GREENPLUM DATABASE

墨天轮

当谈起Greenplum 7时，我们在谈什么？

—《Greenplum 7 新版本大剧透》系列直播

讲师：杨瑜，Greenplum研发总监

活动时间：2021年3月31日 20：00 - 21：00

GREENPLUM DATABASE®

Greenplum中文社区

https://cn.greenplum.org

博文 · 资料 · 文档 · 项目

GREENPLUM DATABASE®

微信技术讨论群
添加入群小助手：gp_assistant

微信公众号
技术干货、行业热点、活动预告

欢迎访问Greenplum中文社区：cn.greenplum.org

GREENPLUM DATABASE®

Greenplum7 New Features

March 2021

vmware®

Confidential | ©2020 VMware, Inc.

GREENPLUM DATABASE®

Greenplum v6

September 2019, GA Release

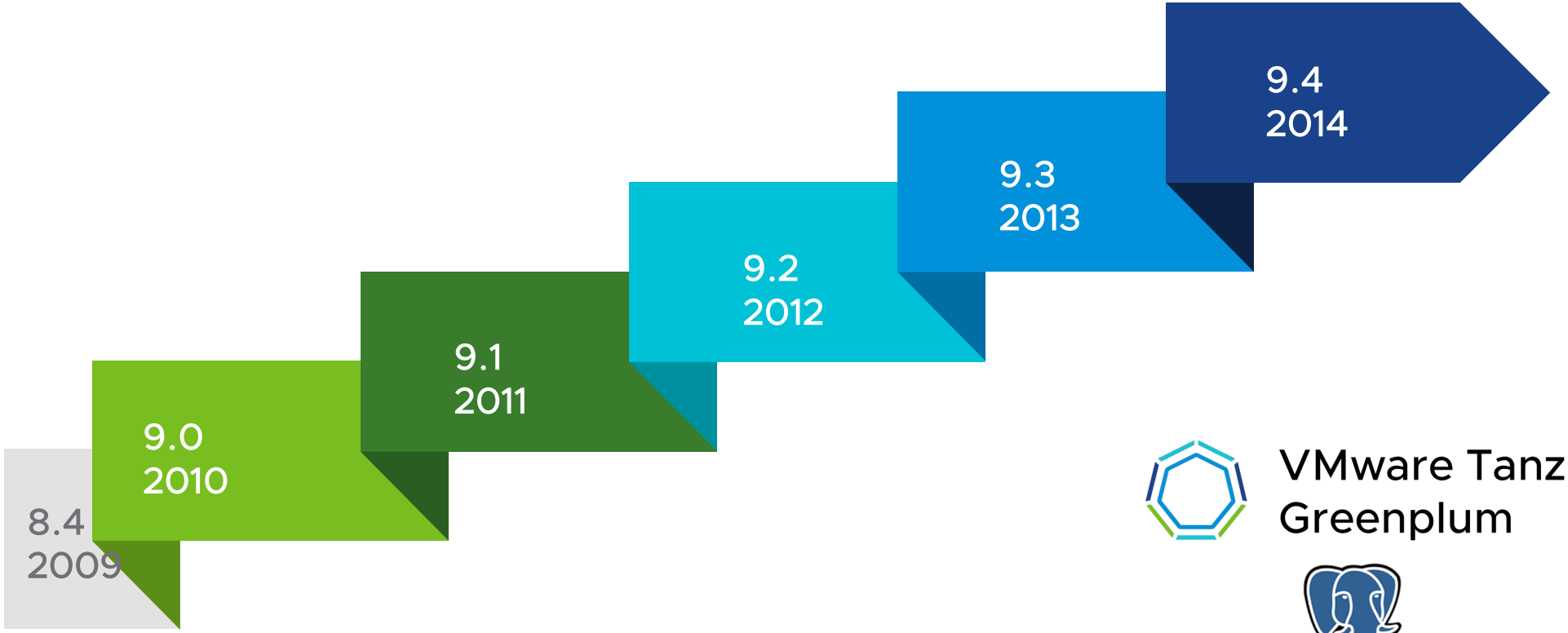
Hundreds of Thousands of Production Hours

- Speed
- Stability
- Features

Confidential | ©2020 VMware, Inc.

Greenplum 6 is based on PostgreSQL 9.4

Content from 6 Major PostgreSQL Versions



TOTAL: Over 1 million lines of code updated

VMware Tanzu Greenplum

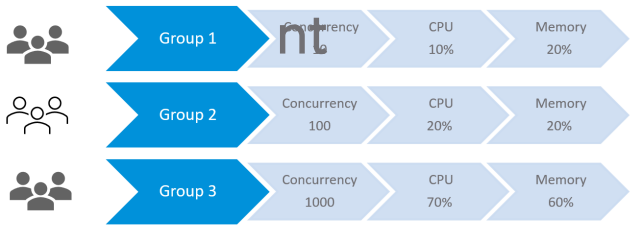
PostgreSQL

Greenplum 6: Diverse Workloads Support

GREENPLUM DATABASE®

Transactions	Index Queries	Traditional DW Queries
Replicated Tables	Complex Many Table Joins	Subqueries, CTEs, Recursive, Views
Materialized Views	Machine Learning	Multi-Modal Analytics

Resource Management

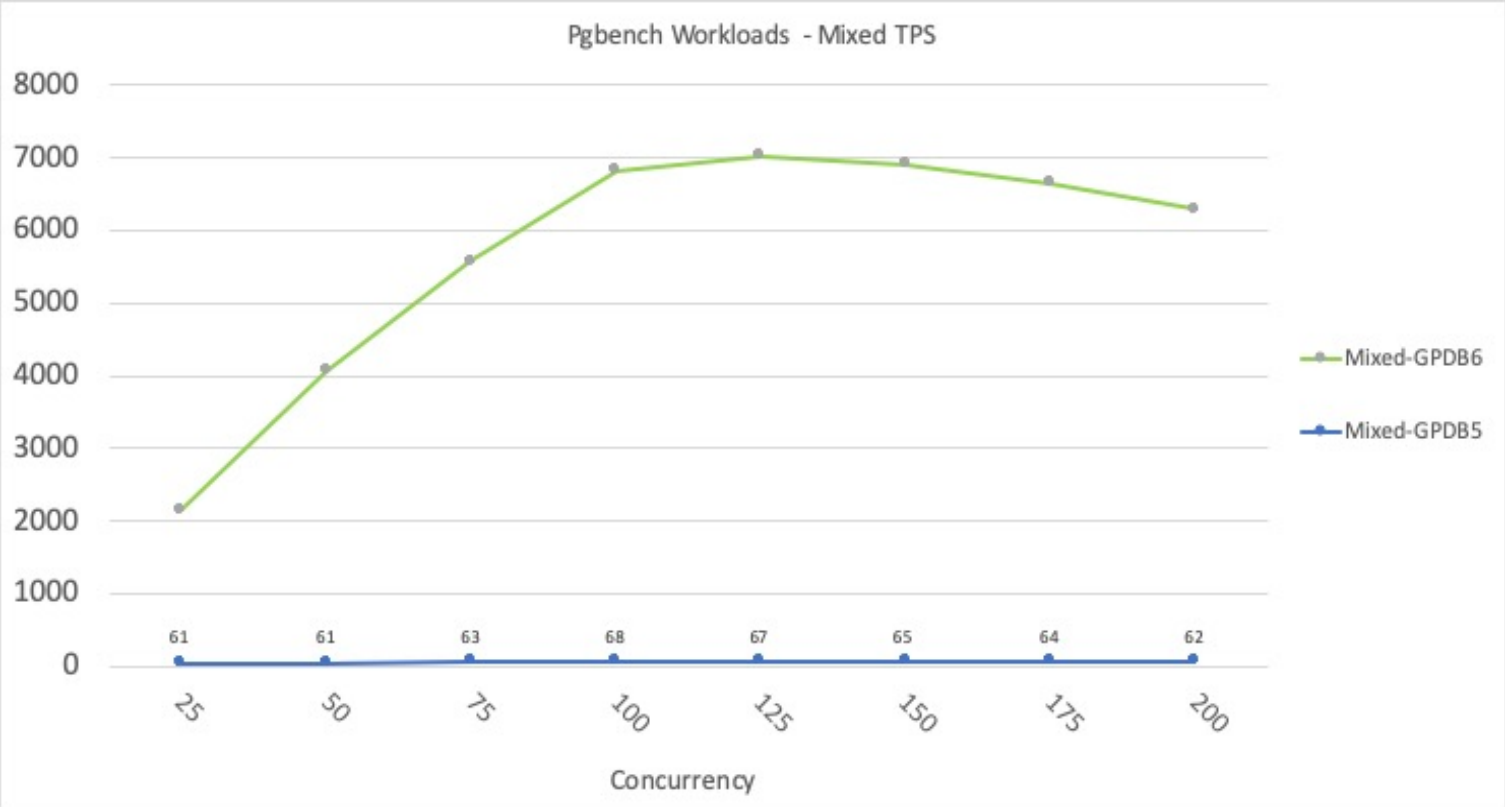


Confidential | ©2020 VMware, Inc.

pgbench Results – 'Mixed'

GREENPLUM DATABASE®

At peak – Greenplum 6 Mixed is faster by a factor of 104X.

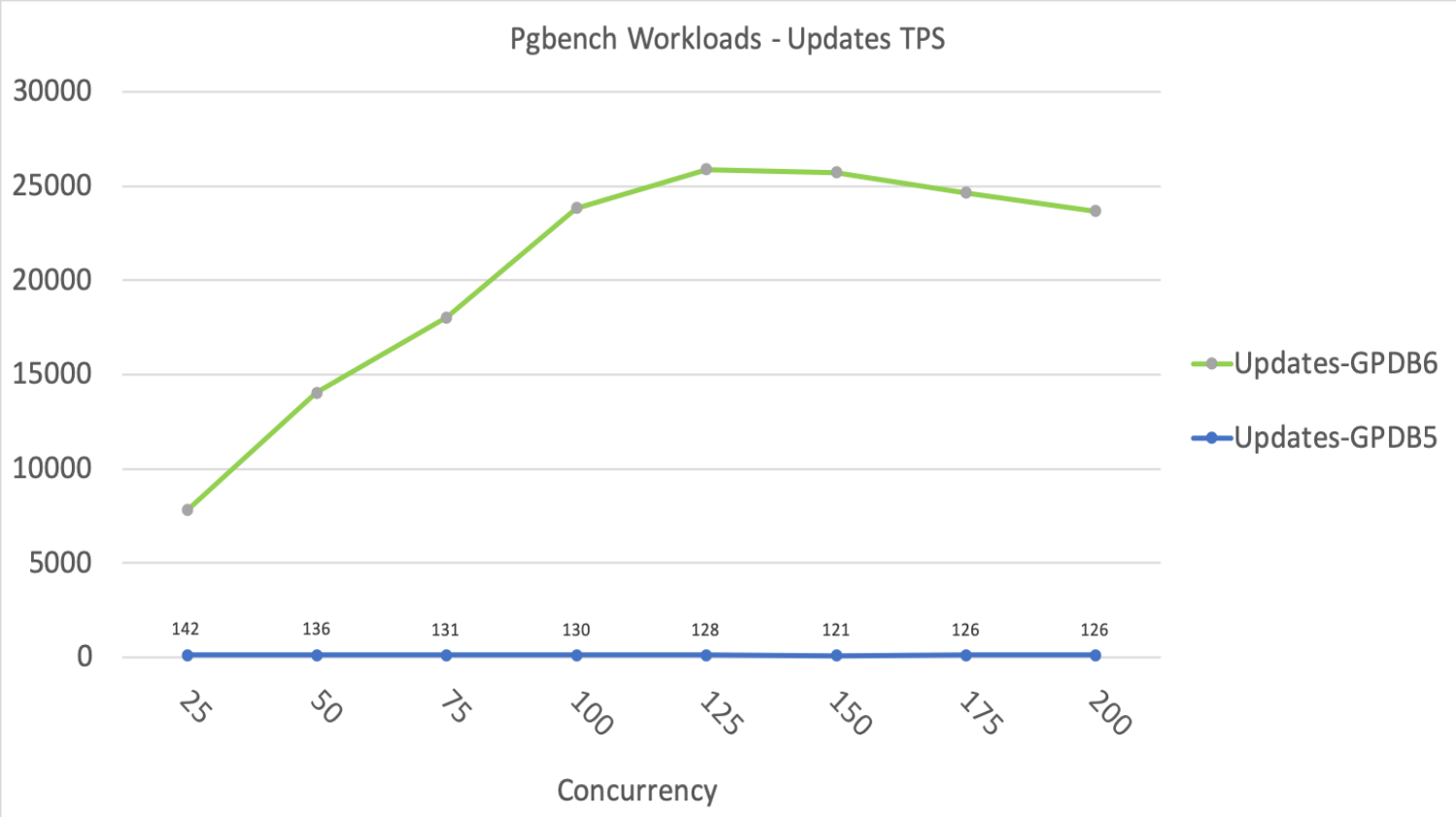


pgbench -c $N -j $N -r -T 60 -s 1000

```
\set scale 1000
\set nbranches 1 * :scale
\set ntellers 10 * :scale
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom bid 1 :nbranches
\setrandom tid 1 :ntellers
\setrandom delta -5000 5000

BEGIN;
UPDATE pgbench_accounts SET abalance =
abalance + :delta WHERE aid = :aid;
SELECT abalance FROM pgbench_accounts
WHERE aid = :aid;
UPDATE pgbench_tellers SET tbalance =
tbalance + :delta WHERE tid = :tid;
UPDATE pgbench_branches SET bbalance =
bbalance + :delta WHERE bid = :bid;
INSERT INTO pgbench_history (tid, bid, aid,
delta, mtime) VALUES (:tid, :bid, :aid, :delta,
CURRENT_TIMESTAMP);
END;
```

pgbench Results – Updates

At peak – Greenplum 6 Updates is faster by a factor of ~200X.

