CitusDB: an Extension for Scaling out PostgreSQL

Marco Slot
marco@citusdata.com

CREATE EXTENSION citusdb;

CitusDB adds ‘Distributed’ (sharded) tables to PostgreSQL.

Makes scaling out PostgreSQL easier.

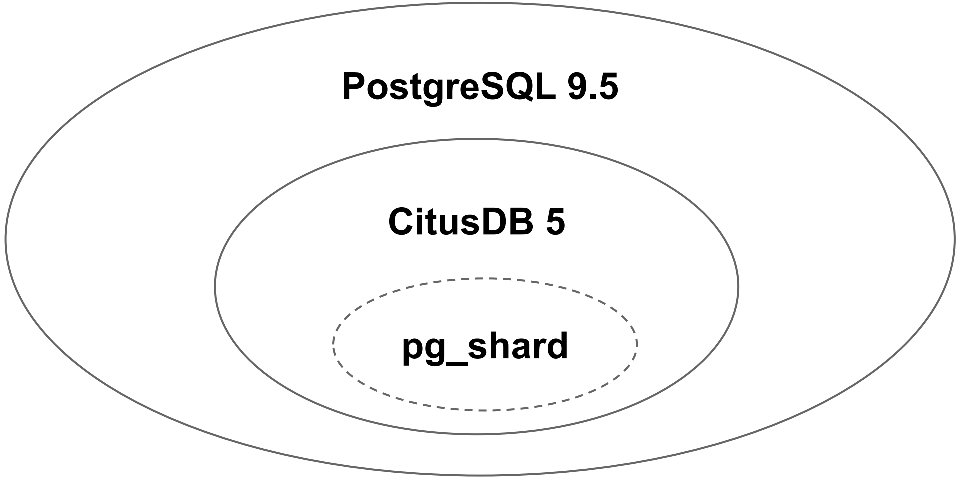
- Transparent sharding and replication across many servers
- Fault-tolerance, hides server failures from user
- Scalable data ingestion

Fast querying of ‘big data’:

- Parallel execution of queries across a cluster of PostgreSQL servers

CitusDB 5

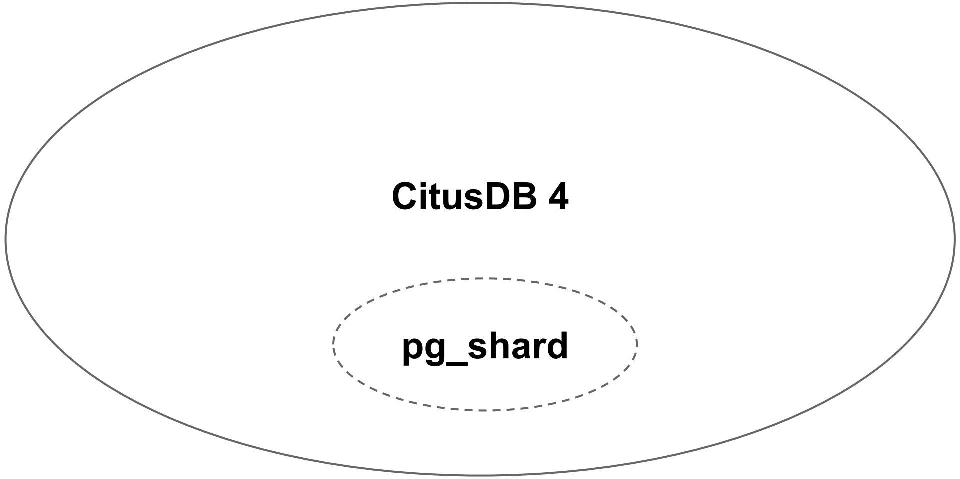
CitusDB 5 is an **extension** for PostgreSQL 9.5 (compatible with 9.4)



Will be released as open source in a few weeks!

