Efficient Indexing of Hashtags using Bitmap Indices

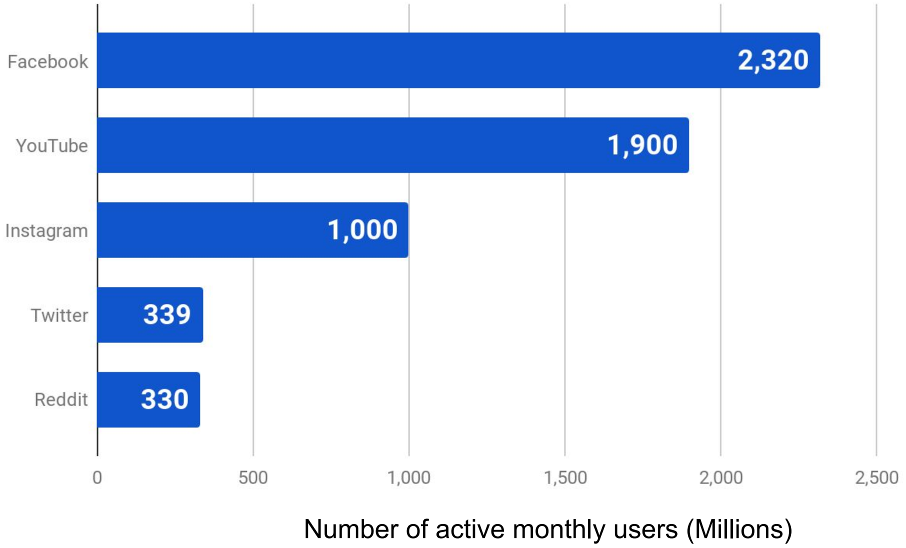
DOLAP 2019 - Lisbon (Portugal) - Mar 26, 2019
Lawan Subba, Christian Thomsen and Torben Bach Pedersen
Aalborg University, Denmark

Outline

1. Introduction
    a. Hashtags
2. Background
    a. Apache Orc
    b. Bitmap Index
    c. Apache Hive
    d. Apache HBase
3. Lightweight Bitmap Indexing Framework
    a. Framework Interface
    b. Framework Use in Hive
    c. Index Creation
    d. Query processing Using Bitmap Index
4. Experiments
5. Related Work
6. Conclusion

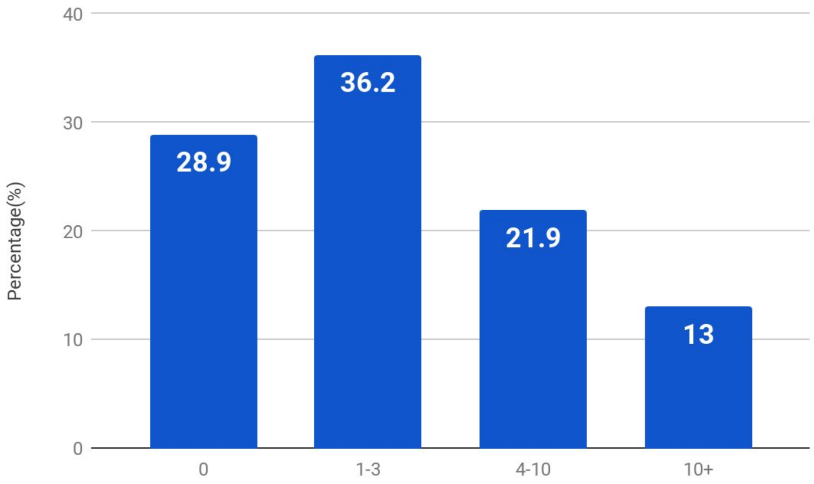
Introduction

- Social media platforms like Facebook, Instagram and Twitter have millions of active monthly users.
- Enormous amounts of data being generated regularly means that rapidly accessing relevant data from data stores is just as important as its storage.



Hashtags

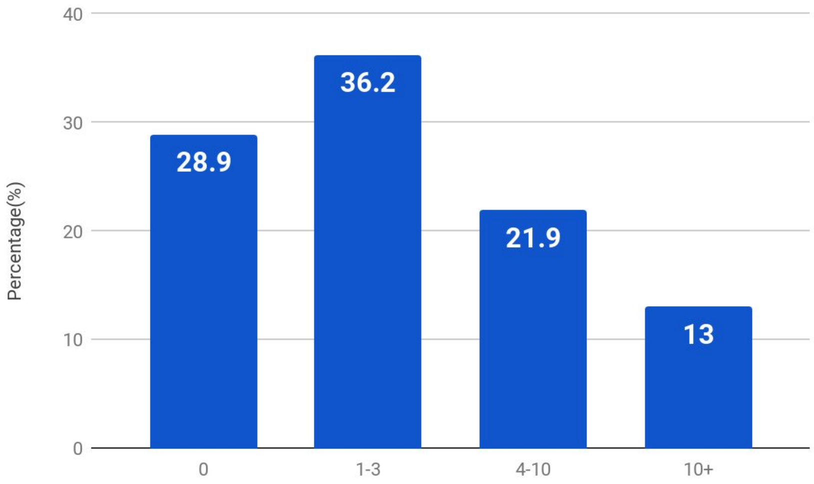
- A keyword containing numbers and letters preceded by a hash sign(#)
- Simplicity and lack of formal syntax