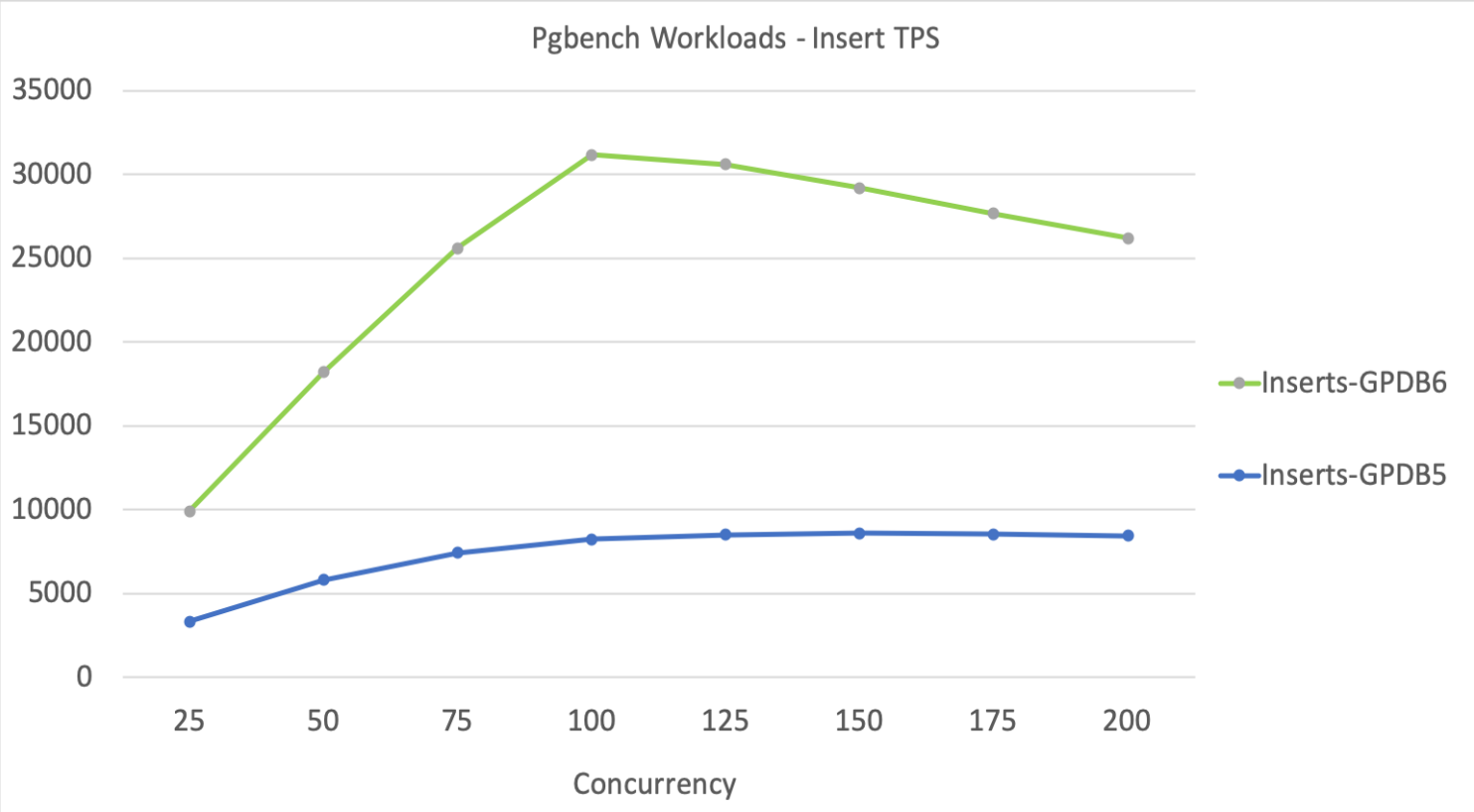


pgbench -c $N -j $N -r -T 60 -s 1000 -f update-only.sql

```
\set scale 1000
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom delta -5000 5000

UPDATE pgbench_accounts SET abalance =
abalance + :delta WHERE aid = :aid;
```

pgbench Results – Inserts

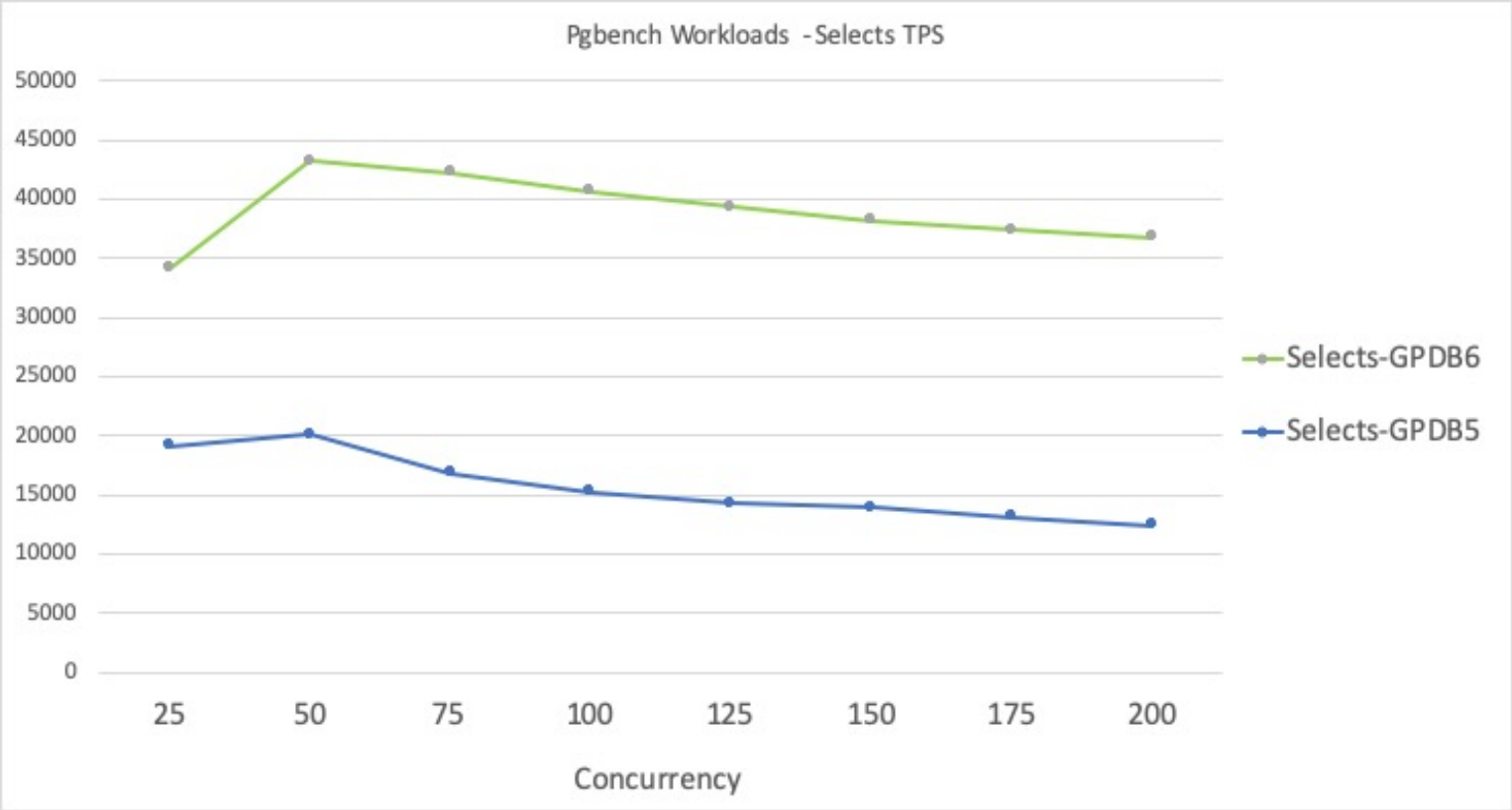
At peak – Greenplum 6 Updates is faster by a factor of 3.8X.




```
pgbench -c $N -j $N -r -T 60 -s 1000  -f insert-only.sql

\set scale 1000
\set nbranches 1 * :scale
\set ntellers 10 * :scale
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom bid 1 :nbranches
\setrandom tid 1 :ntellers
\setrandom delta -5000 5000

INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
```

pgbench Results – Selects

At peak – Greenplum 6 Updates is faster by a factor of 2.5X.



pgbench -c $N -j $N -r -T 60 -s 1000 -S

```
\set scale 1000
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts

SELECT abalance FROM pgbench_accounts
WHERE aid = :aid;
```

GREENPLUM DATABASE®

Greenplum Command Center

vmware®

Confidential | ©2020 VMware, Inc.