CitusDB 4

CitusDB 4 is a slight variant of PostgreSQL 9.4

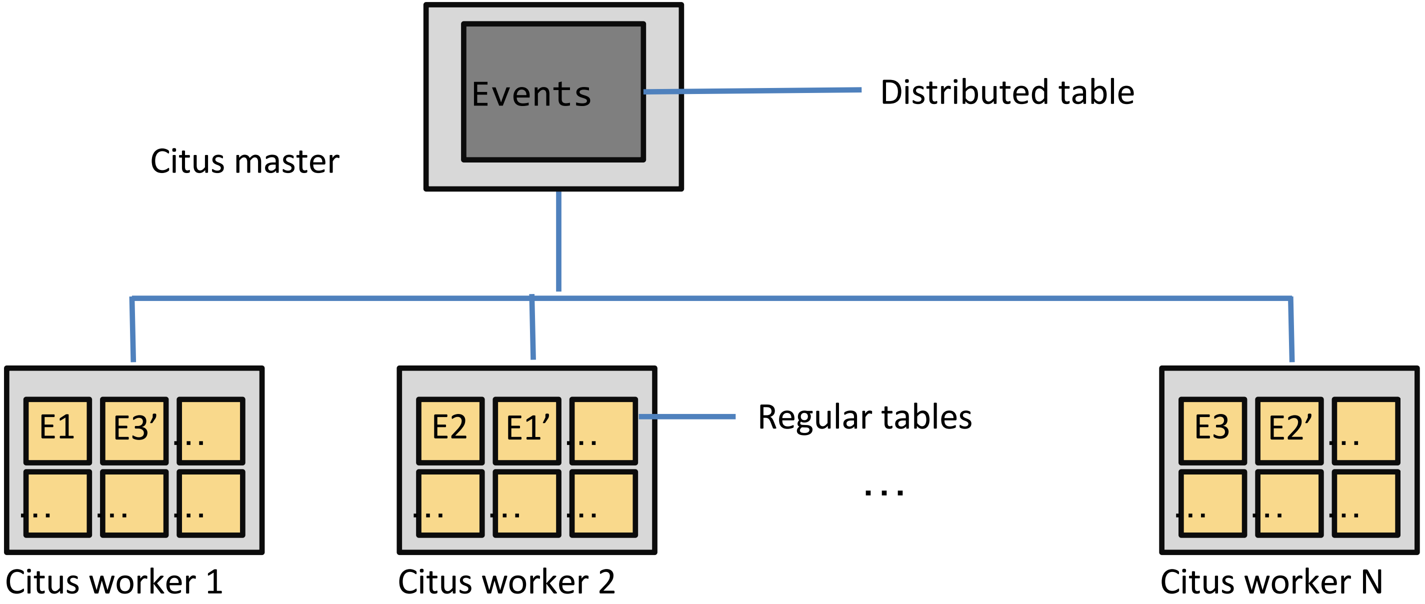


Available at citusdata.com

Citus architecture

Master node has a distributed table and keeps track of metadata on shards.

Worker nodes store shard replicas as regular PostgreSQL tables.



Getting started with Citus

Steps to set up Citus 5:

```
$ ./configure && make && sudo make install
$ cat > /db/pg_worker_list.conf
ip-10-192-0-113.eu-central-1.compute.internal 5432
ip-10-192-0-134.eu-central-1.compute.internal 5432
$ vim /db/postgresql.conf
...
shared_preload_libraries = 'citusdb'
$ pg_ctl -D /db -l /db/logfile restart
$ psql -c "CREATE EXTENSION citusdb"
```

Use-case: Analytical dashboards

Analytical dashboards let users query their data interactively.

I have relevant data for my users:

Logs (events, web logs, click streams, sensors,)

Gigabytes to Terabytes a day

I want my users to gain insight into the data:

Traffic over time

Sales by country

Changes in sign-up rates

etc.

Analytical queries

Examples:

Get number of sign-ups, grouped by day:

```
SELECT time::date AS day, count(*) FROM events
WHERE data->>'type' = 'signup' GROUP BY day ORDER BY day ASC;
```

Get the YTD revenue in Asia:

```
SELECT sum(price) FROM orders, nation WHERE orders.nation = nation.name
AND orders.date >= '2016-01-01' AND nation.region = 'Asia';
```

Analytical dashboard for big data

The trouble with analytical queries:

- Want **fast response times** (0-2 sec.)
- Queries process a **large amount of data**
- Data size, ingestion rate, query load **grows over time**
- Can’t make effective use of caching

