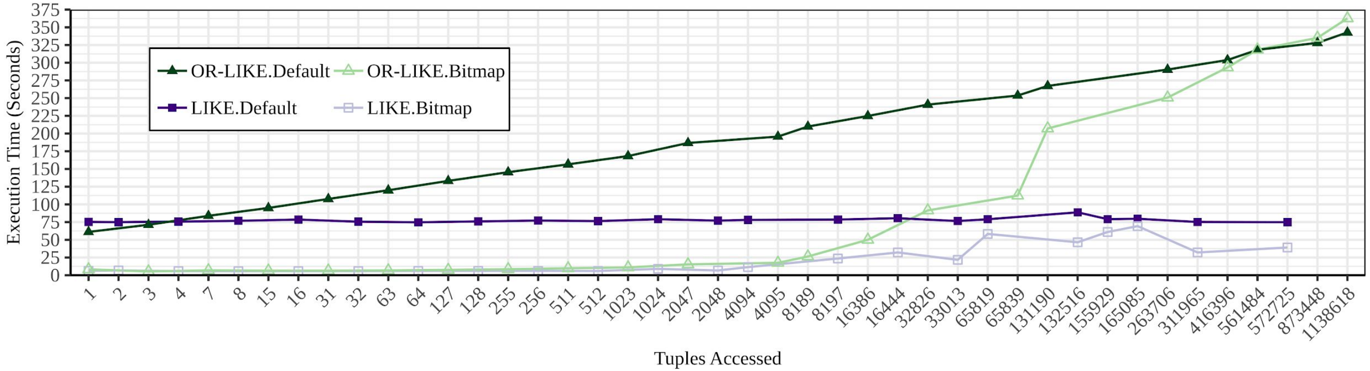


(a) Execution times for LIKE queries on Tweets220

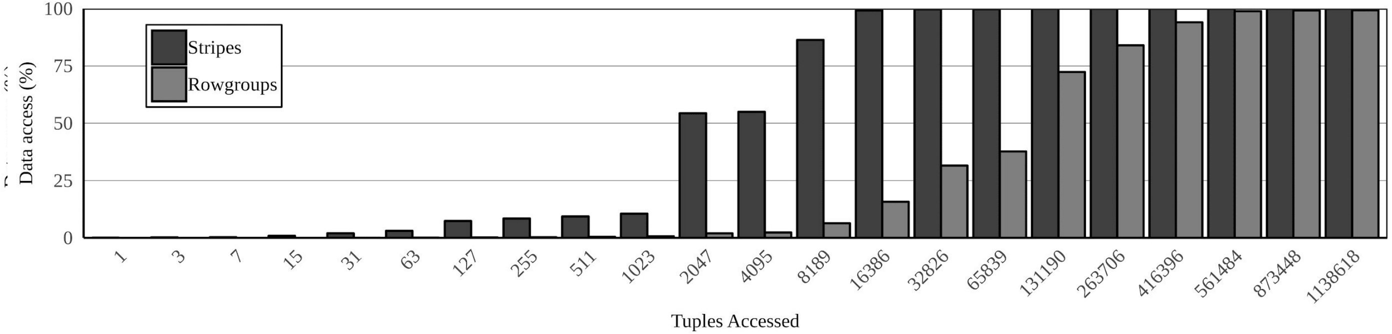


(b) Stripes/Rowgroups accessed by LIKE queries on Tweets220

P1: LIKE and OR-LIKE Queries

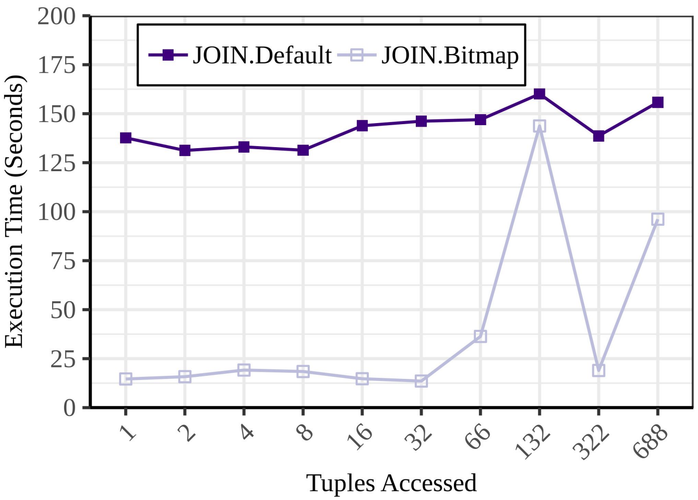


(a) Execution times for LIKE and OR-LIKE queries on Tweets 220

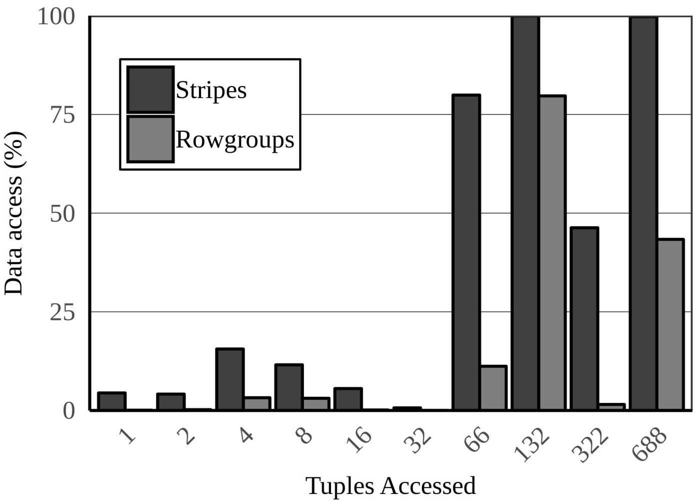


(b) Stripes/Rowgroups accessed by OR-LIKE queries on Tweets220

P1: JOIN Queries

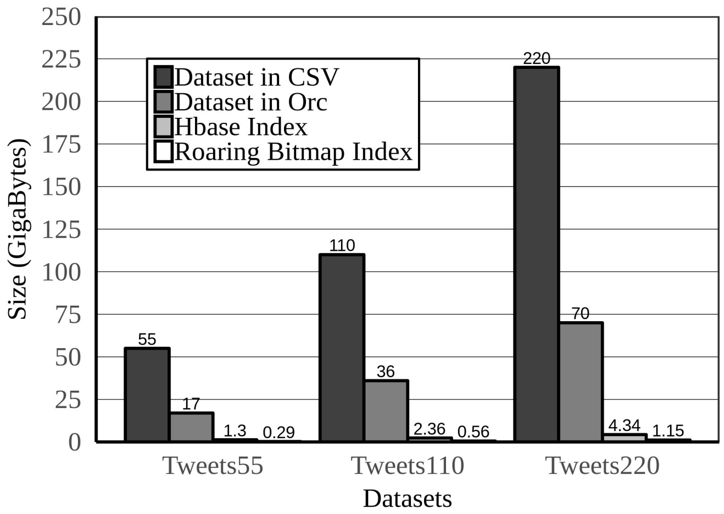