Roadmap

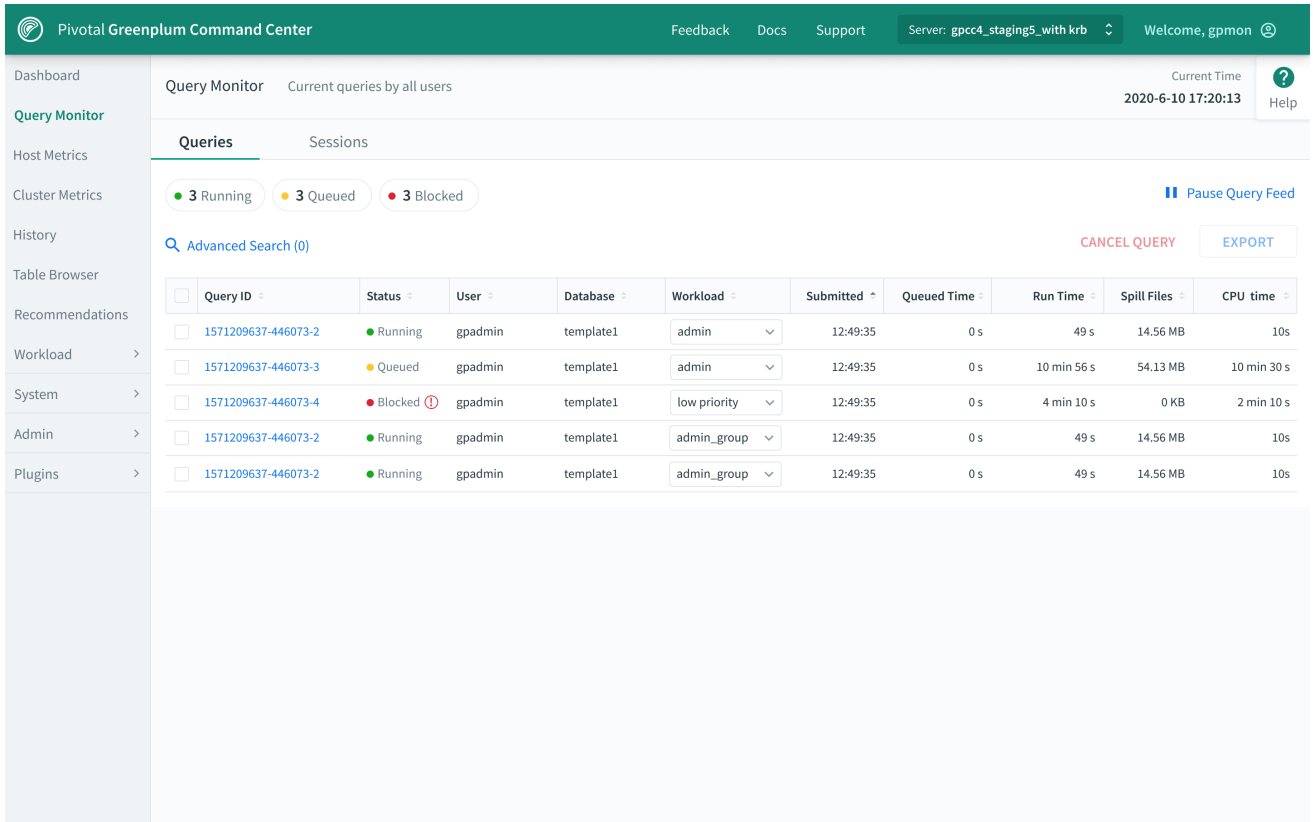
Short-Mid Term

GREENPLUM DATABASE®

Q4 FY'20	Q1 FY'21	Q2 FY' 21	Q3 FY' 21	Q4 FY'21
VMW Feedback Less Alert Notifications History/Table Browser Search Filters In-App Notifications	Advanced Query Monitor (Pause/Filter/Session Monitor) New WLM Rule for Idle Session Killing Query Tag History Real Time Metrics Table WLM Rule by Spill File Size	Resource Group Insights WLM Rule Enhancements(Time Window/Spill File Size) GPCC Plugin Phase 2 Visual Plan Checker	Advanced Recommendations (partitions/distribution keys/cache number…) Customized Alerts Show tables accessed by a query	

Advanced Query Monitor

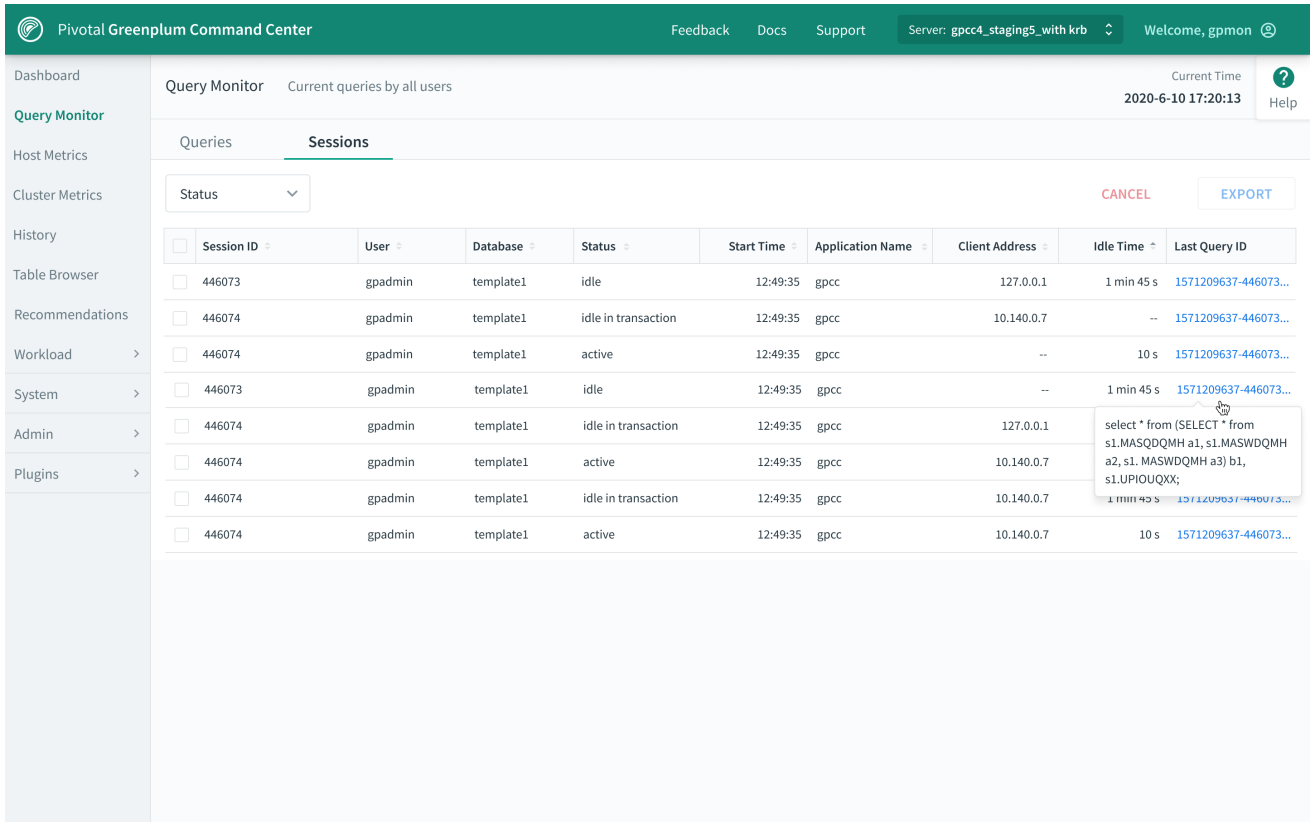
Pause+Filter+Session View



Confidential | ©2020 VMware, Inc.

Advanced Query Monitor

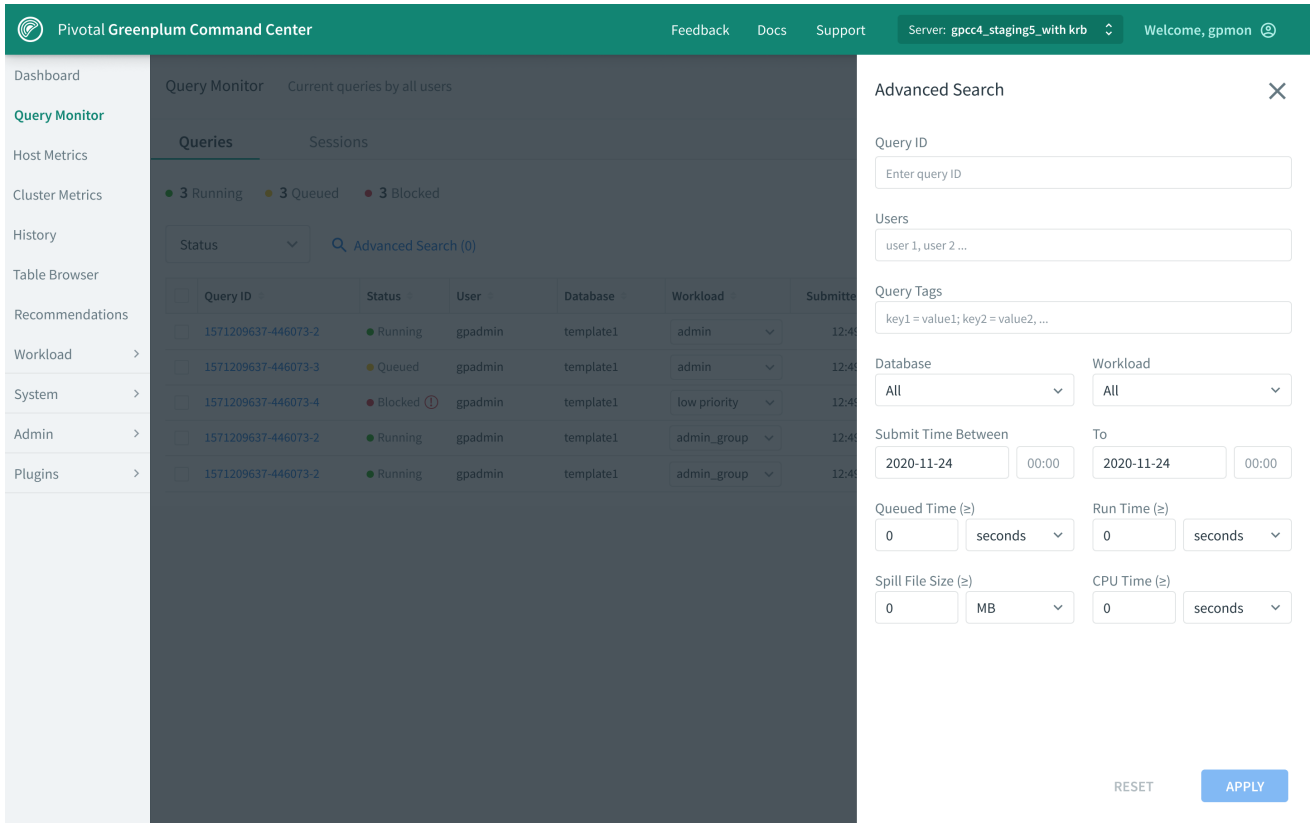
Pause+Filter+Session View



Confidential | ©2020 VMware, Inc.