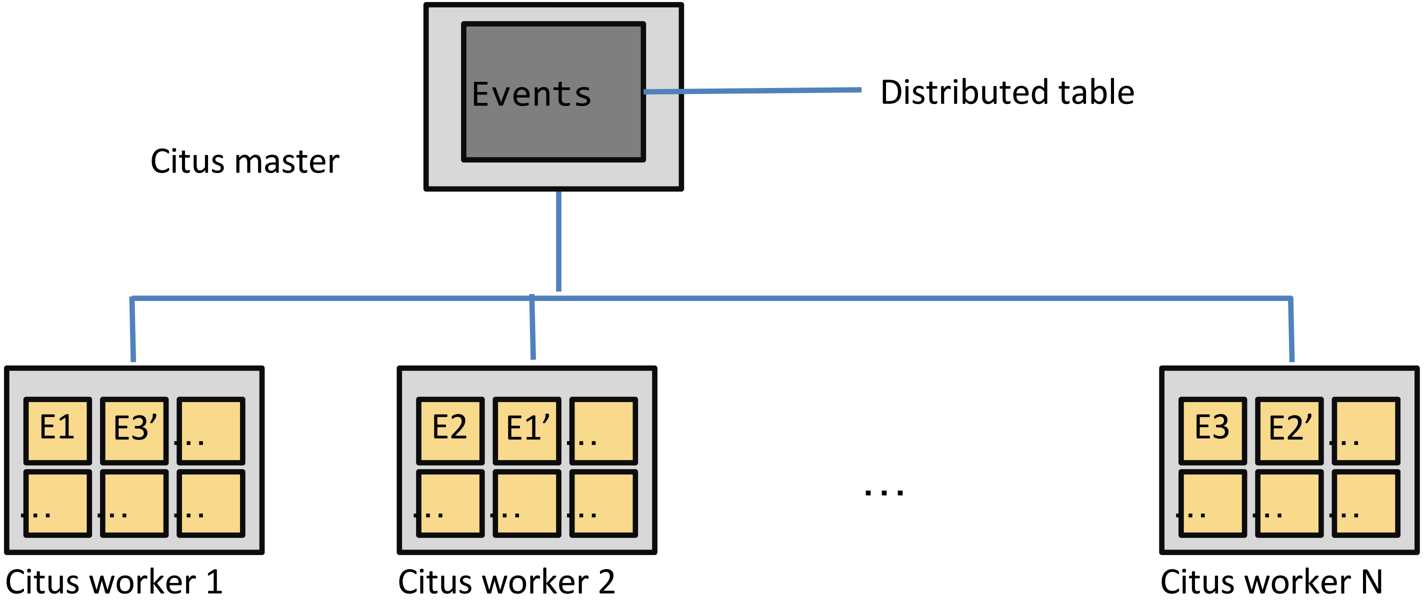
Scale out by distributing data across machines, parallelize queries across cores.

Sharding

Citus architecture

Master node has a distributed table and keeps track of metadata on shards.

Worker nodes store shard replicas as regular PostgreSQL tables.



Transparent sharding

A distributed table is created on a master node.

```
$ psql -h master-node-1
# \d
 Schema |      Name       |      Type       | Owner
--------+----------------+----------------+-------
 public | events         | table          | marco
```

Looks like an ordinary table, but does not store any data.

Instead, Citus intercepts every query to this table using the planner and executor hooks in PostgreSQL and executes the query in a distributed fashion.

Transparent sharding

Shards are stored in regular (or foreign) PostgreSQL tables on worker nodes.

```
$ psql -h worker-node-1
# \d
 Schema |      Name       |      Type       | Owner
--------+----------------+----------------+--------
 public | events_10018   | table           | marco
 public | events_10020   | table           | marco
...
```

You can query these tables as normal, but they only contain a specific subset of the data.

Sharding schemes

A distributed table can be sharded using different partitioning schemes.

- **Append-partitioned by a partition column**
  Best for bulk-loading, partitioning by time
- **Hash-partitioned by a partition column**
  Best for INSERT/UPDATE/DELETE, partitioning by ID

A 'shard' is a fragment of the data that is stored together on one or more worker nodes.

Append-partitioned tables

Each shard contains rows within a particular range of partition column values.

shard	shardminvalue	shardmaxvalue
events_10011	2015-06-09 10:00:00	2015-06-09 10:59:59
events_10012	2015-06-09 11:00:00	2015-06-09 11:59:59
events_10023	2015-06-09 12:00:00	2015-06-09 12:59:59
events_10024	2015-06-09 13:00:00	2015-06-09 13:59:59

We can create a new shard every time we add new log data (fast bulk loading)

Append-partitioned tables in Citus

```
CREATE TABLE events (
   time timestamp,
   data jsonb
);
```

Partition column

```
SELECT master_create_distributed_table('events', 'time', 'append');
CREATE INDEX ON events (time);
CREATE INDEX ON events USING GIN (data);

\STAGE events FROM '/logs/2015-06-09_10:00:00.log'
\STAGE events FROM '/logs/2015-06-09_11:00:00.log'
```

Hash-partitioned tables

Each shard contains rows with partition column values whose hash falls in a particular integer range.