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

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 …)



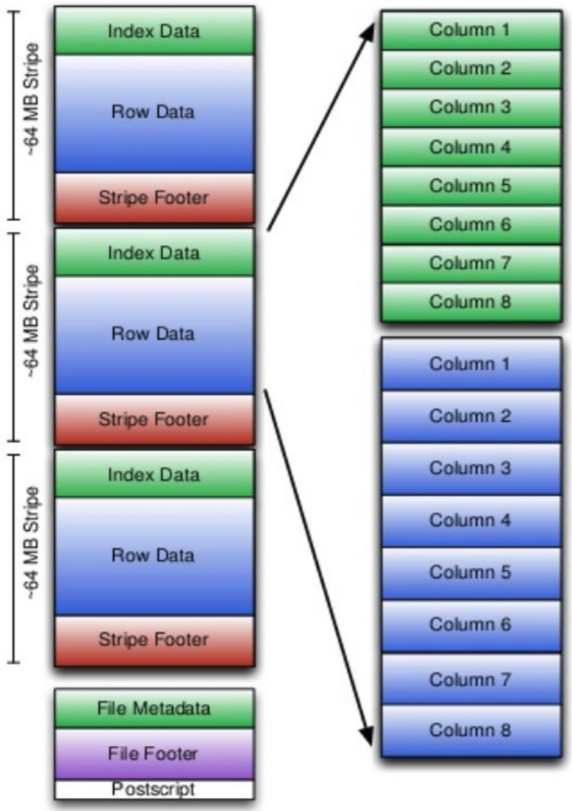
Distribution of Hashtags used in 8.9 million instagram posts in 2018 [1]