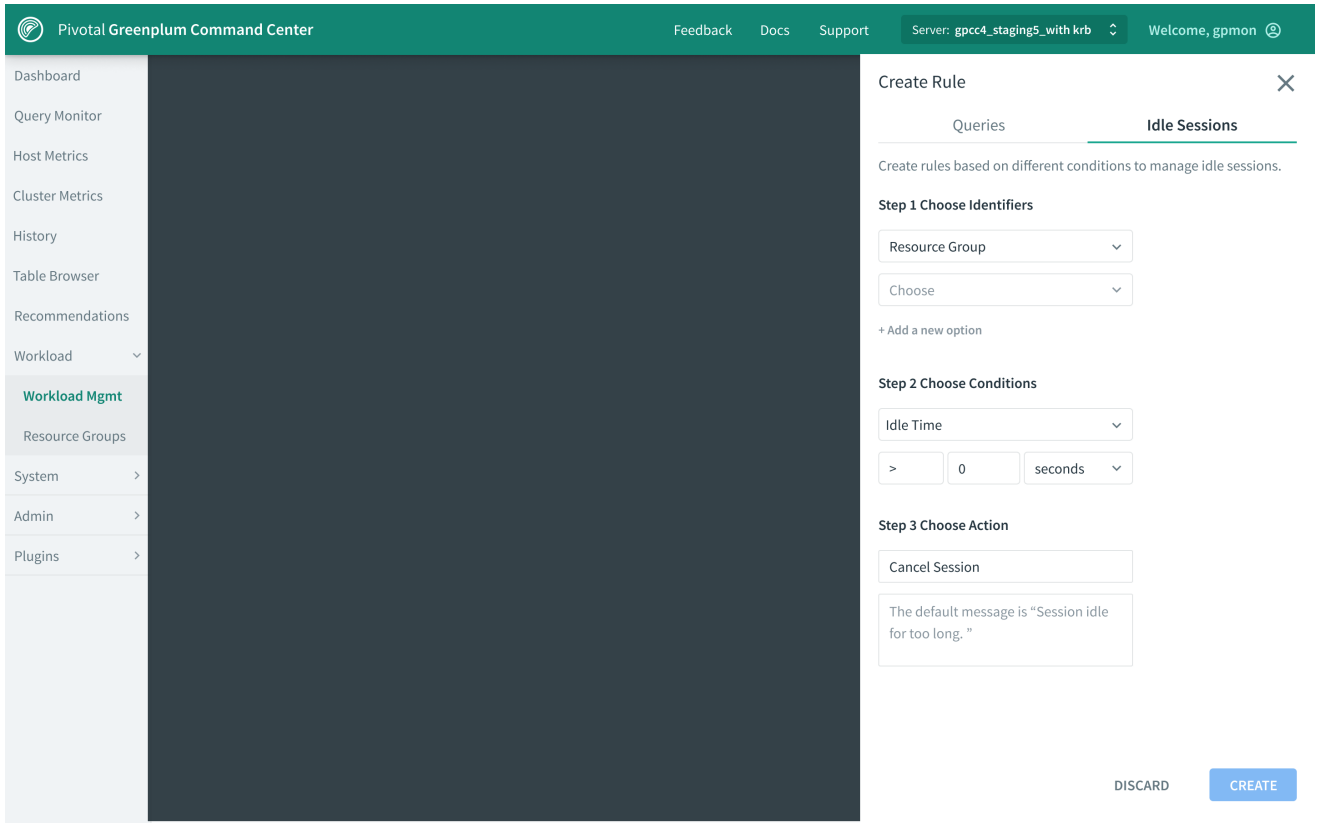
Advanced Query Monitor

Pause+Filter+Session View



New WLM Rule for Idle Session Killing

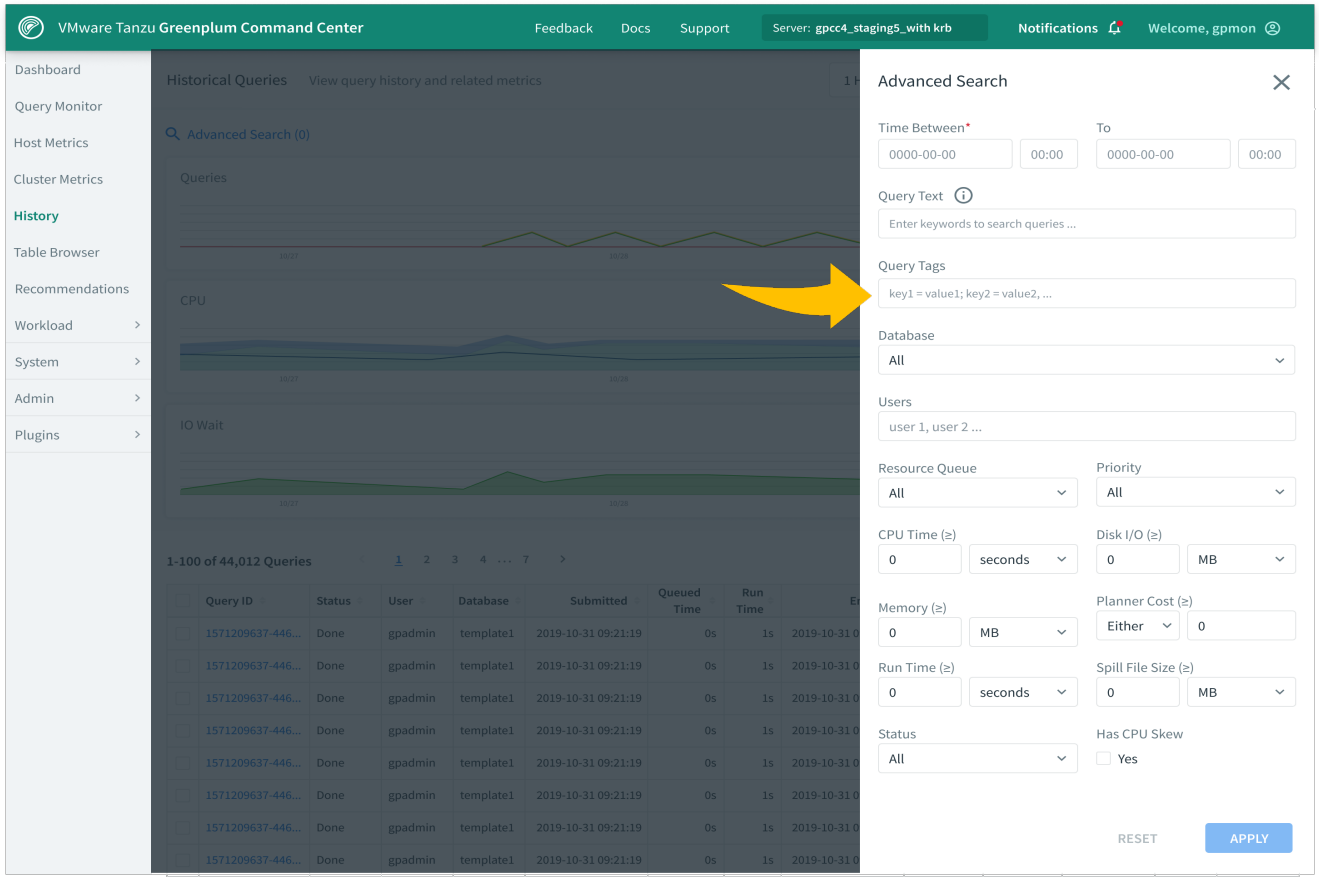
Create From A New Idle Sessions Tab



Confidential | ©2020 VMware, Inc.

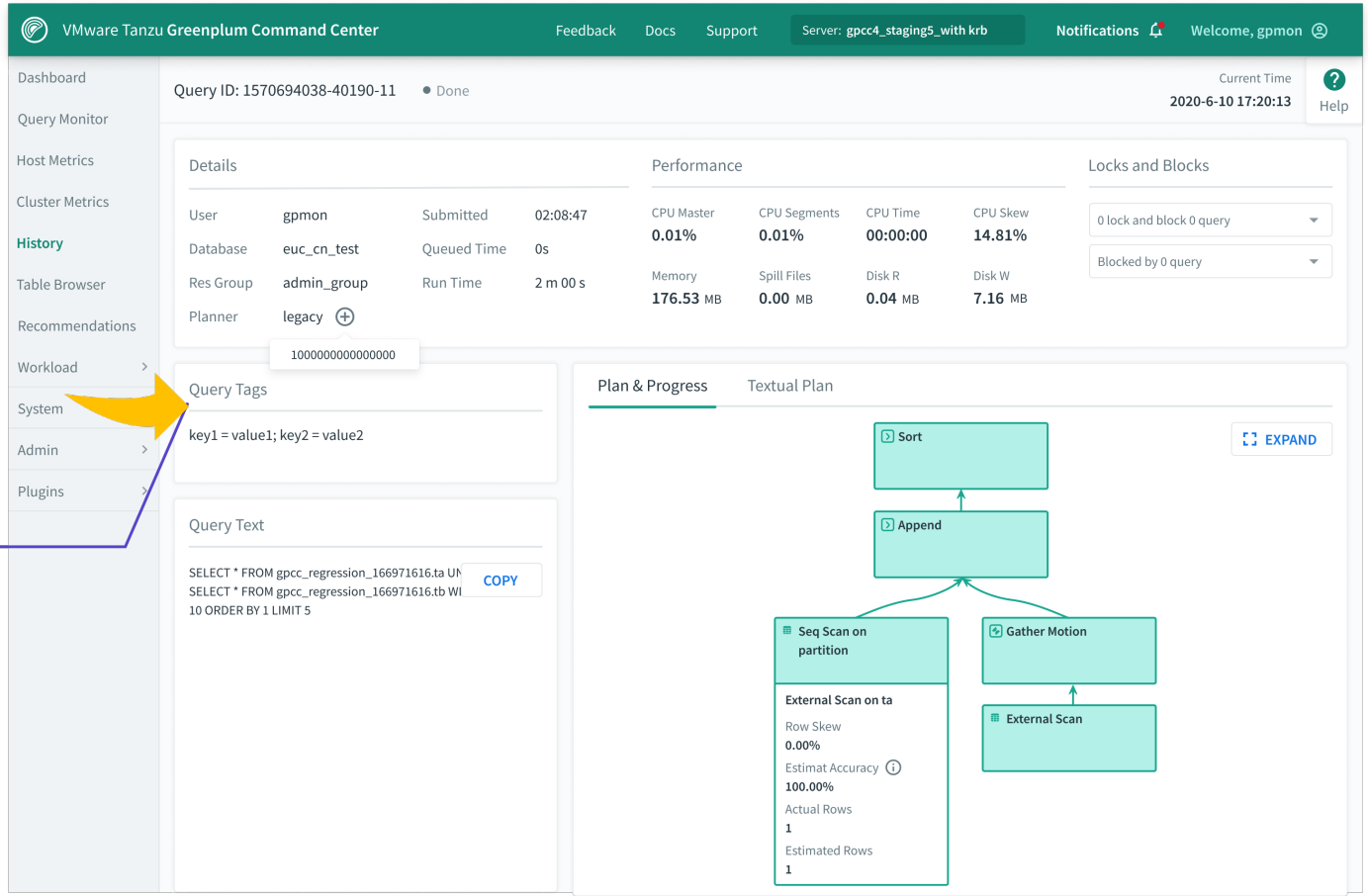
Query Tag History

Filter by Query Tags



Query Tag History

View Query Tags



GREENPLUM DATABASE®

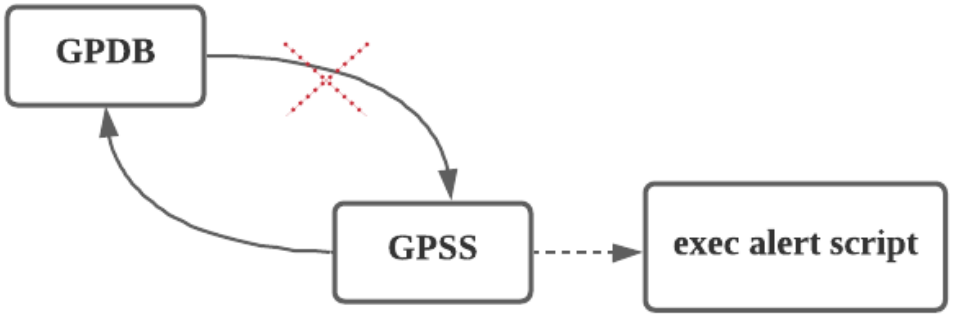
Greenplum GPSS

vmware®

Confidential | ©2020 VMware, Inc.

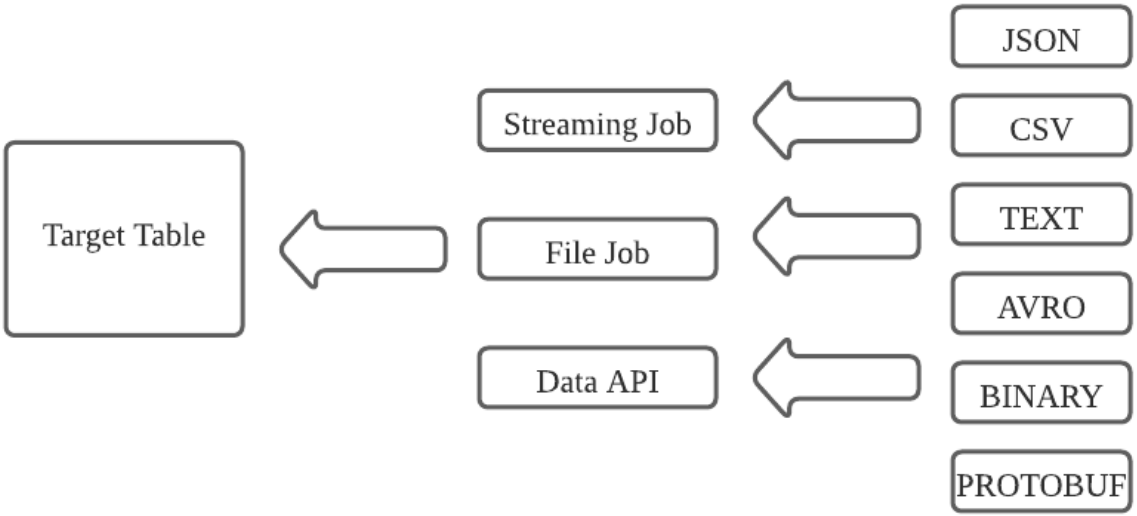
Monitor Enhancement

- More metrics in Prometheus
- Job progress in Prometheus
- Alert in case of job fails