(a) Execution times for JOIN queries on Tweets220

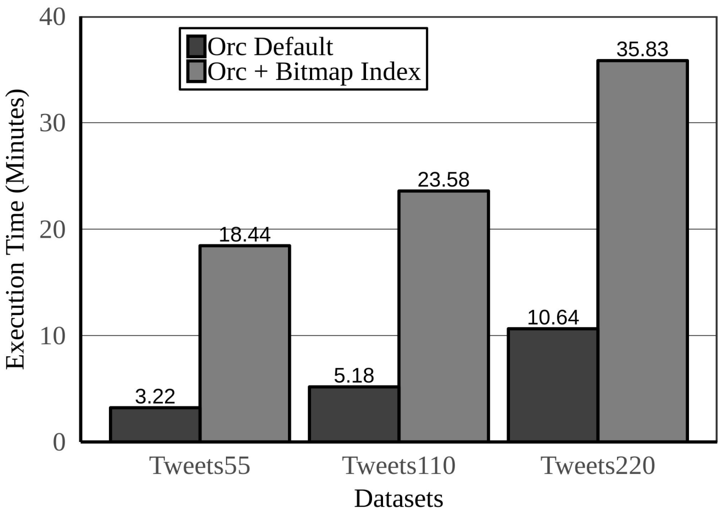


(b) Stripes/Rowgroups accessed by JOIN queries on Tweets220

P1: Index Creation Times and Sizes



(a) Tweets datasets and their Index sizes



(b) Index creation times for Tweets datasets

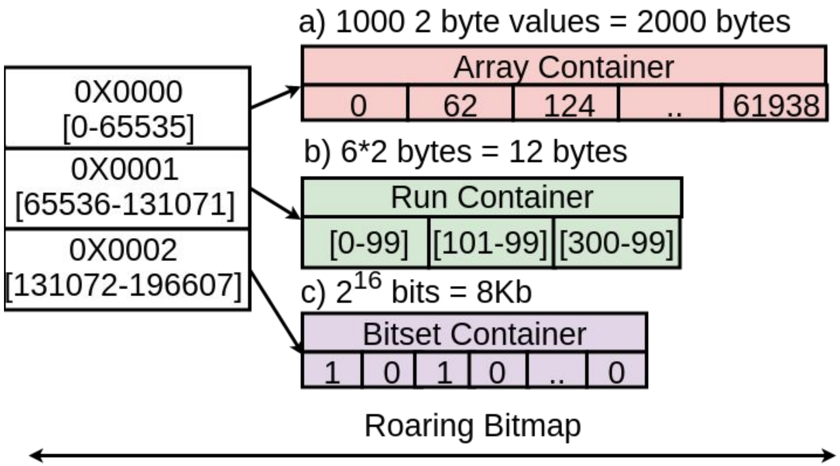
- Size of bitmap indices and the the Hbase table where they are stored are substantially smaller their Orc based tables.

- Runtime overhead due to the index creation process.