Contributions

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

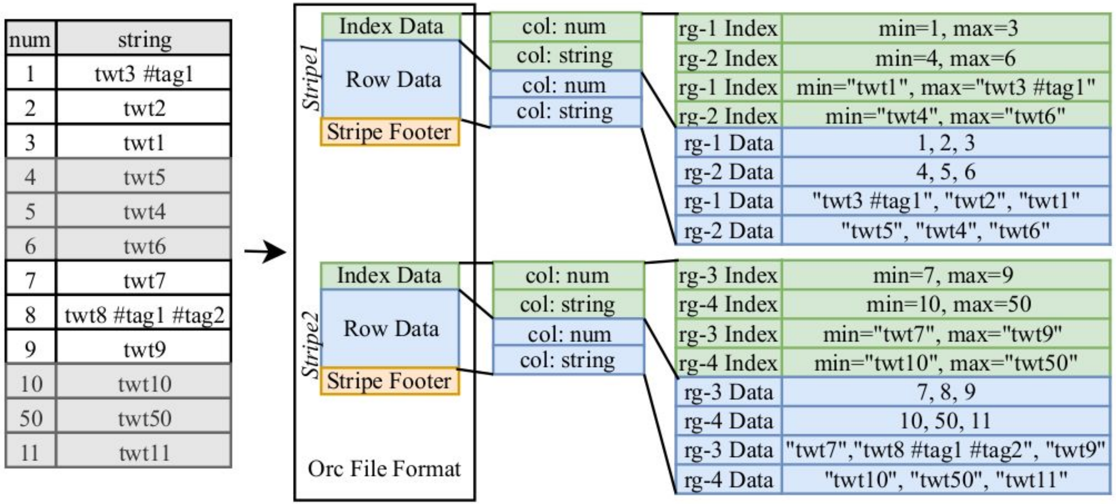
Apache Orc

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



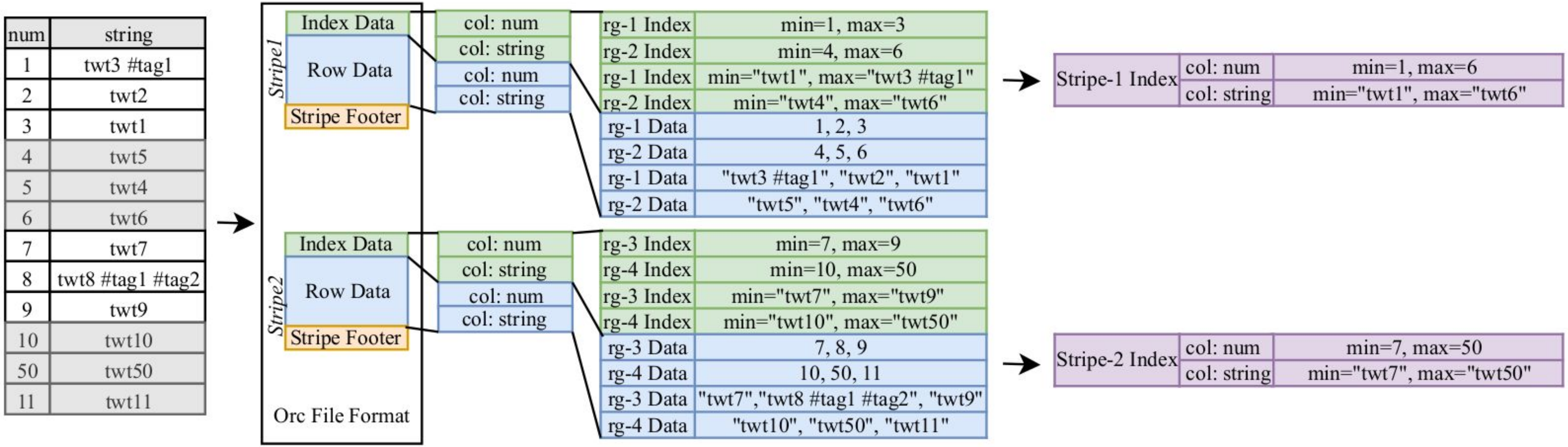
Orc File Format [2]

