Pivotal

Petabyte Scale Data Warehousing Greenplum

Postgres Conf 2018

Distribution

Marshall Presser
Craig Sylvester
Andreas Scherbaum
17 April 2018

CREATE TABLE Define Data Distributions

- One of the most important aspects of GP!
- Every table has a distribution method
- DISTRIBUTED BY (column)
  - Uses a hash distribution
- DISTRIBUTED RANDOMLY
  - Uses a random distribution which is not guaranteed to provide a perfectly even distribution
- Explicitly define a column or random distribution for all tables
  - Do not use the default

DISTRIBUTED BY (*column_name*)

- Use a single column that will distribute data across all segments evenly
- Example

```
CREATE TABLE foo (
    id integer,
    size float8)
distributed by (id);
```

- For large tables significant performance gains can be obtained with local joins (co-located joins)
  - Distribute on the same column for tables commonly joined together
- Co-located join is performed within the segment
  - Segment operates independently of other segments
- Co-located join eliminates or minimizes motion operations
  - Broadcast motion or Redistribute motion

DISTRIBUTED RANDOMLY

- Uses a random algorithm
  - Distributes data across all segments
  - Minimal data skew but not guaranteed to have a perfectly even distribution
- Example

```
CREATE TABLE foo (
    id integer,
    size float8)
distributed randomly;
```

- Any query that joins to a table that is distributed randomly will require a motion operation
  - Redistribute motion
  - Broadcast motion

DISTRIBUTED RANDOMLY

- Uses a random algorithm
  - Distributes data across all segments
  - Minimal data skew but not guaranteed to have a perfectly even distribution
- Example

```
CREATE TABLE foo (
    id integer,
    size float8)
distributed randomly;
```

- Any query that joins to a table that is distributed randomly will require a motion operation
  - Redistribute motion
  - Broadcast motion

ALTERING DISTRIBUTION

- Distribution is done at table creation time, but can be altered.
  - ALTER TABLE sales SET DISTRIBUTED BY (customer_id);
- When you change the hash distribution of a table, table data is automatically redistributed.

- Changing the distribution policy to a random distribution does not cause the data to be redistributed.:
  - ALTER TABLE sales SET DISTRIBUTED RANDOMLY;

- Old school: create a new table with the new distribution and then swap.
  - CREATE TABLE new_foo as select * from foo distributed randomly;
  - DROP TABLE foo;
  - ALTER TABLE new_foo RENAME to foo;

DISTRIBUTED RANDOMLY

- Uses a random algorithm
  - Distributes data across all segments
  - Minimal data skew but not guaranteed to have a perfectly even distribution
- Example

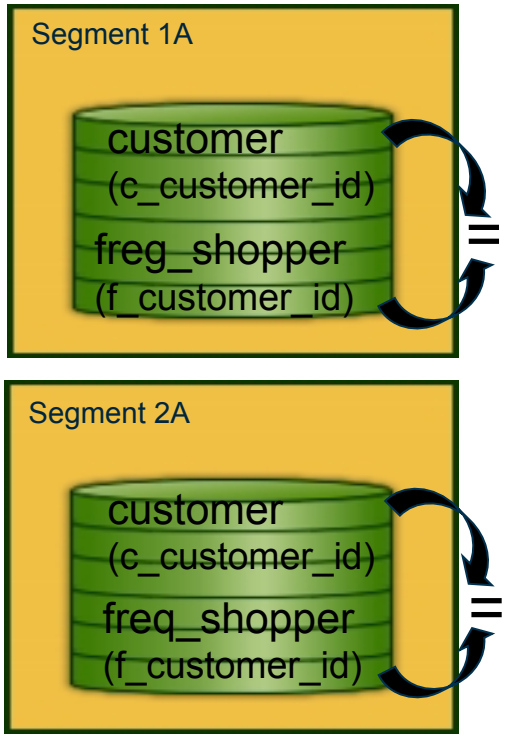
```
CREATE TABLE foo (
    id integer,
    size float8)
distributed randomly;
```