P2: Bitmap Indexing with Storage Structure Considerations

- Issues with Roaring Bitmap

  1) Loss of Storage structure information

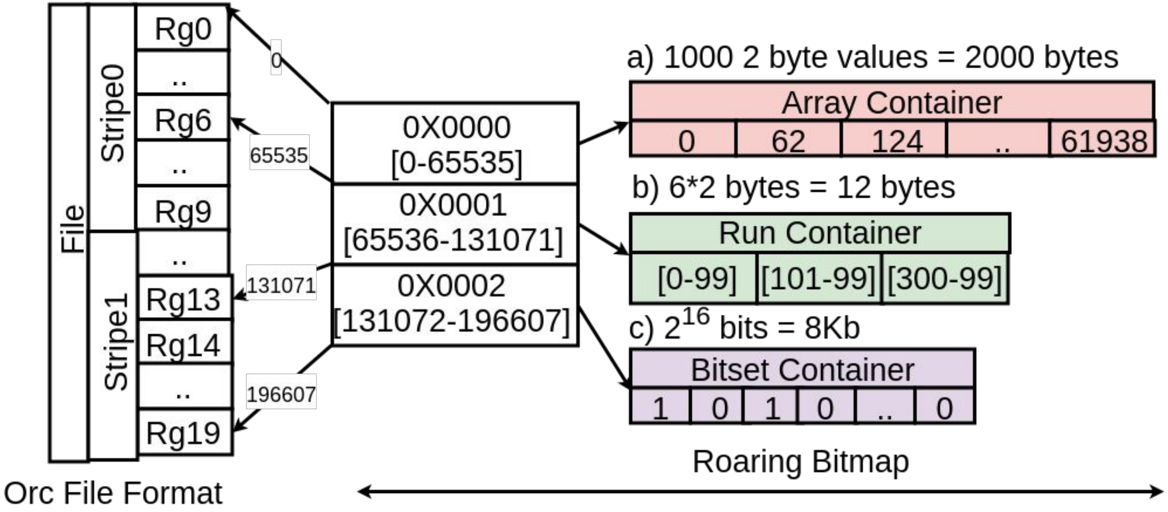


P2: Bitmap Indexing with Storage Structure Considerations

- Issues with Roaring Bitmap

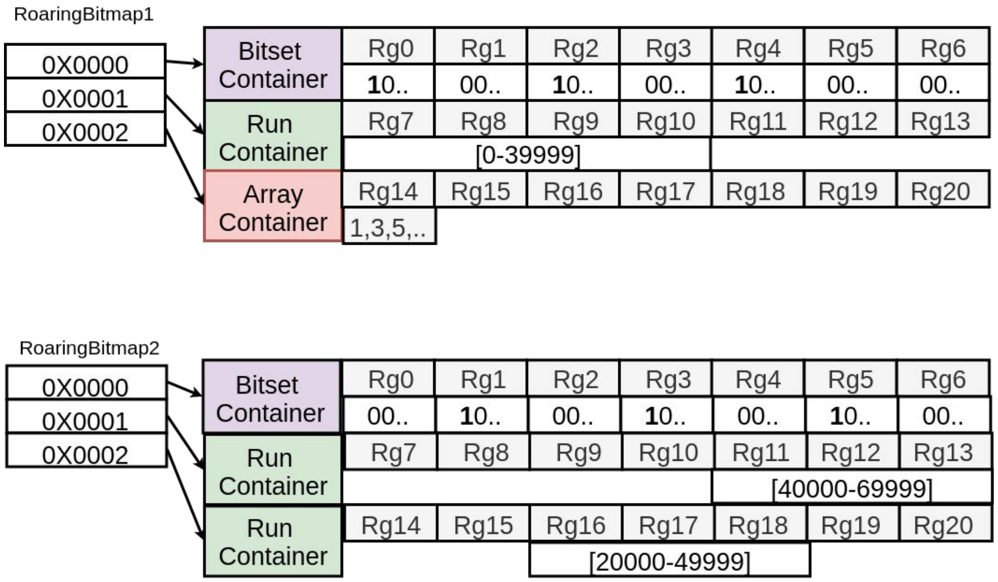
  1) Loss of Storage structure information

  - Expensive to map from row number to block number