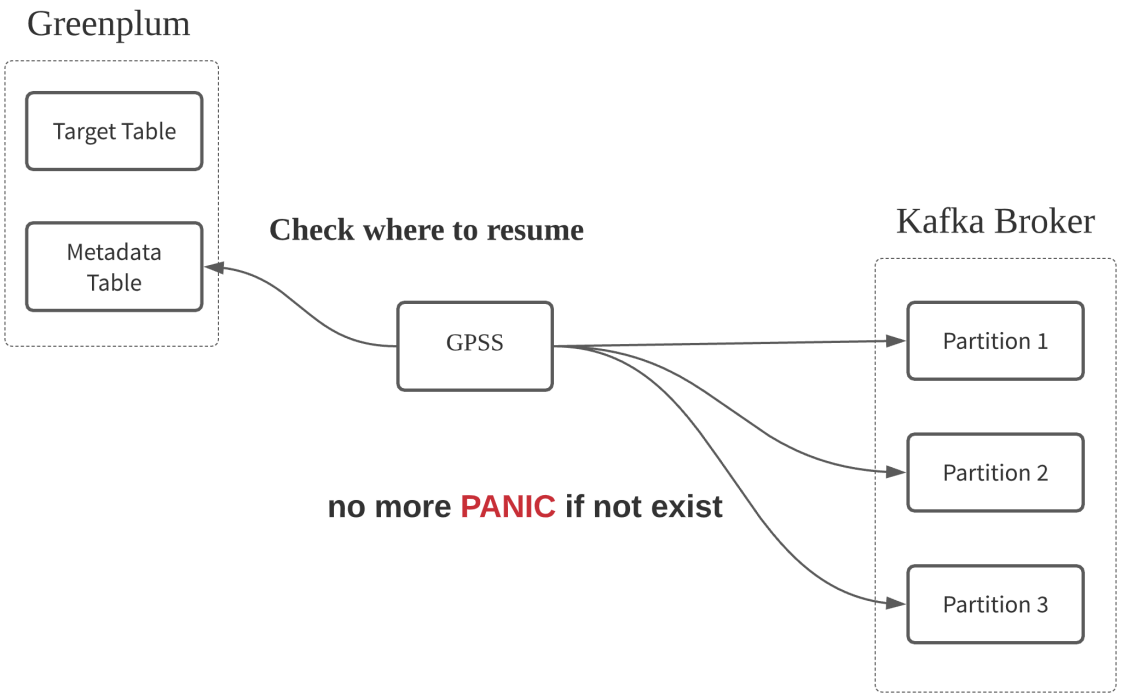
All interfaces support the same data format

- Interfaces
  - Streaming job
  - File job
  - Data API
- Formats
  - TEXT/CSV/JSON
  - AVRO
  - Binary
  - Protobuf
  - …



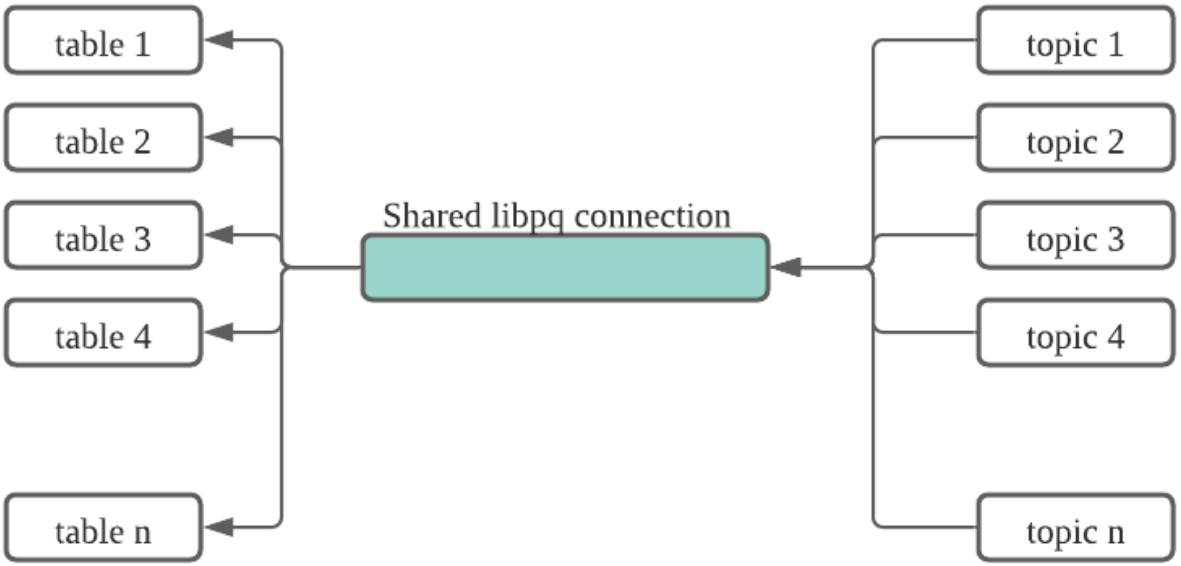
More options for offset GAP

- --force-reset in job.yaml
  - earliest
  - latest
  - timestamp
- Tolerate control message of Kafka



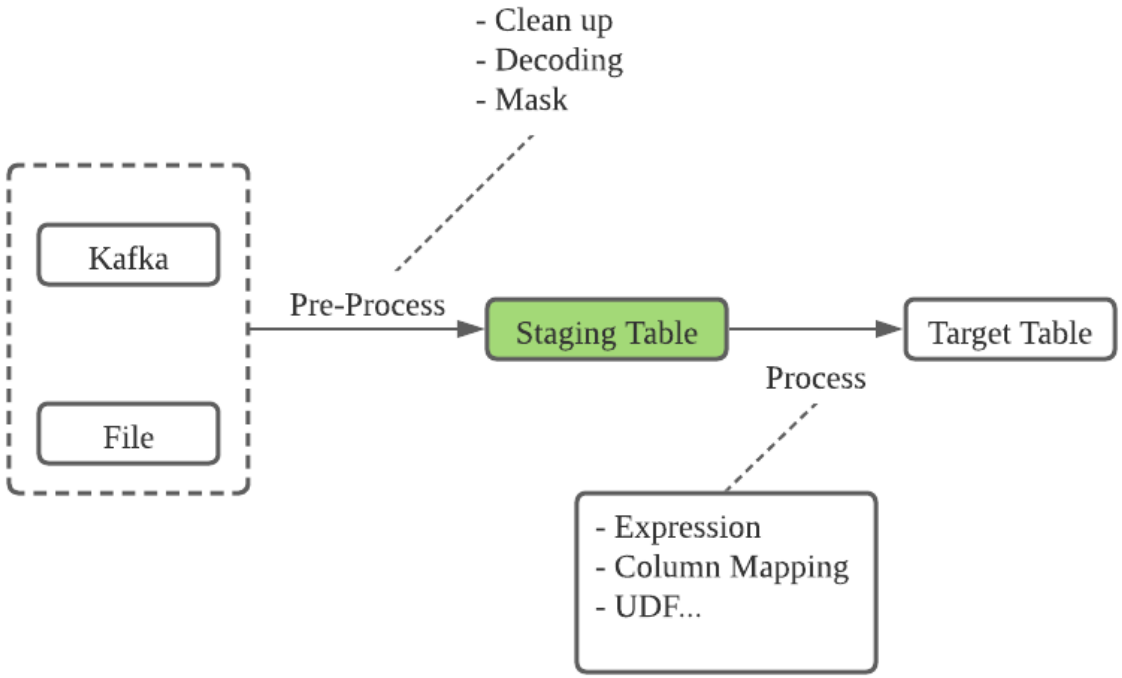
Multiple jobs share the same session

- GPSS can run multiple streaming jobs in single GPDB session
- Help to decrease connection number when the total streaming data throughput is small
- Optimize for CDC use case



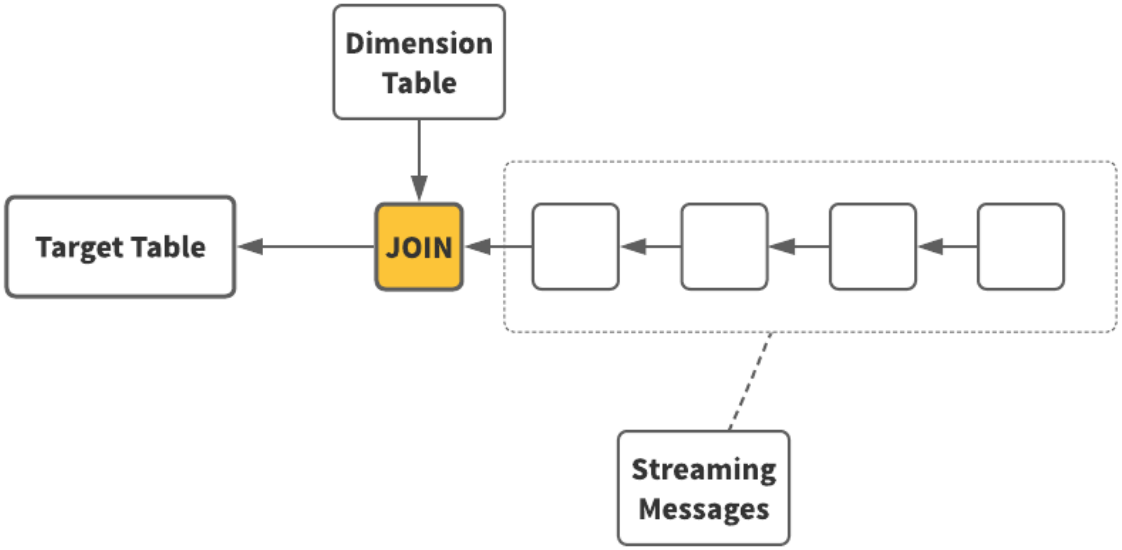
Pre-Process

- GPSS load message/file to a staging table for pre-processing
- For purpose of data cleanup, decoding, (de)mask, etc



JOIN with Dimension table

- Ingest the join result of streaming message and the dimensional table



Confidential | ©2020 VMware, Inc.

GREENPLUM DATABASE®

Greenplum 7

Confidential | ©2020 VMware, Inc.

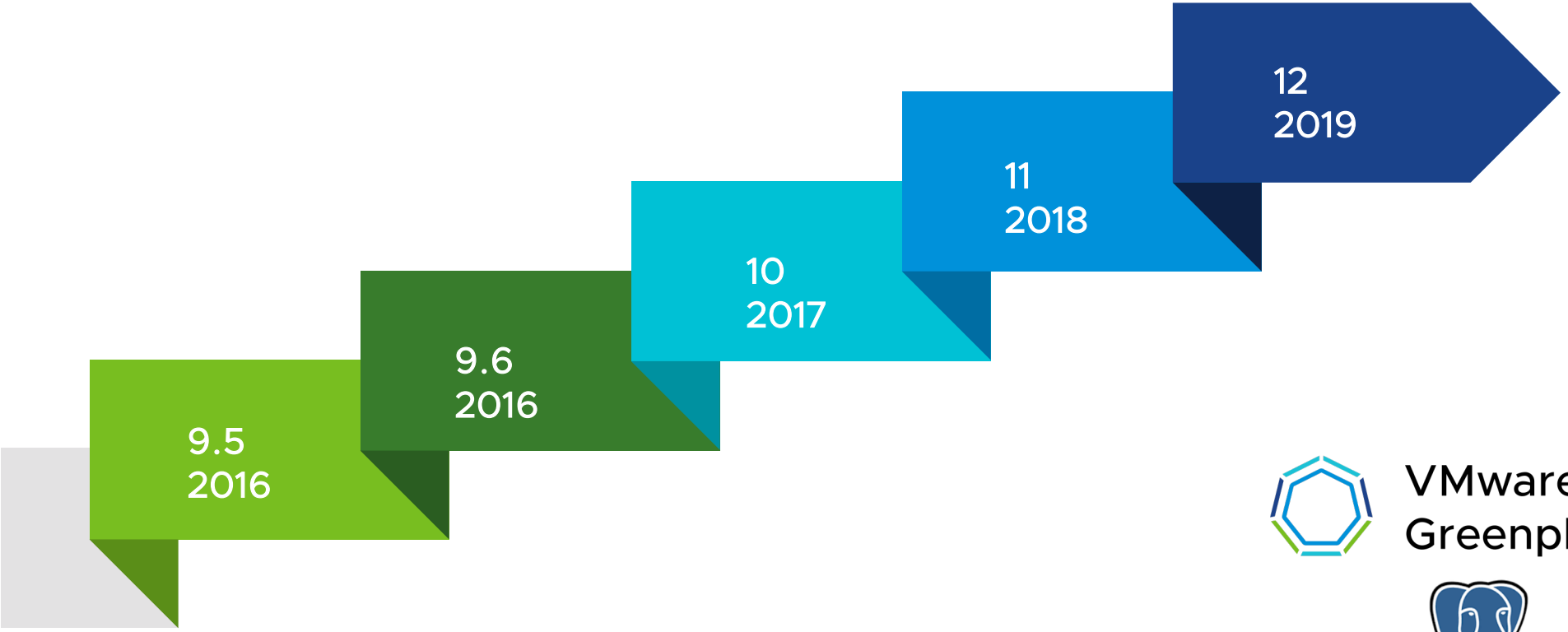
GREENPLUM DATABASE®

Greenplum 7 will be based on PostgreSQL 12

Confidential | ©2020 VMware, Inc.