Apache Orc



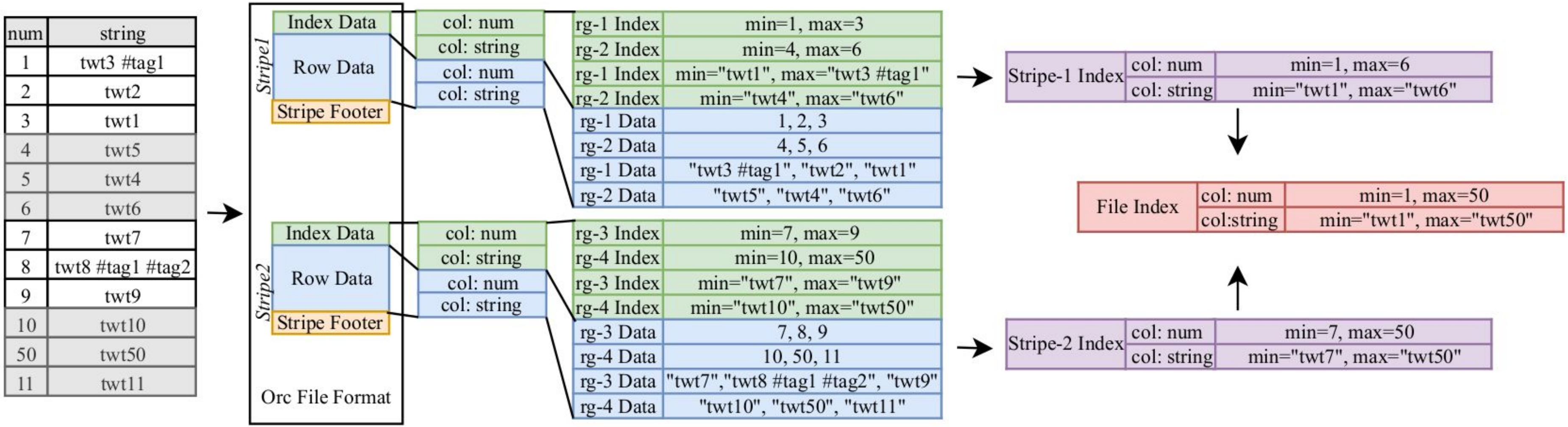
- Min-max based indices

Apache Orc



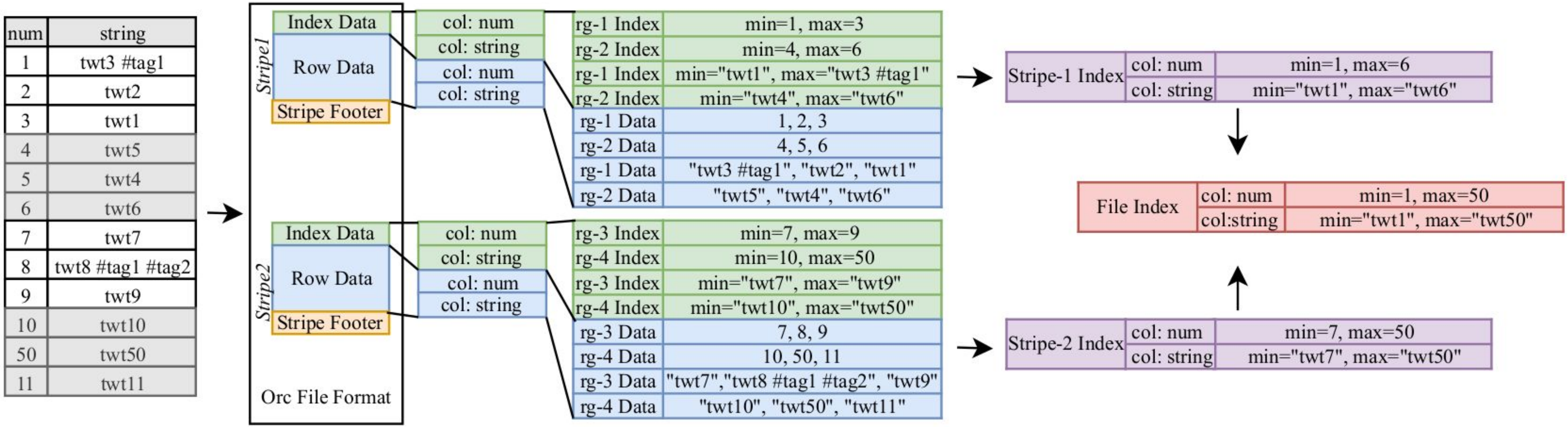
- Min-max based indices

Apache Orc



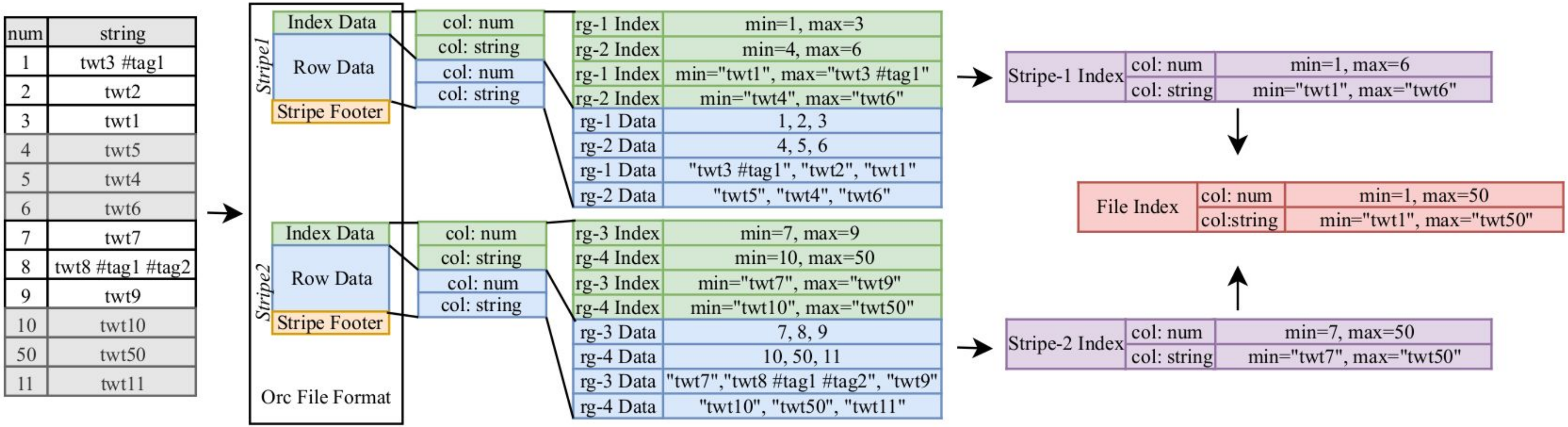
- Min-max based indices

Apache Orc



- Min-max based indices
    - Possibility of false positives
    - No way to index substrings

Apache Orc



- Min-max based indices
  - Possibility of false positives
  - No way to index substrings
- Queries
  - SELECT tweet FROM table WHERE col like “%#tag1%”
  - SELECT tweet FROM table WHERE col like “%#tag1%” AND/OR “%#tag2%”