shard	shardminvalue	shardmaxvalue
events_10018	-2147483648	-1073741826
events_10019	-1073741825	-3
events_10020	*-2*	*1073741820*
events_10021	1073741821	2147483647

A row with partition column value 6, hashint4(6) = *566031088* → *events_10020*

Hash-partitioned tables in Citus

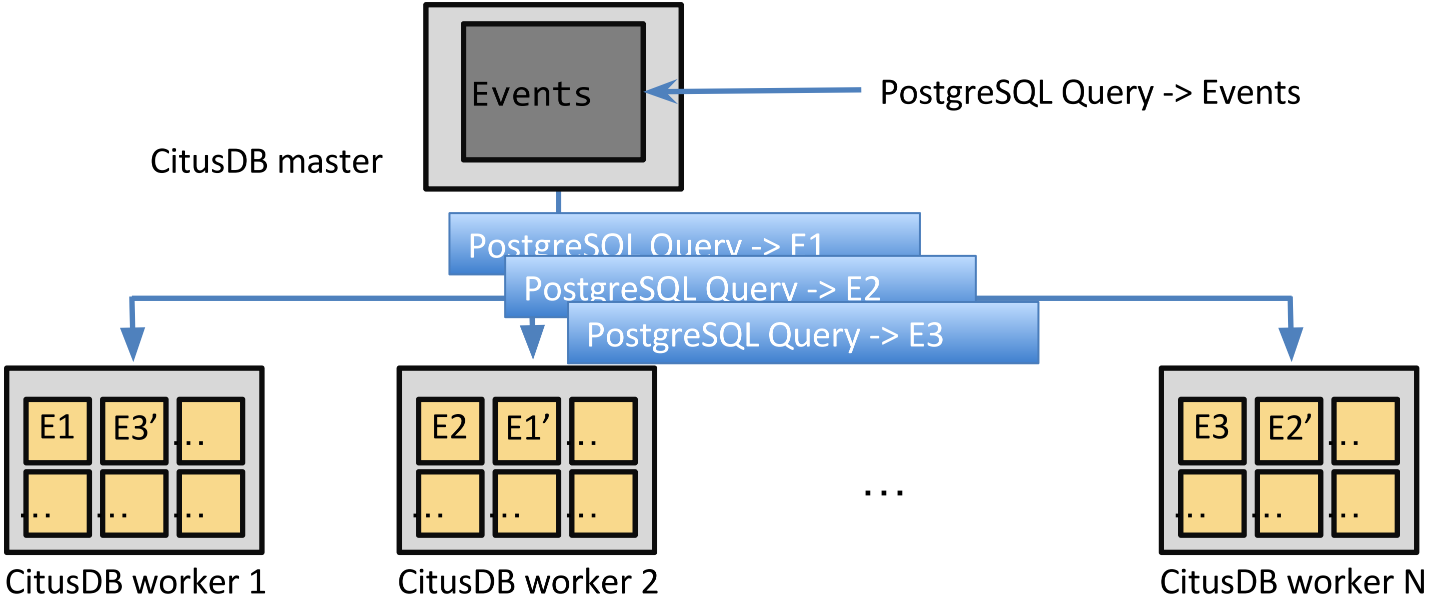
```
CREATE TABLE events (
   userid int PRIMARY KEY,
   time timestamp,
   data jsonb,
   …
);
SELECT master_create_distributed_table('events', 'userid', 'hash');
SELECT master_create_worker_shards('events', 128, 2);

INSERT INTO events VALUES(6, '2015-06-09 10:30:16', '{type:"click", ...}');
```