  - [1, 5, 500, 9999, 11000, 15000] -> [Rg0, Rg1]



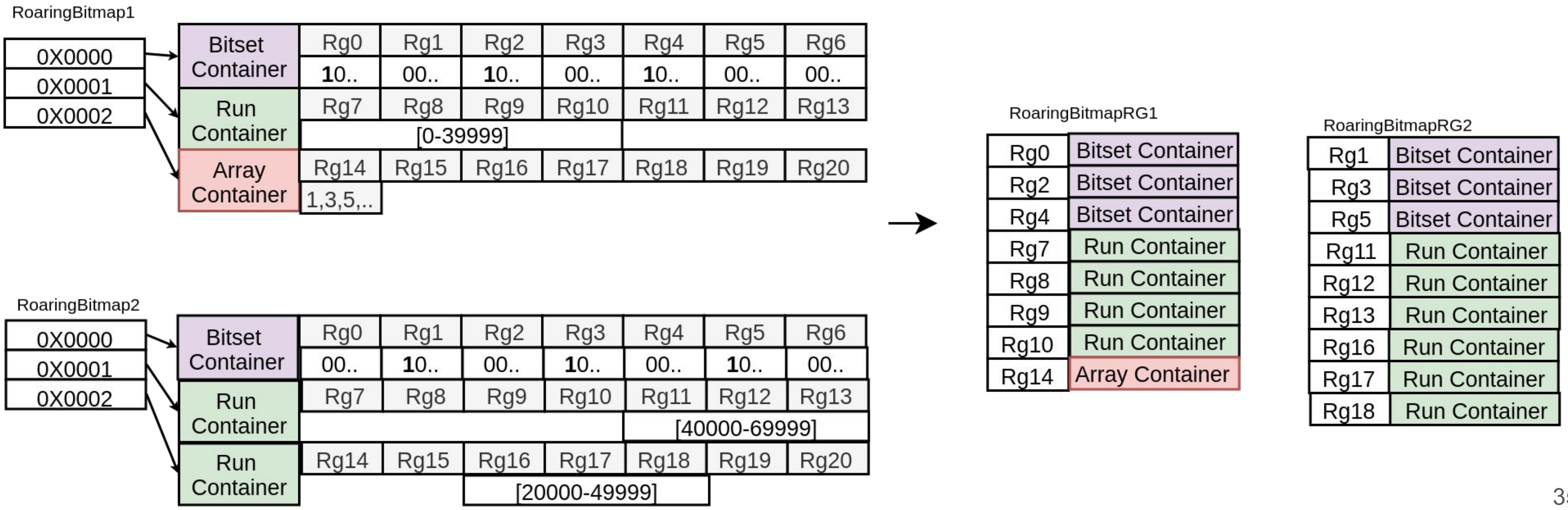
P2: Bitmap Indexing with Storage Structure Considerations

- Issues with Roaring Bitmap
  - Possibility of false positives



P2: Explored Solutions

- Containers set to use Storage structure information
- However, more containers than Roaring bitmaps



P2: Datasets

- Publicly available dataset provide by [3]