Bitmap Index

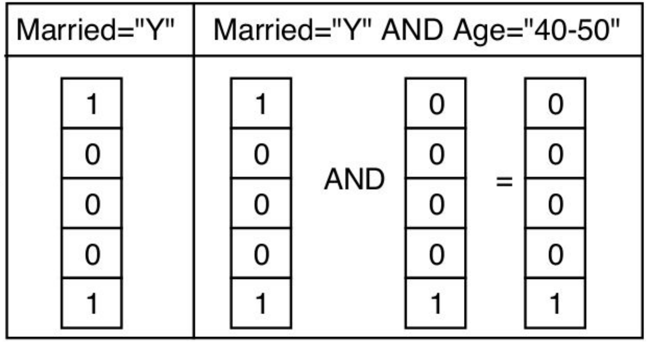
RowId	Name	Married	Age
1	Alice	Y	20-30
2	Bob	N	20-30
3	Carol	N	30-40
4	Dave	N	30-40
5	Ed	Y	40-50

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	



Roaring Bitmap

- Divides the data into chunks of (2^{16}=65536) integers (e.g., [0, 2^{16}), [2^{16}, 2 x 2^{16}), …).
- Each chunk can be stored in a uncompressed bitmap, a simple list of integers, or a list of runs.
- Fast random access.

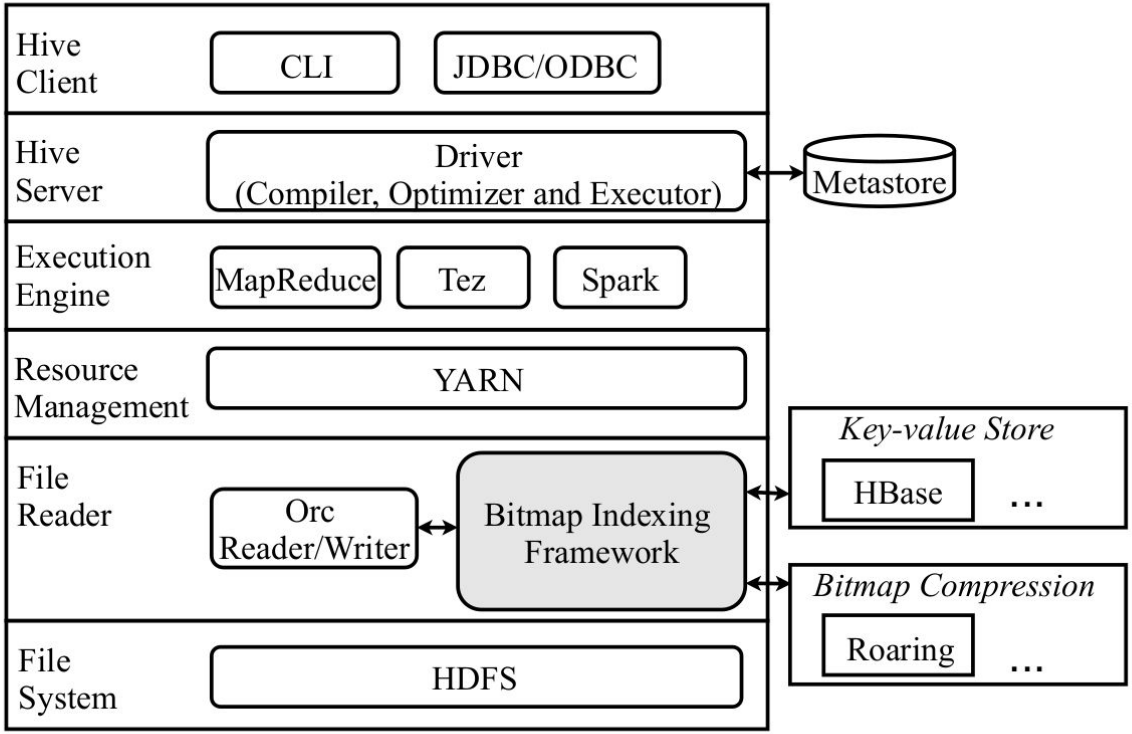
Apache Hive

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

Apache HBase

- Column oriented key-value store.
- The major operations that define a key-value database are **put**(key, value), **get**(key) and **delete**(key).
- Data in HBase is organized as labeled tables containing rows, each row is defined by a sorting key and an arbitrary number of columns.
- High throughput and low input/output latency

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.