Greenplum 7 is based on PostgreSQL 12

Content from 5 Major PostgreSQL Versions



VMware Tanzu Greenplum

PostgreSQL

GREENPLUM DATABASE

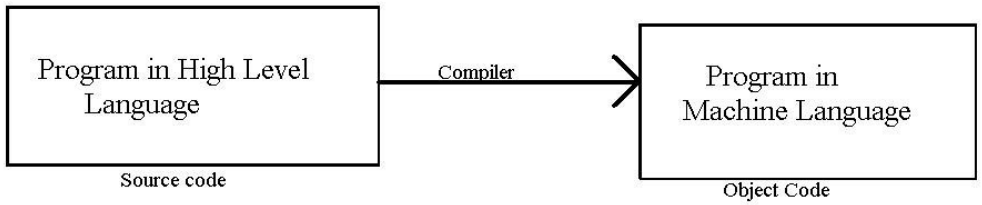
Just-In-Time (JIT) Compilation

Just-In-Time (JIT) Compilation

what is JIT

Just-in-Time (JIT) compilation is the process of turning some form of interpreted program evaluation into a native program, and doing so at run time.

- ➔ SQL is an interpreted language
- ➔ Can we do real time SQL Query compilation to make execution faster?
- ➔ Yes we can, and it produces tangible and sometimes massive gains!



Just-In-Time (JIT) Compilation

PostgreSQL's JIT Implementation

By default uses LLVM to perform JIT

Accelerate expression evaluation

- WHERE clauses
- target lists
- aggregates
- projections

Accelerate tuple deforming: transforming an on-disk tuple into its in-memory representation

Inlining: inline the bodies of small functions into the expressions using them

Optimization: optimize generated code

Just-In-Time (JIT) Compilation

when to JIT

Long-running CPU-bound queries

Cost-based, controlled by GUCs:

- jit (on by default)
- jit_above_cost
- jit_inline_above_cost
- jit_optimize_above_cost
- jit_provider (llvmjit by default, rarely required)

Confidential | ©2020 VMware, Inc.

Just-In-Time (JIT) Compilation

setup

```
-- create TPC-H lineitem table
CREATE TABLE lineitem
(
   l_orderkey integer,
   l_partkey integer,
   l_suppkey integer,
   l_linenumber integer,
   l_quantity real,
   l_extendedprice real,
   l_discount real,
   l_tax real,
   l_returnflag "char",
   l_linestatus "char",
   l_shipdate date,
   l_commitdate date,
   l_receiptdate date,
   l_shipinstruct char(25),
   l_shipmode char(10),
   l_comment char(44),
   l_dummy char(1)
)
DISTRIBUTED BY (l_orderkey);

-- file generated using TPC-H ./dbgen with scale 10
COPY lineitem FROM '/tmp/lineitem.tbl' DELIMITER '|' csv;
COPY 59986052


ANALYZE lineitem;
```

```
-- TPC-H query 1
EXPLAIN ANALYZE
SELECT
    l_returnflag,
    l_linestatus,
    SUM(l_quantity) AS sum_qty,
    SUM(l_extendedprice) AS sum_base_price,
    SUM(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
    SUM(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS
sum_charge,
    AVG(l_quantity) AS avg_qty,
    AVG(l_extendedprice) AS avg_price,
    AVG(l_discount) AS avg_disc,
    COUNT(*) AS count_order
FROM
    lineitem
WHERE
    l_shipdate <= date '1998-12-01' - interval ':1' day
GROUP BY
    l_returnflag,
    l_linestatus
ORDER BY
    l_returnflag,
    l_linestatus;
```

Just-In-Time (JIT) Compilation

run

EXPLAIN ANALYZE TPC-H query 1 with the following JIT options:

	jit_above_cost	jit_optimize_above_cost	jit_inline_above_cost
JIT	0	-1	-1
JIT_OPT	0	0	-1
JIT_INLINE	0	-1	0
JIT_ALL	0	0	0
NO JIT	-1	-1	-1

* 0: always enabled
* -1: always disabled

Confidential | ©2020 VMware, Inc.

Just-In-Time (JIT) Compilation

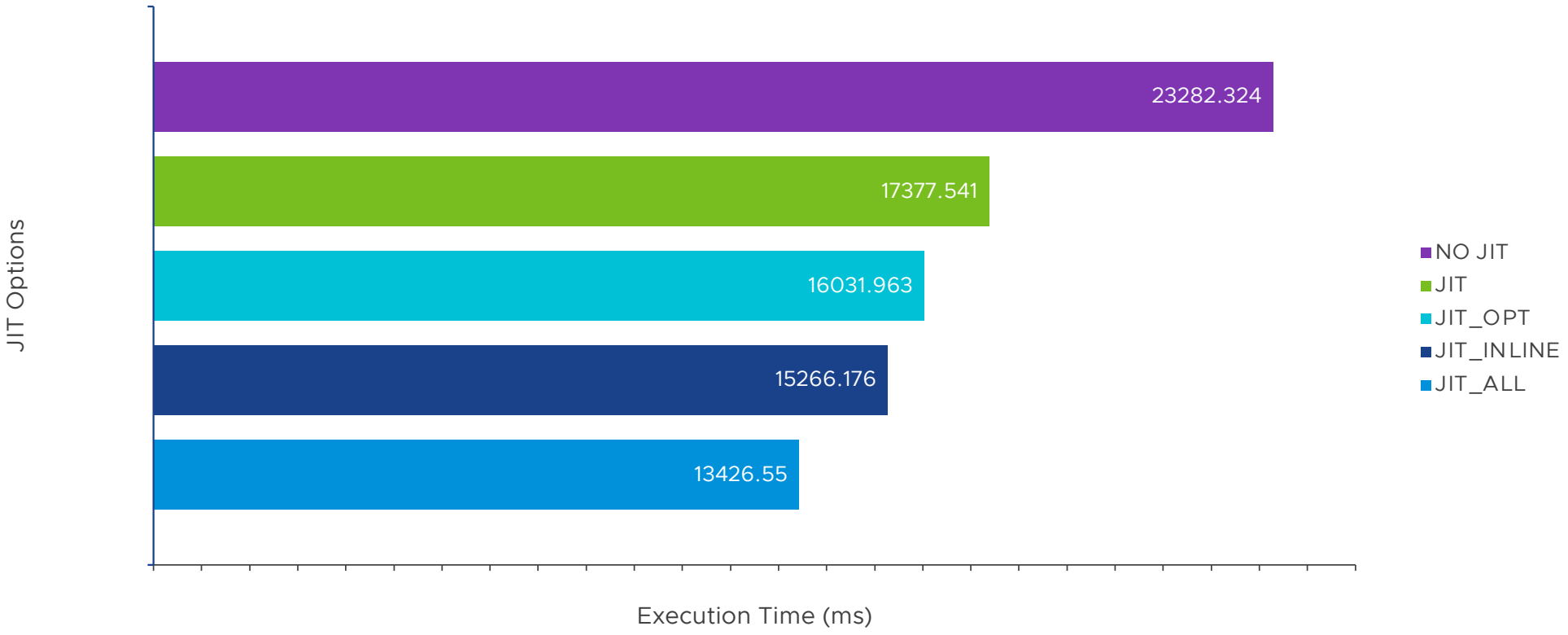
EXPLAIN ANALYZE

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
------
Gather Motion 3:1  (slice1; segments: 3)  (cost=3464757.82..3464757.95 rows=6 width=58)
   Merge Key: l_returnflag, l_linestatus
   ->  Sort  (cost=3464757.82..3464757.83 rows=2 width=58) ...
         Sort Key: l_returnflag, l_linestatus
         Sort Method:  quicksort
         ->  Finalize HashAggregate  (cost=3464757.63..3464757.74 rows=2 width=58) ...
               Group Key: l_returnflag, l_linestatus
               ->  Redistribute Motion 3:3  (slice2; segments: 3)  (cost=3464756.64..3464757.18 rows=6
width=130) ...
                     Hash Key: l_returnflag, l_linestatus
                     ->  Partial HashAggregate  (cost=3464756.64..3464756.82 rows=6 width=130) ...
                           Group Key: l_returnflag, l_linestatus
                           ->  Seq Scan on lineitem  (cost=0.00..1065544.90 rows=19993432 width=18) ...
                                 Filter: (l_shipdate <= '1998-11-30 00:00:00'::timestamp without time zone)
                                 Rows Removed by Filter: 86
...
JIT:
   Functions: 6
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 1.563 ms, Inlining 0.000 ms, Optimization 0.000 ms, Emission 0.000 ms, Total 1.563 ms
Execution Time: 13426.550 ms
(27 rows)
```

Confidential | ©2020 VMware, Inc.

Just-In-Time (JIT) Compilation

TPC-H query 1 execution times



Block Range Index - BRIN

min - max index

GREENPLUM DATABASE®

- New index type USING BRIN
- Uses metadata like minimum and maximum values about ranges of data
- Significantly less space on disk compared to standard B-tree index
- Efficiently search over large naturally sorted data
- Usable for heap, ao_row and ao_column tables
- Using pages_per_range parameter can select range for brin index entry

Confidential | ©2020 VMware, Inc.

Block Range Index - BRIN

setup

```
CREATE TABLE aotable (id int, scan float NOT NULL, created_at timestamptz NOT NULL)
with (appendonly=true, orientation=column, compresstype=zstd);
```

```
INSERT INTO aotable (id, scan, created_at) SELECT random(), random(), x
FROM generate_series('2012-01-01 0:00'::timestamptz, '2018-05-03 20:33:20'::timestamptz, '2 seconds'::interval) x;
```

```
SELECT count(*) FROM aotable;
   count
-----------
100000001
(1 row)
```

Confidential | ©2020 VMware, Inc.

Block Range Index - BRIN

creation time

```
CREATE INDEX aotable_btree_index ON aotable USING btree(created_at);
```

Time: 231246.288 ms

```
CREATE INDEX aotable_brin_index ON aotable USING brin(created_at) with(pages_per_range=1);
```

Time: 18907.757 ms

Confidential | ©2020 VMware, Inc.

Block Range Index - BRIN

data load time