- Any query that joins to a table that is distributed randomly will require a motion operation
  - Redistribute motion
  - Broadcast motion

Hash Distributions: Data Skew and Computational Skew

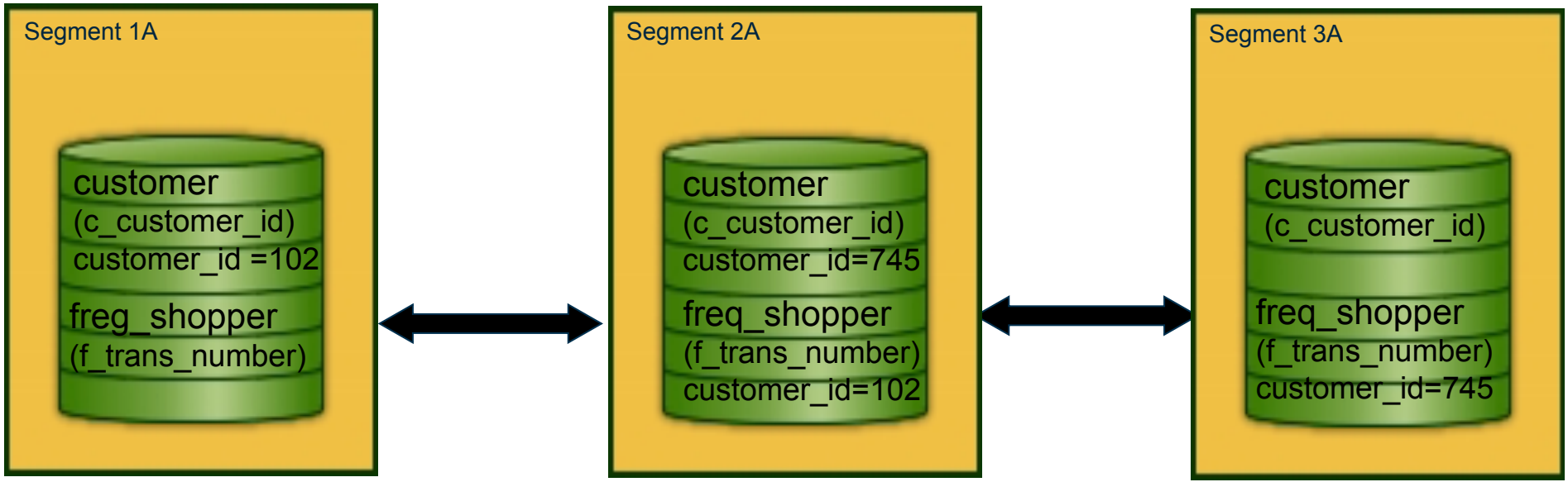
- Select a distribution key with unique values and high cardinality that will not result in data skew
  - Do not distribute on boolean keys and keys with low cardinality
    - The system distributes rows with the same hash value to the same segment instance therefore resulting in the data being located on only a few segments
- Select a distribution key that will not result in computational skew (in flight when a query is executing)
  - Operations on columns that have low cardinality or non-uniform distribution