Parallel Querying

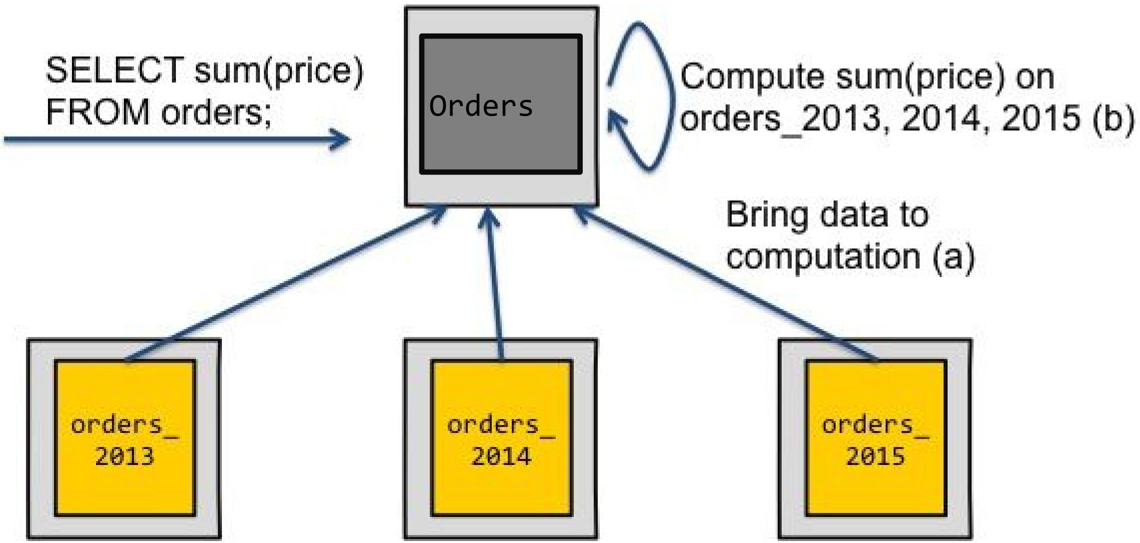
Querying sharded tables

Master sends queries on the shard tables to worker nodes and merges them into a result for the distributed table.



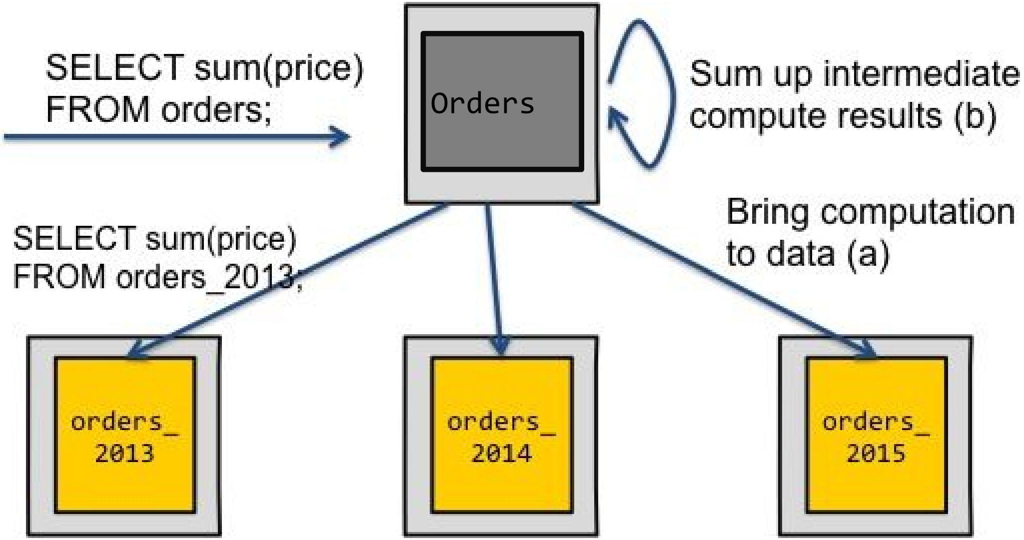
Querying sharded tables

The simple way: Bring **data to computation**



Querying sharded tables

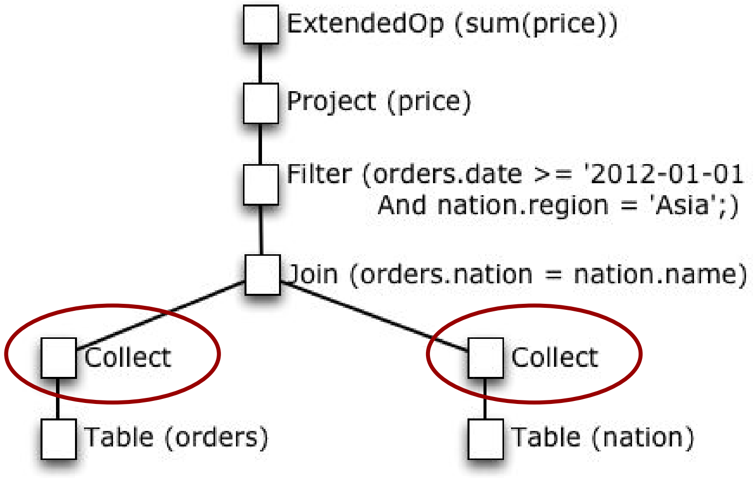
The fast (Citus) way: Bring **computation to data**



Logical planning

Building a logical plan using relational algebra for distributed query:

```
SELECT sum(price) FROM orders, nation WHERE orders.nation =
nation.name AND orders.date >= '2016-01-01' AND nation.region =
'Asia';
```



Start with simple distributed plan:

Need to make sure I have the tables.
Add a Collect operation to pull to master.