	Number of Bitmaps	Universe Size	Average count per bitmap
CENSUS_INCOME	200	199,523	34,610.1
CENSUS_INCOME_SRT	200	199,523	30,464.3
CENSUS1881	200	4,277,806	5019.3
CENSUS1881_SRT	200	4,277,735	3404.0
WEATHER_SEPT_85	200	1,015,367	64,353.1
WEATHER_SEPT_85_SRT	200	1,015,367	80,540.5
WIKILEAKS_NOQUOTES	200	1,353,179	1376.8
WIKILEAKS_NOQUOTES_SRT	200	1,353,133	1440.1

P2: Preliminary Results (AND)

1. Experiments
   a. Performed on my laptop
   b. Throughput
2. AND
   a. AND operation between 200 bitmaps
3. AND + RG
   a. Calculate operation between 200 bitmap + mapping from rownumber to rowgroups

	AND		AND + RG Calculate	
	Roaring	RoaringRG	Roaring	RoaringRG
CENSUS_INCOME	**858.174 ±6 ops/s**	581.037 ±53 ops/s	82.845 ±2 ops/s	**612.398 ±13 ops/s**
CENSUS_INCOME_SRT	2176.746 ±8 ops/s	**2584.645 ±32 ops/s**	143.931 ±6 ops/s	**2538.521 ±26 ops/s**
CENSUS1881	25523.716 ±185 ops/s	**27693.969 ±378** ops/s	26380.113 ±264 ops/s	**28524.012 ±371 ops/s**
CENSUS1881_SRT	148118.962 ±1177 ops/s	**211173.6 4 ±6790 ops/s**	135202.337 ±1168 ops/s	**213418.927 ±2989 ops/s**
WEATHER_SEPT_85	**195.189 ±2 ops/s**	151.188 ±3 ops/s	31.240 ±0.2 ops/s	**155.313 ±2 ops/s**
WEATHER_SEPT_85_SRT	**1873.571 ±14 ops/s**	1646.546 ±90 ops/s	97.263 ±4 ops/s	**1652.788 ±8 ops/s**
WIKILEAKS_NOQUOTES	**11604.997 ±84 ops/s**	9504.886 ±541 ops/s	3328.400±62ops/s	**8683.909 ±11 ops/s**
WIKILEAKS_NOQUOTES_SRT	71065.61 ±1507 ops/s	**72055.737 ±1046 ops/s**	**64263.370 ±1143 ops/s**	63226.929 ±876 ops/s

P2: Preliminary Results (OR)

1. OR
   a. OR operation between 200 bitmaps
2. OR + RG Calculate
   a. OR operation between 200 bitmap + mapping from rownumber to rowgroups

	OR		OR + RG Calculate	
	Roaring	RoaringRG	Roaring	RoaringRG
CENSUS_INCOME	**621.432 ±3.09 ops/s**	327.162 ±3.412 ops/s	9.555 ±0.2 ops/s	**342.015 ±2 ops/s**
CENSUS_INCOME_SRT	1049.444 ±12.317 ops/s	**1313.316 ±5.363 ops/s**	9.854 ±0.3 ops/s	**1291.065 ±16 ops/s**
CENSUS1881	**1748.733 ±145.411 ops/s**	1743.181 ±18.263 ops/s	4.773 ±0.02 ops/s	**1589.701 ±66 ops/s**
CENSUS1881_SRT	11178.567 ±50.238 ops/s	**11816.204 ±47.35** ops/s	31.154 ±0.4 ops/s	**8486.935 ±42 ops/s**
WEATHER_SEPT_85	**147.944 ±1.4 ops/s**	84.647 ±1.603 ops/s	1.360 ±0.1 ops/s	**83.673 ±0.3 ops/s**
WEATHER_SEPT_85_SRT	**874.785 ±1.209 ops/s**	840.423 ±4.253 ops/s	1.300 ±0.1 ops/s	**813.437 ±6 ops/s**
WIKILEAKS_NOQUOTES	**3810.786 ±52.963 ops/s**	3541.734 ±27.676 ops/s	52.897 ±0.2 ops/s	**2493.273 ±21 ops/s**
WIKILEAKS_NOQUOTES_SRT	**15858.955 ±67.97 ops/s**	10791.506 ±138.839 ops/s	146.613 ±2 ops/s	**7903.992 ±29 ops/s**

P2: Ongoing Work

1. Mapping from rownumber to rowgroup
   a. [1, 5, 500, 9999, 11000, 15000] -> [Rg0, Rg1]
   b. Is there a better approach?