```
INSERT INTO aotable (id, scan, created_at) SELECT random(), random(), x
   FROM generate_series('2012-01-01 0:00'::timestamptz, '2018-05-03
         20:33:20'::timestamptz, '2 seconds'::interval) x;
```

Block Range Index - BRIN

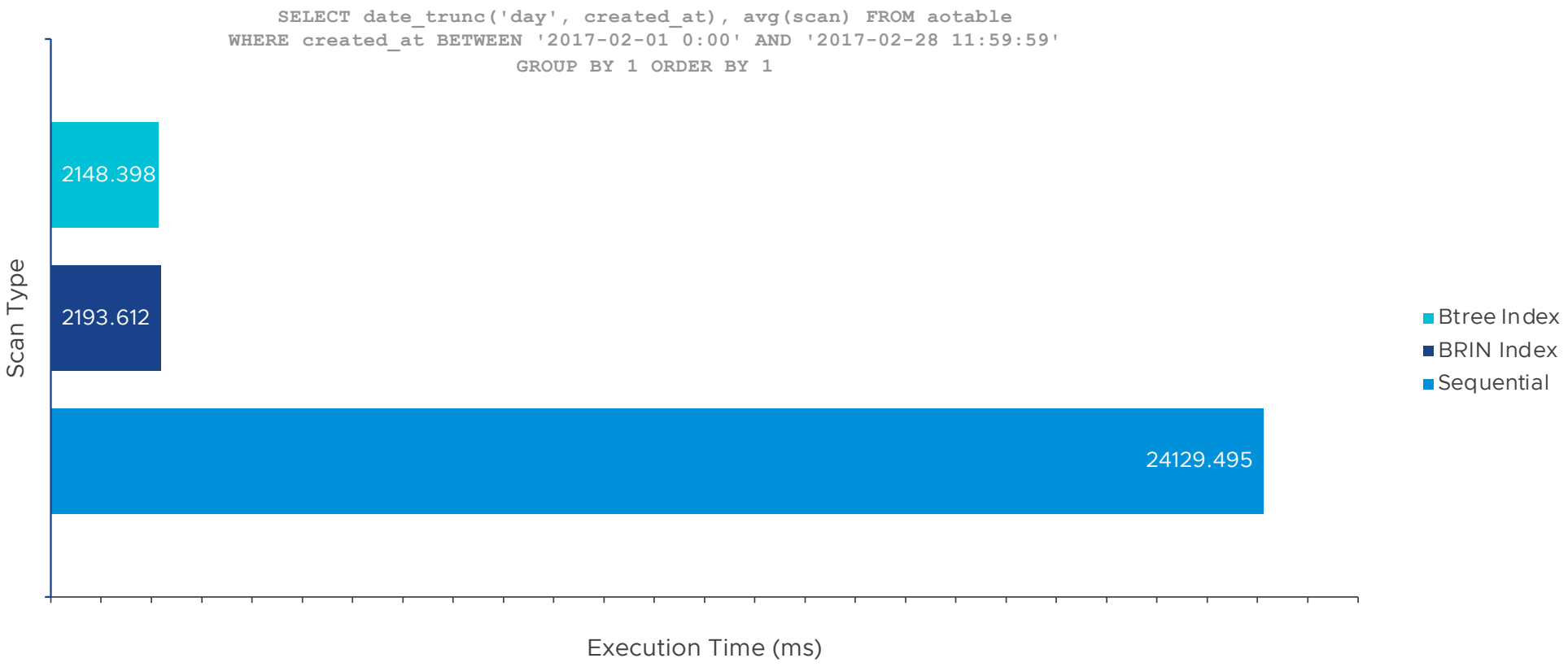
disk footprint

```
SELECT relname, relkind, pg_size_pretty(pg_relation_size(oid)) FROM pg_class WHERE relname LIKE 'aotable%' ORDER BY relname;

      relname       | relkind | pg_size_pretty

--------------------+---------+----------------

aotable              | r       | 796 MB

aotable_brin_index   | i       | 13 MB

aotable_btree_index | i       | 2125 MB

(3 rows)
```

Confidential | ©2020 VMware, Inc.

Block Range Index - BRIN

execution time



Multi-Column Statistics

helps generate better query plans

EXAMPLE: Table with Country Name and City Name will have correlation between these 2 columns; User can inform the Optimizer

GREENPLUM DATABASE

Inform the Optimizer of Correlated Columns

Functional Dependencies

Column b is functionally dependent on column a if knowledge of value of a is sufficient to determine value of b. There are no two rows having the same value of a but different values of b

Multivariate N-Distinct Counts

Estimates of number of distinct values when combining more than one column (for example, for GROUP BY a, b) are frequently wrong based on single-column statistics

Multivariate MCV Lists

Most common value lists on combinations of columns. Create MCV statistic objects only on combinations of columns used in conditions together (for example, WHERE a=<> and b=<>)

Upsert

update or insert on conflict

```
# CREATE TABLE moon(a int, b int);
CREATE TABLE

# CREATE UNIQUE INDEX ON moon(a);
CREATE INDEX

# INSERT INTO moon VALUES(1, 1);
INSERT 0 1

# BEGIN;
BEGIN
# INSERT INTO moon VALUES (1, 2), (3, 3);
psql: ERROR:  duplicate key value violates unique constraint "moon_a_idx"
DETAIL:  Key (a)=(1) already exists.
```

Allow INSERTs that would generate constraint conflicts to be turned into UPDATEs or ignored

Upsert

do nothing

```
# SELECT * FROM moon;
a | b
--+--
1 | 1
(1 row)

# INSERT INTO moon VALUES (1, 2), (3, 3) ON CONFLICT DO NOTHING;
INSERT 0 1

# SELECT * FROM moon;
a | b
--+--
3 | 3
1 | 1
(2 rows)
```

Confidential | ©2020 VMware, Inc.

Upsert

do update

```
# SELECT * FROM moon;
a | b
--+--
1 | 1
(1 row)

# INSERT INTO moon VALUES (1, 2), (3, 3) ON CONFLICT (a) DO UPDATE SET b = EXCLUDED.b;
INSERT 0 2

# SELECT * FROM moon;
a | b
--+--
3 | 3
1 | 2
(2 rows)
```

Using tablename can access current row values

Using EXCLUDED can access row values being inserted

Partitioned Tables

CREATE TABLE

Existing Syntax

```
CREATE TABLE part (a int, b int, c int)
DISTRIBUTED BY (a)
PARTITION BY range(b)
SUBPARTITION BY list(c)
(
    PARTITION p1 START (0) END (10)
    (
        SUBPARTITION sp1 VALUES (1, 2),
        SUBPARTITION sp2 VALUES (3, 4)
    ),
    PARTITION p2 START (10) END (20)
    (
        SUBPARTITION sp1 VALUES (1, 2),
        SUBPARTITION sp2 VALUES (3, 4)
    )
);
```

New Syntax

```
CREATE TABLE part (a int, b int, c int) DISTRIBUTED BY (a)
PARTITION BY range(b);

-- create first level partitions of part
CREATE TABLE part_p1 PARTITION OF part FOR VALUES FROM (0) TO (10)
PARTITION BY list(c);
CREATE TABLE part_p2 PARTITION OF part FOR VALUES FROM (10) TO (20)
PARTITION BY list(c);

-- create second level partitions inherit part_p1
CREATE TABLE part_p1_sp1 PARTITION OF part_p1 FOR VALUES IN (1, 2);
CREATE TABLE part_p1_sp2 PARTITION OF part_p1 FOR VALUES IN (3, 4);

-- create second level partitions inherit part_p2
CREATE TABLE part_p2_sp1 PARTITION OF part_p2 FOR VALUES IN (1, 2);
CREATE TABLE part_p2_sp2 PARTITION OF part_p2 FOR VALUES IN (3, 4);
```

- Partitions are first-class table
- part_p{1,2}, part_p{1,2}_sp{1,2} are actual table names, not aliases
- A partitioned table could have zero partitions
- Heterogeneous partitioning hierarchy is allowed

Partitioned Tables

ATTACH/DETACH PARTITION

```
CREATE TABLE part_new (a int, b int, c int) DISTRIBUTED BY (a) PARTITION BY range(c);
CREATE TABLE part_new_sp1 PARTITION OF part_new FOR VALUES FROM (1) TO (3);
CREATE TABLE part_new_sp2 PARTITION OF part_new FOR VALUES FROM (3) TO (4);

ALTER TABLE part ATTACH PARTITION part_new FOR VALUES FROM (100) TO (200);
ALTER TABLE part DETACH PARTITION part_p2;

\d+ part
                          Partitioned table "public.part"
 Column |  Type   | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
 a      | integer |           |          |         | plain   |              |
 b      | integer |           |          |         | plain   |              |
 c      | integer |           |          |         | plain   |              |
Partition key: RANGE (b)
Partitions: part_new FOR VALUES FROM (100) TO (200), PARTITIONED,
            part_p1 FOR VALUES FROM (0) TO (10), PARTITIONED
Distributed by: (a)
```

```
\d+ part_p1
                          Partitioned table "public.part_p1"
 Column |  Type   | Collation | Nullable | Default | Storage | Stats target |
Description
--------+---------+-----------+----------+---------+---------+--------------+----------
---
 a      | integer |           |          |         | plain   |              |
 b      | integer |           |          |         | plain   |              |
 c      | integer |           |          |         | plain   |              |
Partition of: part FOR VALUES FROM (0) TO (10)
Partition constraint: ((b IS NOT NULL) AND (b >= 0) AND (b < 10))
Partition key: LIST (c)
Partitions: part_p1_sp1 FOR VALUES IN (1, 2),
            part_p1_sp2 FOR VALUES IN (3, 4)
Distributed by: (a)

\d+ part_new
                          Partitioned table "public.part_new"
 Column |  Type   | Collation | Nullable | Default | Storage | Stats target |
Description
--------+---------+-----------+----------+---------+---------+--------------+----------
---
 a      | integer |           |          |         | plain   |              |
 b      | integer |           |          |         | plain   |              |
 c      | integer |           |          |         | plain   |              |
Partition of: part FOR VALUES FROM (100) TO (200)
Partition constraint: ((b IS NOT NULL) AND (b >= 100) AND (b < 200))
Partition key: RANGE (c)
Partitions: part_new_sp1 FOR VALUES FROM (1) TO (3),
            part_new_sp2 FOR VALUES FROM (3) TO (4)
Distributed by: (a)
```

Partitioned Tables

catalog tables & builtin functions

```
SELECT * FROM pg_partition_tree('part');
    relid     | parentrelid | isleaf | level
--------------+-------------+--------+-------
part          |             | f      |     0
part_p1       | part        | f      |     1
part_new      | part        | f      |     1
part_p1_sp1   | part_p1     | t      |     2
part_p1_sp2   | part_p1     | t      |     2
part_new_sp1  | part_new    | t      |     2
part_new_sp2  | part_new    | t      |     2
(7 rows)
```

```
SELECT pg_partition_root('part_p1_sp1');
 pg_partition_root
-------------------
 part
(1 row)
```

```
SELECT * FROM pg_partition_ancestors('part_p1_sp1');
    relid
-------------
part_p1_sp1
part_p1
part
(3 rows)
```

Confidential | ©2020 VMware, Inc.

Partitioned Tables

catalog tables & builtin functions

```
SELECT  c.relname,
        c.relkind,
        c.relispartition,
        pt.partstrat,
        pt.partattrs,
        pg_get_expr(c.relpartbound, c.oid) as partbound
FROM pg_class c
LEFT JOIN pg_partitioned_table pt
ON c.oid = pt.partrelid
WHERE pg_partition_root(c.oid) = 'part'::regclass
ORDER BY c.relname;
```

```
  relname     | relkind | relispartition | partstrat | partattrs |            partbound
-------------+---------+----------------+-----------+-----------+--------------------------------
part          | p       | f              | r         | 2         |
part_new      | p       | t              | r         | 3         | FOR VALUES FROM (100) TO (200)
part_new_sp1  | r       | t              |           |           | FOR VALUES FROM (1) TO (3)
part_new_sp2  | r       | t              |           |           | FOR VALUES FROM (3) TO (4)
part_p1       | p       | t              | l         | 3         | FOR VALUES FROM (0) TO (10)
part_p1_sp1   | r       | t              |           |           | FOR VALUES IN (1, 2)
part_p1_sp2   | r       | t              |           |           | FOR VALUES IN (3, 4)
(7 rows)
```

Confidential | ©2020 VMware, Inc.

Partitioned Tables

hash partitioning

```
CREATE TABLE hash_parted (a int, b int) DISTRIBUTED BY (a) PARTITION BY HASH (a);


-- every hash partition modulus must be a factor of the next larger modulus
CREATE TABLE hpart_1 PARTITION OF hash_parted FOR VALUES WITH (MODULUS 4, REMAINDER 0);
CREATE TABLE hpart_2 PARTITION OF hash_parted FOR VALUES WITH (MODULUS 4, REMAINDER 1);
CREATE TABLE hpart_3 PARTITION OF hash_parted FOR VALUES WITH (MODULUS 4, REMAINDER 3);

INSERT INTO hp SELECT i, i FROM generate_series(1, 1000)i;

SELECT 'hp1' AS partition, count(*) AS row_count FROM hp1
UNION ALL SELECT 'hp2', count(*) FROM hp2
UNION ALL SELECT 'hp3', count(*) from hp3;
partition | row_count
----------+-----------
hp1       |       324
hp2       |       333
hp3       |       343
(3 rows)
```

Confidential | ©2020 VMware, Inc.

Table Access Method (AM) Interface

available AMs

```
-- the catalog pg_am stores information about relation access methods.