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

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%';
```

Index Creation

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

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

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

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

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

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)

Experiments

- Distributed cluster on Microsoft Azure with 1 node acting as master and 7 nodes as slaves.
- Ubuntu OS with 4 VCPUS, 8 GB memory, 192 GB SSD
- Hive 2.2.0, HDFS 2.7.4 and HBase 1.3.1
- Datasets
  - Three datasets: 55GB, 110GB and 220GB
  - 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

Queries Used

LIKE:

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

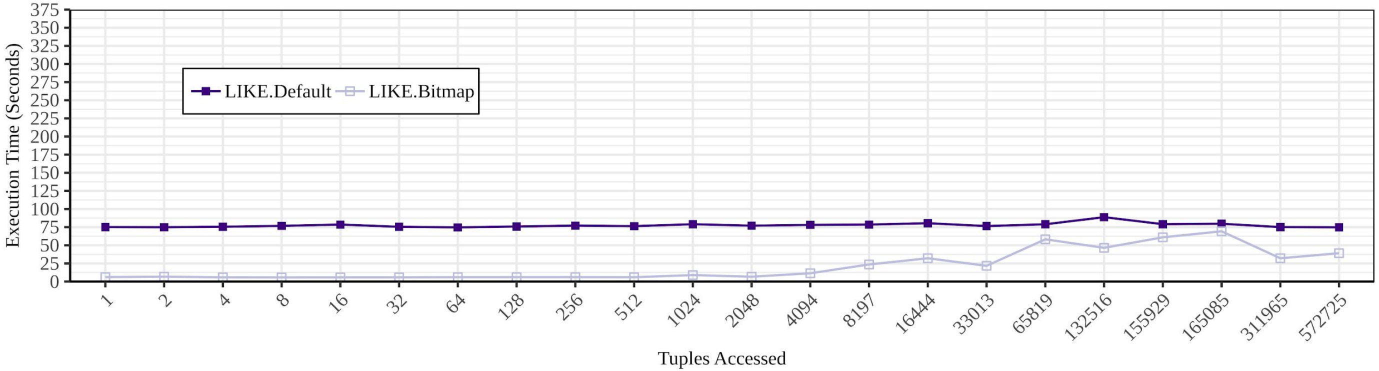
OR-LIKE:

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

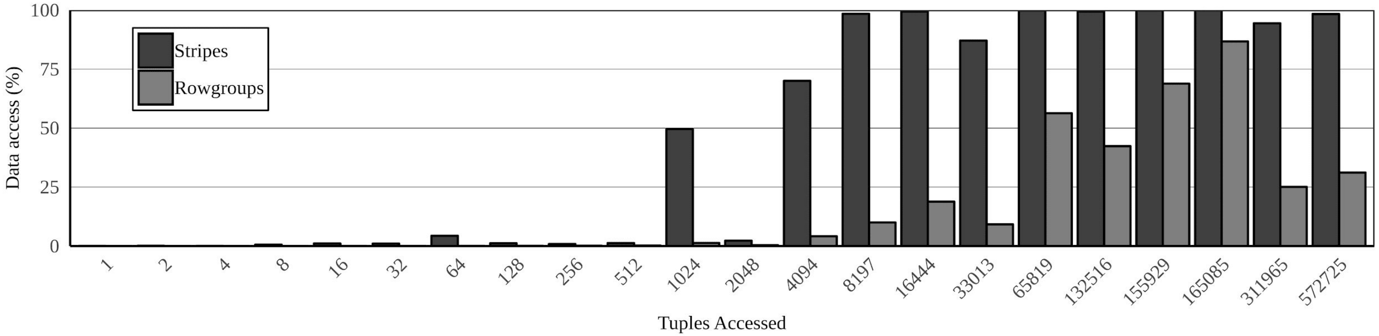
JOIN:

```
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;
```

LIKE Queries

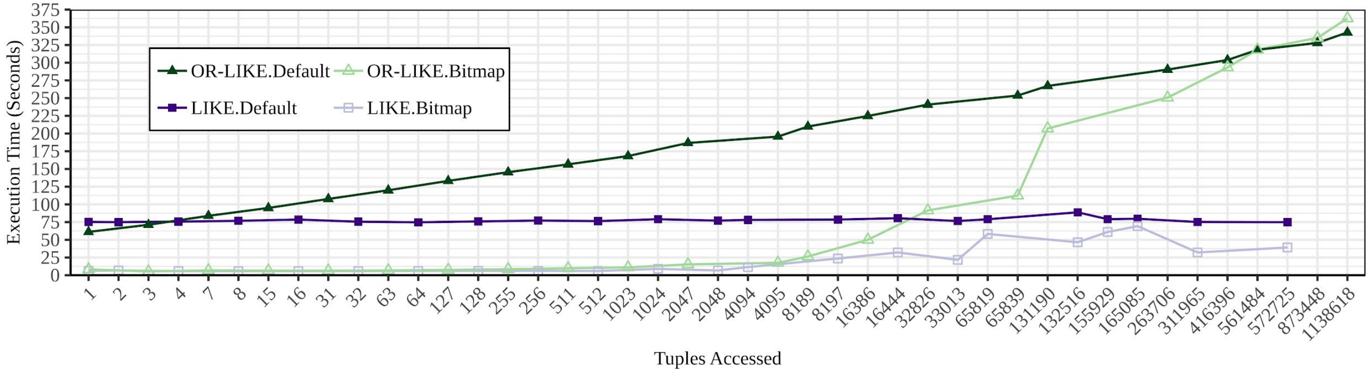


(a) Execution times for LIKE queries on Tweets220

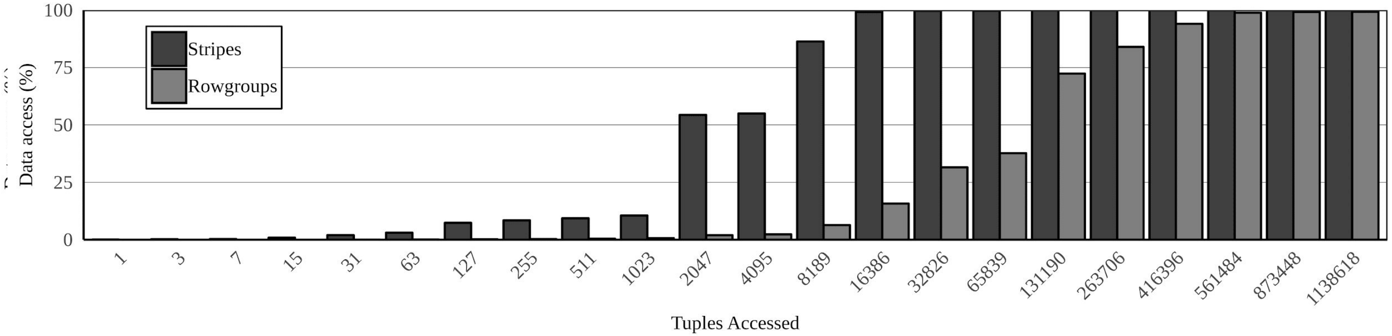


(b) Stripes/Rowgroups accessed by LIKE queries on Tweets220

LIKE and OR-LIKE Queries

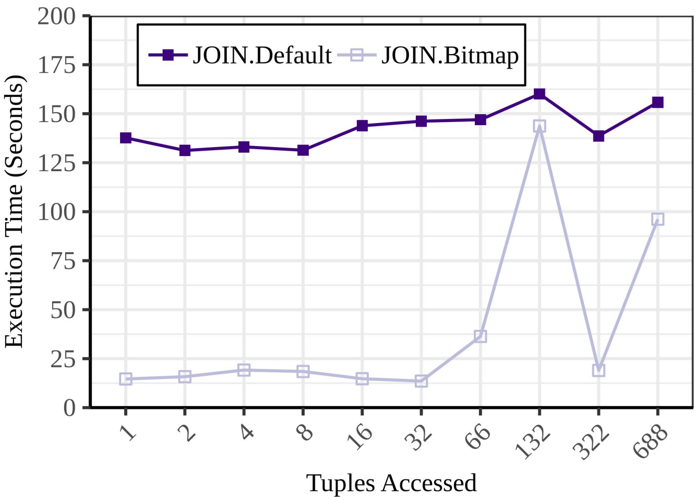


(a) Execution times for LIKE and OR-LIKE queries on Tweets 220

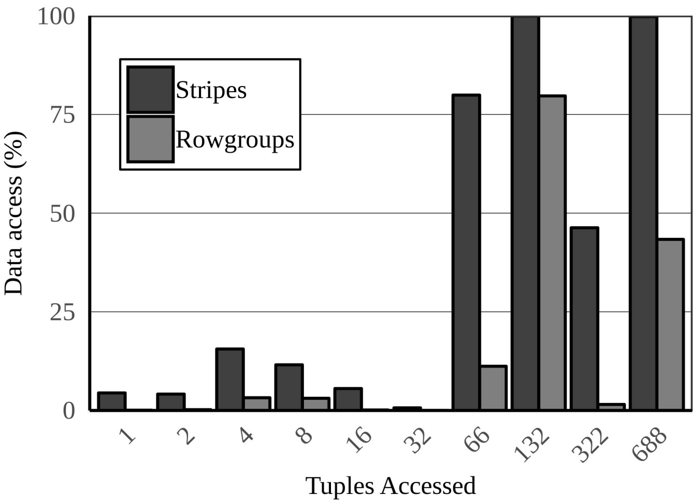


(b) Stripes/Rowgroups accessed by OR-LIKE queries on Tweets220

JOIN Queries

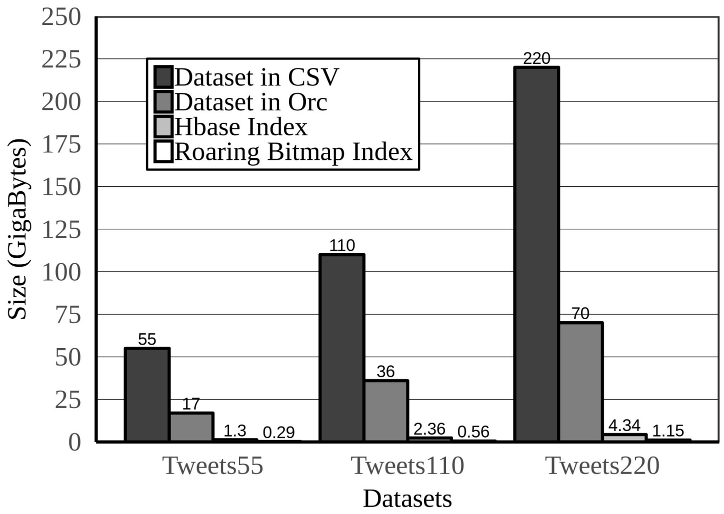


(a) Execution times for JOIN queries on Tweets220

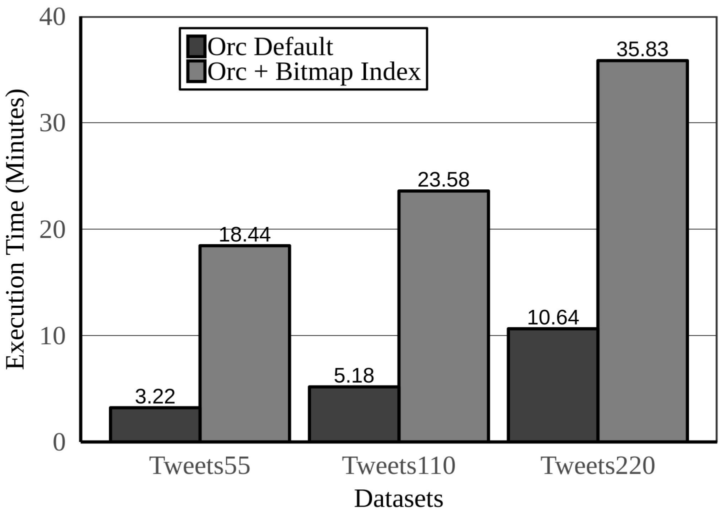


(b) Stripes/Rowgroups accessed by JOIN queries on Tweets220

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.

Related Work

- Bitmap Index for Database Service (BIDS)
  - *Peng Lu, Sai Wu, Lidan Shou, and Kian-Lee Tan.* ***2013.*** *An efficient and compact indexing scheme for large-scale data store. In Data Engineering (**ICDE**), 2013 IEEE 29th International Conference on. IEEE, 326–337.*
  - Uses WAH[3], bit-sliced encoding or partial indexing depending on the data characteristics.
  - The compute nodes are organized according to the Chord protocol, and the indexes are distributed across the nodes.
- Pilosa
  - Open source (https://www.pilosa.com/)
  - Modified version of roaring bitmap for compression.
  - Bitmaps are stored in disk using their own data model.

Related Work