2. Comparison of Memory consumption Roaring vs RoaringRG
   a. RoaringRG uses more containers

Remaining Publications:

P3: An Adaptive Bitmap Indexing Scheme for Distributed Environments

a. Index creation is expensive
b. What do you index
c. Index might be only be used a fraction of the time
d. Adaptively build the index

P5: Bitmap Indexing on Distributed Environments

a. Work from paper 1, 2 and 3
b. Efficient updates of bitmap indices

P6: DBIF: A demonstration of DBIF on Big Data

a. Demonstration of our indexing framework

P4: Multidimensional Online Analytical Processing on Cell Stores

1. Cell Stores
   a. Disclaimer: Concept paper on ArXiv [Not peer-reviewed]
   b. Cells viewed as atom of data
   c. Cells can be converted into cubes or spreadsheets
2. Support Cell Stores on our framework.

Dimension	Value
Concept	Equity
Period	Dec. 31st, 2012
Entity	Championcard
Unit	US Dollars
5,000,000,000	

P4:

Dimension	Value
Concept	Equity
Period	Dec. 31st, 2012
Entity	Championcard
Unit	US Dollars
5,000,000,000	

....

Dimension	Value
Concept	Liabilities
Period	Dec. 31st, 2012
Entity	American Rapid
Unit	US Dollars
3,000,000,000	

a) Cells

Dimension	Value
Concept	Assets, Equity, Liabilities
Period	Sept. 30th, 2012, Dec. 31st, 2012
Entity	Visto, Championcard, American Rapid
Unit	US Dollars

b) Hypercube

Concept	Period	Entity	Unit	Region	Value
Assets	Sept. 30th, 2012	Visto	USD	United States	3,000,000,000
Assets	Sept. 30th, 2012	Visto	USD	[World]	4,000,000,000
Assets	Sept. 30th, 2012	Championcard	USD	United States	6,000,000,000
Assets	Sept. 30th, 2012	Championcard	USD	[World]	8,000,000,000
Assets	Sept. 30th, 2012	American Rapid	USD	United States	5,000,000,000
Assets	Sept. 30th, 2012	American Rapid	USD	[World]	9,000,000,000
Equity	Sept. 30th, 2012	Visto	USD	United States	2,000,000,000
Equity	Sept. 30th, 2012	Visto	USD	[World]	3,000,000,000
Equity	Sept. 30th, 2012	Championcard	USD	United States	4,000,000,000
Equity	Sept. 30th, 2012	Championcard	USD	[World]	5,000,000,000
Equity	Sept. 30th, 2012	American Rapid	USD	United States	3,000,000,000
Equity	Sept. 30th, 2012	American Rapid	USD	[World]	6,000,000,000
Liabilities	Sept. 30th, 2012	Visto	USD	United States	1,000,000,000
Liabilities	Sept. 30th, 2012	Visto	USD	[World]	1,000,000,000
Liabilities	Sept. 30th, 2012	Championcard	USD	United States	2,000,000,000
Liabilities	Sept. 30th, 2012	Championcard	USD	[World]	3,000,000,000
Liabilities	Sept. 30th, 2012	American Rapid	USD	United States	2,000,000,000
Liabilities	Sept. 30th, 2012	American Rapid	USD	[World]	3,000,000,000

c) Materialized Hypercube

PhD Courses

General

Course	Organizer	ECTS	Status
Danish Language	**AAU**	2	Fall 16/ **Compete**
Introduction to the PhD Study	**AAU**	1	Spring 16/ **Complete**
Writing and Reviewing Scientific Papers	**AAU**	3.75	Spring 16/ **Complete**
Professional Communication Skills	**AAU**	2.75	Fall 16/ **Complete**
Library Information Management	**AAU**	1	Spring 17/ **Complete**
Spanish Language	**UPC**	2	To be decided
To be decided	**UPC**	2	To be decided
Project Management and Interpersonal skills	**AAU**	2	Fall 19/ Planned
Total		**16.5**	

PhD Courses

Project Related

Course	Organizer	ECTS	Status
Business Intelligence Study Group	**AAU**	2	Fall 16/ **Compete**
Integrated Analytics on Big Data	**AAU**	2	Fall 16/ **Complete**
Scalable Tools for Linked Data Analytics	**AAU**	2	Fall 16/ **Complete**
EBISS summer school (Attendance)	**AAU**	2	Fall 16/ **Complete**
Big Data management on Modern Hardware	**AAU**	2	Spring 17/ **Complete**
EBISS Summer School (Participation)	**AAU**	2	In progress
Conference attendance	**tbd**	2	To be decided
Total		**14**	