Lots of network traffic and no parallelization

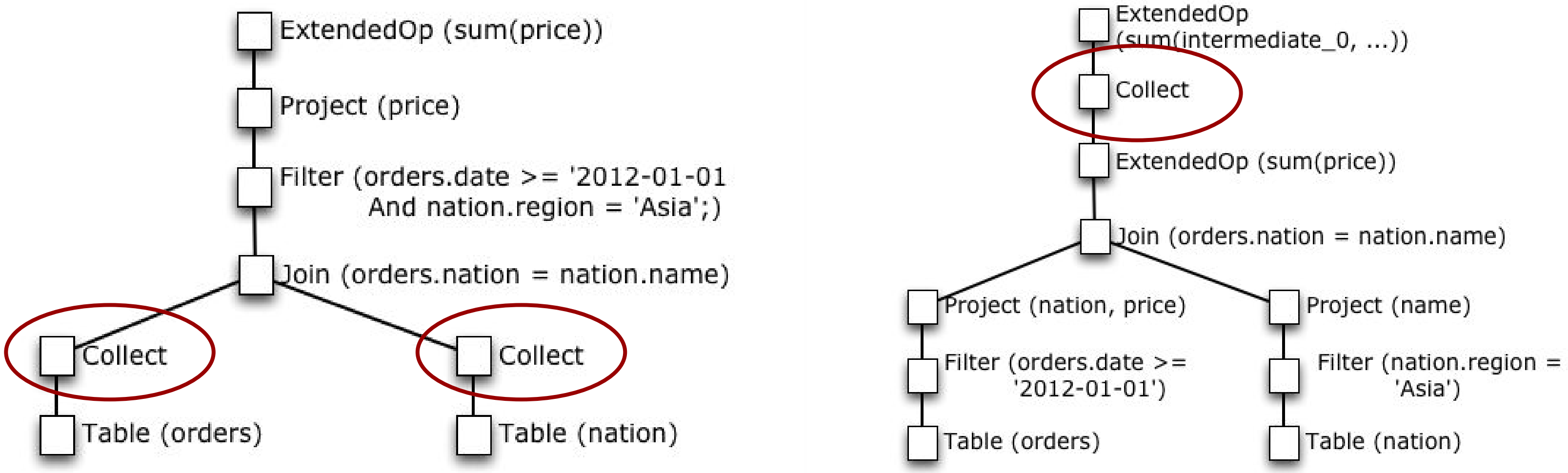
Now… let's optimize...

Logical optimization

Find computational nodes that can be pushed down, possibly with transformation.

Commutative push-down example: project of (collect of x) = collect of (project of x)

Distributive push-down example: (collect of x) join (collect of y) = collect of (x join y)



Physical planning

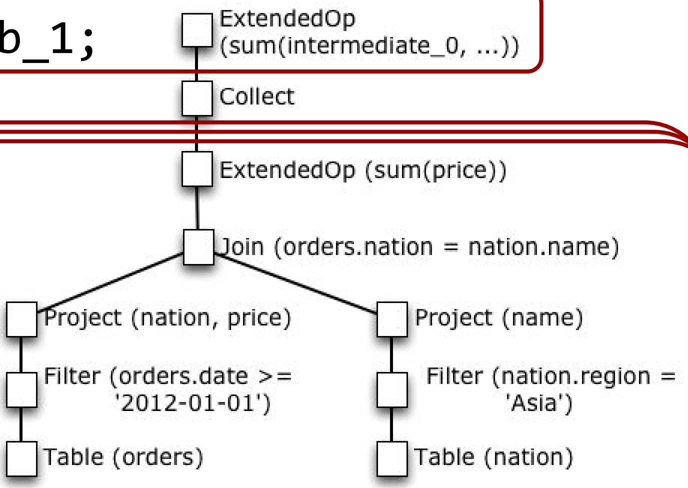
Transform the top part of the logical plan to a query to run on the master after merging.

Transform the bottom part of the logical plan to queries to run on each shard (tasks).