Use the Same Distribution Key for Commonly Joined Tables



Distribute on the same key
used in the join
to obtain local joins

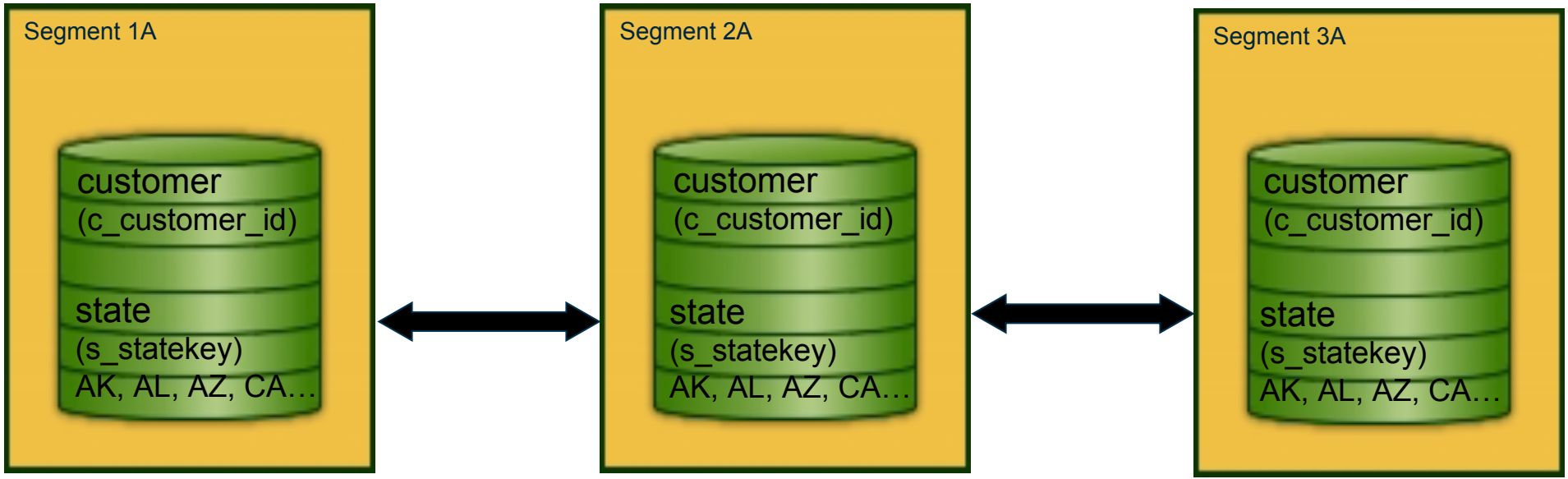
Redistribution Motion



WHERE customer.c_customer_id = freg_shopper.f_customer_id

freq_shopper table is dynamically redistributed on f_customer_id

Broadcast Motion



WHERE customer.c_statekey = state.s_statekey

The state table is dynamically broadcasted to all segments

Commonly Joined Large Tables Use the Same Data Type for Distribution Keys

customer (c_customer_id)	745::int
freq_shopper (f_customer_id)	745::varchar(10)

- Values might appear the same but they are stored differently at the disk level
- Values might appear the same but they HASH to different values
  - Resulting in *like* rows being stored on different segments
  - Requiring a redistribution before the tables can be joined

DISTRIBUTED BY (*column_name*)

- Do not distribute on columns that will be used in the WHERE clause of a query
- Do not distribute on dates or timestamps
- Never distribute and partition a table on the same column

Always Check for Data Skew on Initial Load and Subsequent Loads

- SELECT COUNT(*), gp_segment_id FROM <table-name> GROUP BY gp_segment_id;
- SELECT 'facts' as "Table Name",max(c) as "Max Seg Rows", min(c) as "Min Seg Rows", (max(c)-min(c))*100.0/ max(c) as "Percentage Difference Between Max & Min" from (SELECT count(*) c, gp_segment_id from facts group by 2) as a;

Distribution skew examples

```
gpuser=> SELECT count(*), gp_segment_id
from otp_x group by gp_segment_id order by 1;
  count      | gp_segment_id
-------------+---------------
     7800 |             7
   699342 |             0
  1121978 |             5
  2002998 |             6
  2256304 |             2
  4125946 |             3
  5270874 |             1
  5374803 |             4
(8 rows)
```

```
gpuser=> SELECT count(*), gp_segment_id from otp_r
group by gp_segment_id order by 1;
  count     | gp_segment_id
------------+---------------
 2607504 |             7
 2607505 |             5
 2607505 |             4
 2607505 |             0
 2607505 |             2
 2607506 |             3
 2607507 |             1
 2607508 |             6
(8 rows)
```