- Bitmap Index for Database Service (BIDS)
  - *Peng Lu, Sai Wu, Lidan Shou, and Kian-Lee Tan.* ***2013.*** *An efficient and compact indexing scheme for large-scale data store. In Data Engineering (**ICDE**), 2013 IEEE 29th International Conference on. IEEE, 326–337.*
  - Uses WAH[3], bit-sliced encoding or partial indexing depending on the data characteristics.
  - The compute nodes are organized according to the Chord protocol, and the indexes are distributed across the nodes.
- Pilosa
  - Open source (https://www.pilosa.com/)
  - Modified version of roaring bitmap for compression.
  - Bitmaps are stored in disk using their own data model.

- Existing Work
  - Use a fixed compression algorithm
  - Lock users to their specific implementation to store, distribute and retrieve bitmap indices.

Conclusion

- A lightweight, flexible and open source bitmap indexing framework is proposed to efficiently index and search for substrings in big data.
- Execution times can be significantly accelerated for queries with high selectivity.
- Storage costs are minimal.
- Initial runtime overhead due to the index creation process.

Thank You - DOLAP 2019

- Workshop Chairs
  - Il-Yeol Song, Drexel University, United States (General Chair)
  - Oscar Romero, Universitat Politecnica de Catalunya, Spain (Program Chair)
  - Robert Wrembel, Poznan University of Technology, Poland (Program Chair)
- Steering Committee
- Program Committee

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.