Skip shards that fall outside the partition column filter.

```
SELECT sum(intermediate_0) FROM merge_job_1;
```

```
SELECT sum(price)
FROM   orders_109, nation_101
WHERE  orders.date >= '2016-01-01'
AND nation.region = 'Asia'
AND orders_109.nation = nation_101.name;
```



Parallel Execution

Execution is simple:

- Send the queries to worker nodes in parallel, each using a separate connection
- *Collect* results in a merge table
- On failure, try a replica
- Finally, run the master query on the merge table

One PostgreSQL process (core) per task on the workers => Parallelization!

More complicated for re-partition joins.

Parallel Execution: Intuition

Data distribution matters:

- Distributed query execution time =
  slowest task + network overhead + master query time
- If data is not evenly distributed, parallelization is less effective

Partitioning matters:

- Queries are parallelized across the partition column dimension (e.g. 1 shard per hour, querying 1 day, can use 24 cores)
- If data is partitioned differently, queries perform differently

Shard count / cluster size matters:

- If #tasks < #cores, then the query will not use all cores
- If #tasks > #cores, then tasks are competing for resources

Limitations

SQL limitations

No union, window functions, limited subqueries

Can work around many by putting results in a temporary table

No multi-shard transactions

Requires Postgres Professional's Distributed Transaction Manager

Single master node

Masterless Citus under development

Common database features work differently

Can put triggers on shard tables, but not on distributed tables

Demo!

(sort of)