SELECT * FROM pg_am;

oid  |  amname   |         amhandler         | amtype

-----+-----------+---------------------------+--------

   2 | heap      | heap_tableam_handler      | t

3434 | ao_row    | ao_row_tableam_handler    | t

3435 | ao_column | ao_column_tableam_handler | t

 403 | btree     | bthandler                 | i

 405 | hash      | hashhandler               | i

 783 | gist      | gisthandler               | i

2742 | gin       | ginhandler                | i

4000 | spgist    | spghandler                | i

3580 | brin      | brinhandler               | i

7013 | bitmap    | bmhandler                 | i

(10 rows)
```

Confidential | ©2020 VMware, Inc.

Table Access Method (AM) Interface

append-optimized tables

Existing Syntax

```
CREATE TABLE ao_row (a int, b varchar) WITH (appendoptimized=true,orientation=row) DISTRIBUTED BY (a);

CREATE TABLE ao_column (a int, b varchar) WITH (appendoptimized=true, orientation=column) DISTRIBUTED BY (a);
```

New Syntax

```
CREATE TABLE ao_row (a int, b varchar) USING ao_row DISTRIBUTED BY (a);

CREATE TABLE ao_column (a int, b varchar) USING ao_column DISTRIBUTED BY (a);

-- or

SET default_table_access_method TO ao_column;

CREATE TABLE ao_column_default_am (a int, b varchar) DISTRIBUTED BY (a);
```

Table Access Method (AM) Interface

append-optimized tables

```
-- create an append-optimized column-oriented table with compression using new syntax

CREATE TABLE ao_column_compressed (a int, b varchar) USING ao_column WITH (compresstype=zstd, compresslevel=5) DISTRIBUTED BY (a);

\d+ ao_column_compressed

                                                    Table "public.ao_column_compressed"

Column |       Type        | Collation | Nullable | Default | Storage  | Stats target | Compression Type | Compression Level | Block Size | Description

--------+-------------------+-----------+----------+---------+----------+--------------+------------------+-------------------+------------+-------------

a      | integer           |           |          |         | plain    |              | zstd             | 5                 | 32768      |

b      | character varying |           |          |         | extended |              | zstd             | 5                 | 32768      |

Checksum: t

Distributed by: (a)

Access method: ao_column

Options: compresstype=zstd, compresslevel=5
```

Confidential | ©2020 VMware, Inc.

Greenplum 7 VACUUM Enhancements

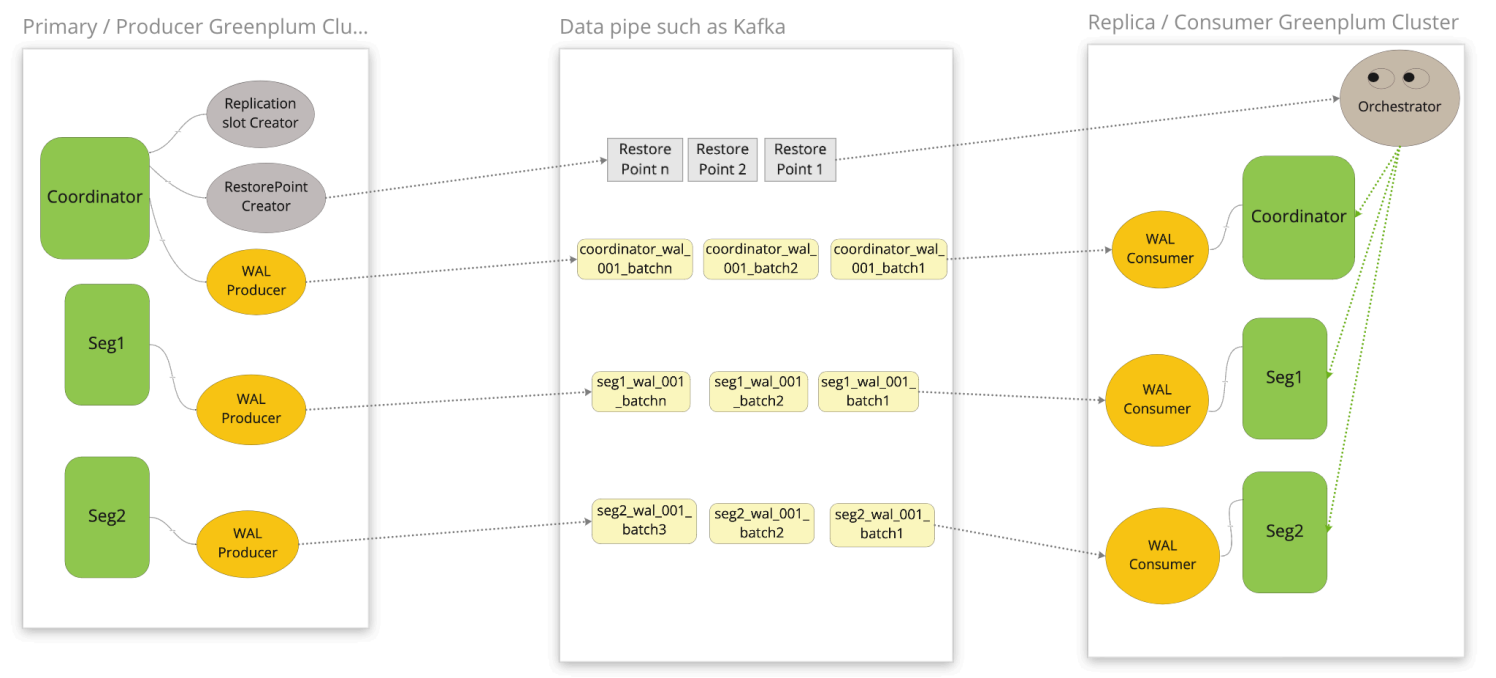
GREENPLUM DATABASE®

Reduced disk IO and locking improvements

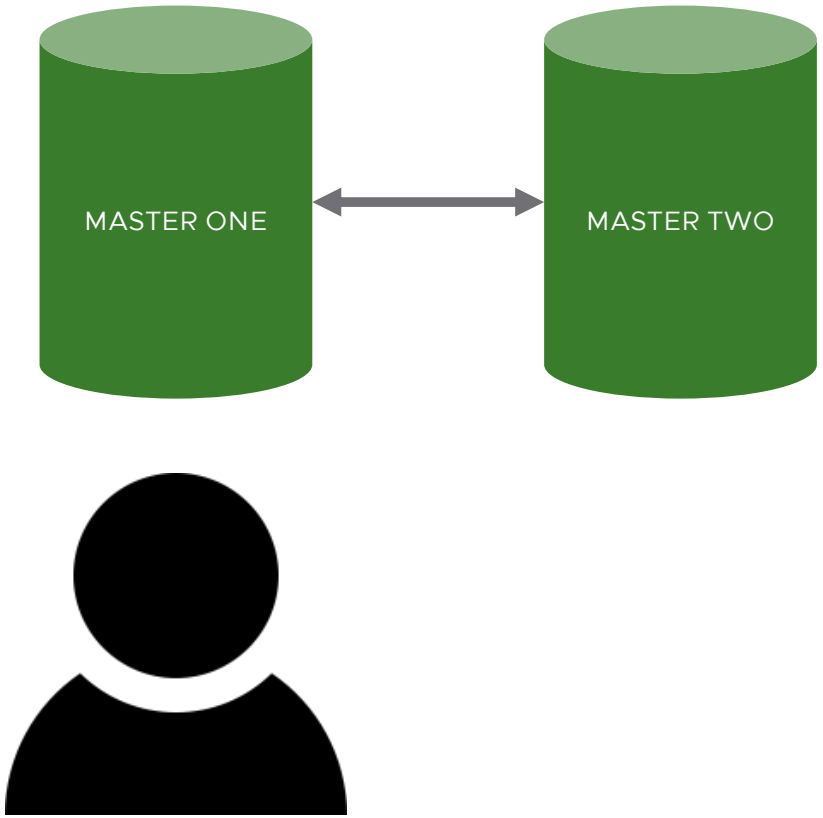
- avoid scanning and re-vacuuming pages containing only frozen tuples
- avoid unnecessary index scans on VACUUM
- avoid useless heap-truncation attempts and taking exclusive lock during VACUUM
- improve speed of VACUUM's removal of trailing empty heap pages
- INDEX_CLEANUP option to skip index cleanup if desired
- SKIP_LOCKED option to skip VACUUM and ANALYZE operations on tables that cannot be locked immediately
- vacuumdb to vacuum in parallel using new --jobs option, --skip-locked also available

Multi Site Replication

- WAL **transport** across two data centers
- Allow for user defined, consistent Restore Points across all segments
- Supports failover and failback
- Maximum data availability



Automated Master Failover

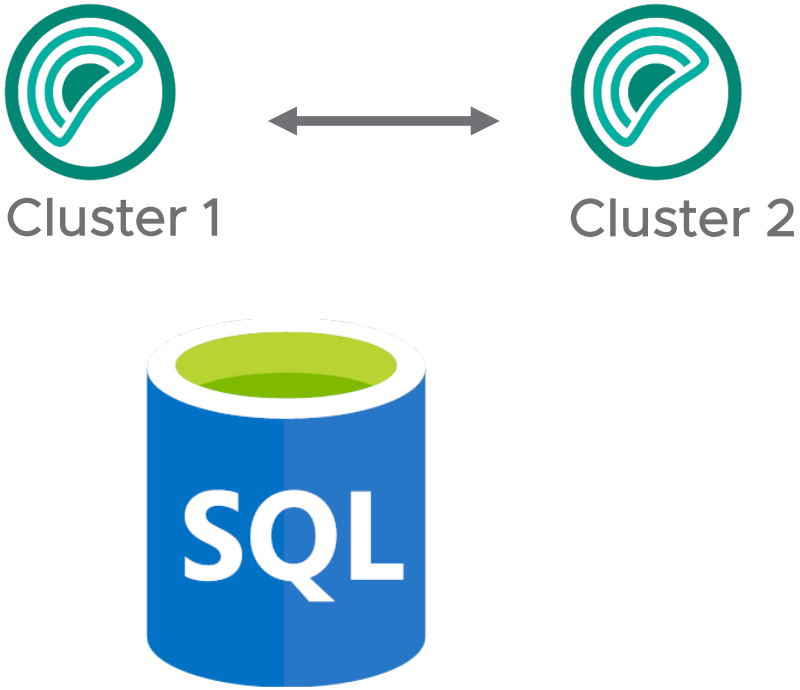


- Allows continued operation of queries without manual intervention
- Solved at software level without hardware appliance requirement
- Automated Failover, Secondary becomes Primary
- New Primary can then auto-failover to new secondary

Confidential | ©2020 VMware, Inc.

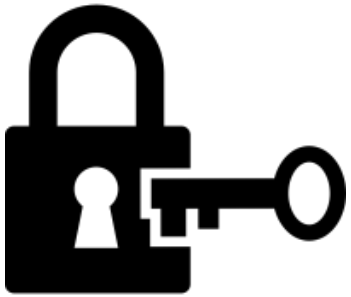
Greenplum Federation

GREENPLUM DATABASE®



- Access remote cluster with SQL
- Created a federated set of clusters
- Smart data transfer from segment to segment in parallel high speed
- Optimizer awareness for smart push downs
- Built with Foreign Data Wrapper APIs

Greenplum 7 Security Enhancement Roadmap



- Transparent Data Encryption
- File Level Encryption
- Secured Logs
- Greenplum Operator Data Protection
- Separation of Duties
- Auditing of activity

Confidential | ©2020 VMware, Inc.

GREENPLUM DATABASE®

微信技术讨论群
添加入群小助手：gp_assistant

微信公众号
技术干货、行业热点、活动预告

欢迎访问Greenplum中文社区：cn.greenplum.org