Knowledge Dissemination

1. Project group supervision
    a. 12 groups (42 Students)
2. Teaching assistant for 2 semesters
    a. Database Development course
3. DOLAP 2019
    a. Lisbon, Portugal

Semester	Hours
Spring 2016	185
Fall 2016	165
Spring 2017	230
Fall 2018	105
Spring 2019	90
Total	**775**

References

[1] https://www.quintly.com/blog/instagram-study

[2] https://www.slideshare.net/Hadoop_Summit/orc-file-optimizing-your-big-data

[3] Kesheng Wu, Ekow J Otoo, and Arie Shoshani. 2006. Optimizing bitmap indices with efficient compression. ACM Transactions on Database Systems (TODS) 31, 1 (2006), 1–38.

[4] Lemire, D., Ssi‐Yan‐Kai, G., & Kaser, O. (2016). Consistently faster and smaller compressed bitmaps with roaring. Software: Practice and Experience, 46(11), 1547-1569.

Orc Index Processing

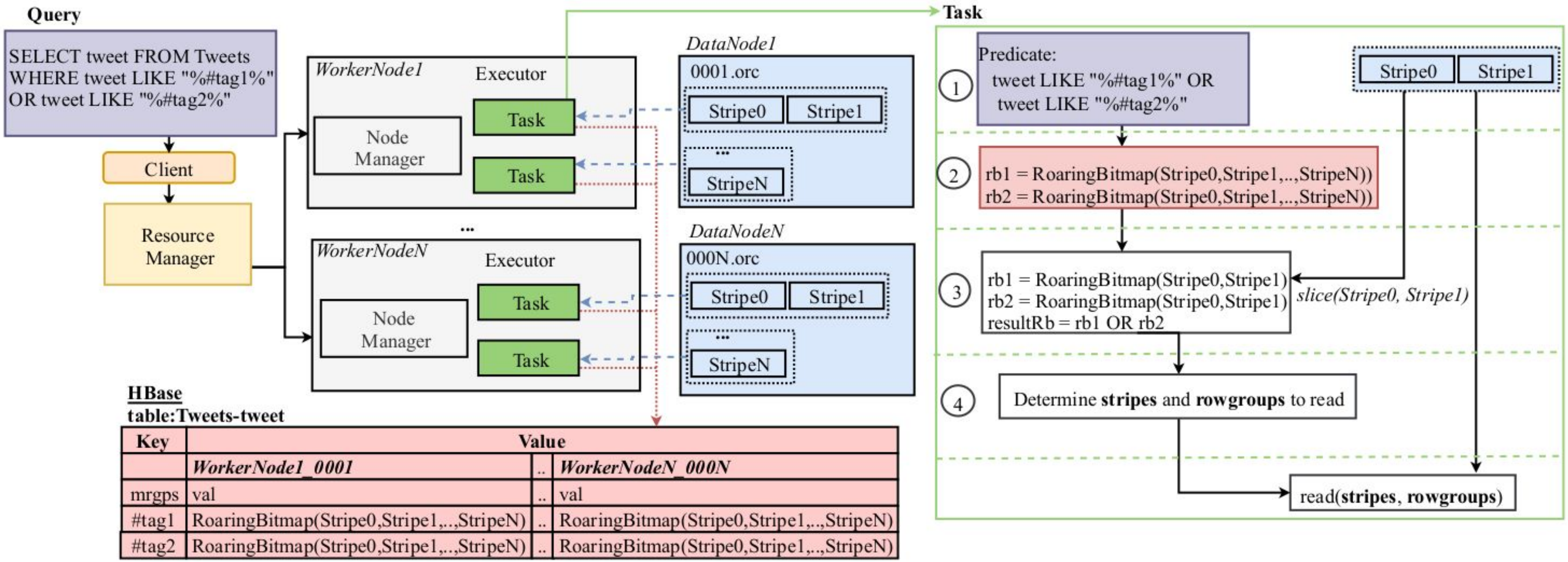


Figure 4: Orc Index Processing.

Stripe and Rowgroup Calculation

mrgps = maximum rowgroups per stripe ()

rprg = rows per rowgroup () and

rn = row number for a particular tuple (rn) can

str = stripe number

rg = rowgroup number

$$str = rn/(mrgps * rprg)$$
$$rg = (rn \bmod (mrgps * rprg))/rprg$$