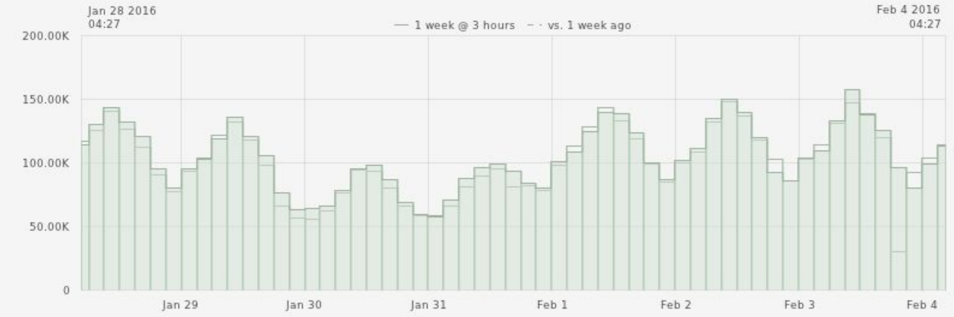
GitHub Archive ×

Marco

https://www.githubarchive.org

GitHub Archive

Star 1,120 G+1 259 Tweet

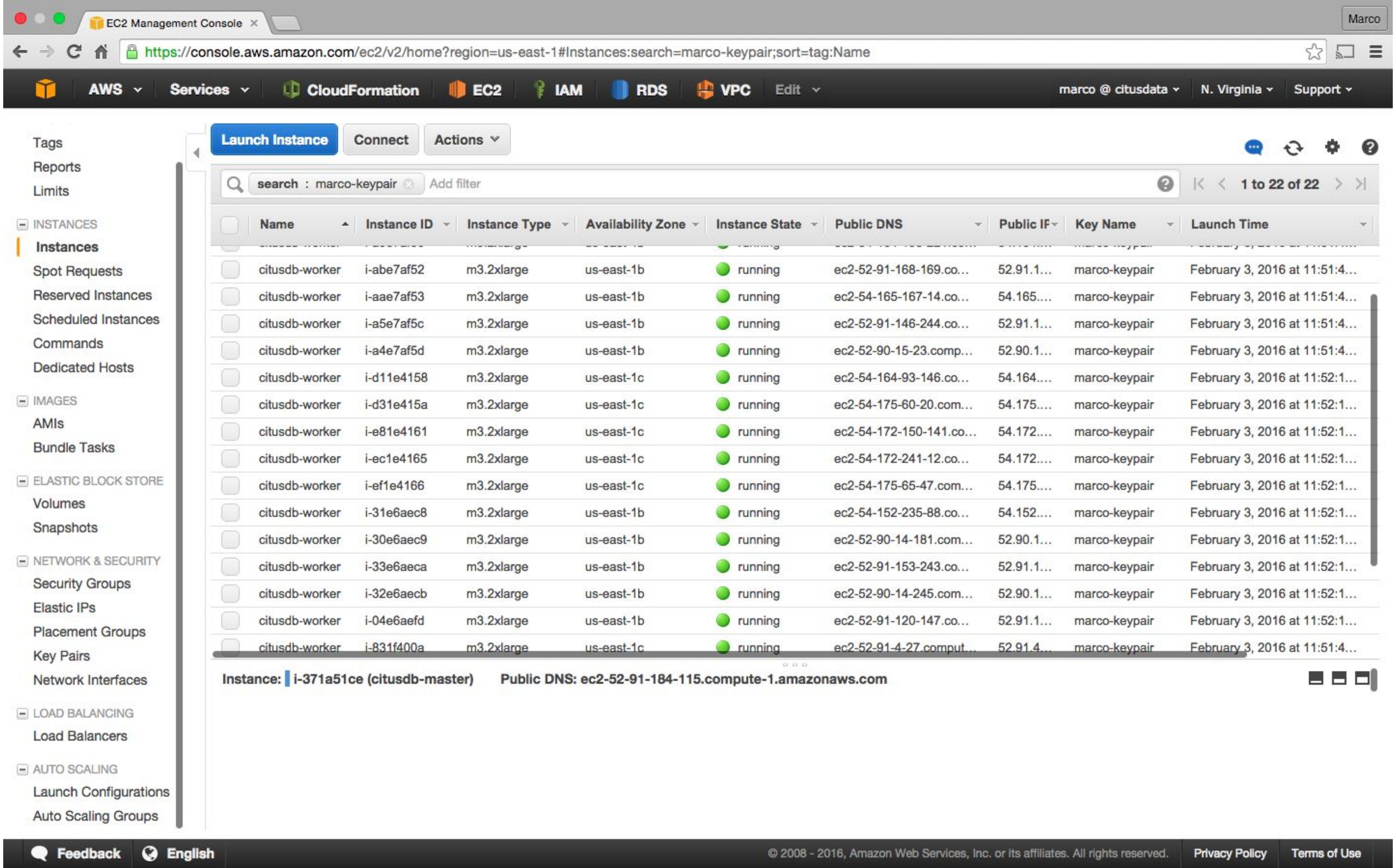


Open-source developers all over the world are working on millions of projects: writing code & documentation, fixing & submitting bugs, and so forth. GitHub Archive is a project to **record** the public GitHub timeline, **archive it**, and **make it easily accessible** for further analysis.

GitHub provides 20+ event types, which range from new commits and fork events, to opening new tickets, commenting, and adding members to a project. These events are aggregated into hourly archives, which you can access with any HTTP client:

Query	Command
Activity for 1/1/2015 @ 3PM UTC	`wget http://data.githubarchive.org/2015-01-01-15.json.gz`
Activity for 1/1/2015	`wget http://data.githubarchive.org/2015-01-01-{0..23}.json.gz`
Activity for all of January 2015	`wget http://data.githubarchive.org/2015-01-{01..30}-{0..23}.json.gz`

Each archive contains JSON encoded events as reported by the GitHub API. You can download the raw data and apply own


EC2 Management Console
Marco
https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#Instances:search=marco-keypair;sort=tag:Name
AWS
Services
CloudFormation
EC2
IAM
RDS
VPC
Edit
marco @ citusdata
N. Virginia
Support
Tags
Reports
Limits
INSTANCES
Instances
Spot Requests
Reserved Instances
Scheduled Instances
Commands
Dedicated Hosts
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
LOAD BALANCING
Load Balancers
AUTO SCALING
Launch Configurations
Auto Scaling Groups
Launch Instance
Connect
Actions
search : marco-keypair
Add filter
1 to 22 of 22
Name
Instance ID
Instance Type
Availability Zone
Instance State
Public DNS
Public IP
Key Name
Launch Time
citusdb-worker i-abe7af52 m3.2xlarge us-east-1b running ec2-52-91-168-169.co... 52.91.1... marco-keypair February 3, 2016 at 11:51:4...
citusdb-worker i-aae7af53 m3.2xlarge us-east-1b running ec2-54-165-167-14.co... 54.165.... marco-keypair February 3, 2016 at 11:51:4...
citusdb-worker i-a5e7af5c m3.2xlarge us-east-1b running ec2-52-91-146-244.co... 52.91.1... marco-keypair February 3, 2016 at 11:51:4...
citusdb-worker i-a4e7af5d m3.2xlarge us-east-1b running ec2-52-90-15-23.comp... 52.90.1... marco-keypair February 3, 2016 at 11:51:4...
citusdb-worker i-d11e4158 m3.2xlarge us-east-1c running ec2-54-164-93-146.co... 54.164.... marco-keypair February 3, 2016 at 11:52:1...
citusdb-worker i-d31e415a m3.2xlarge us-east-1c running ec2-54-175-60-20.com... 54.175.... marco-keypair February 3, 2016 at 11:52:1...
citusdb-worker i-e81e4161 m3.2xlarge us-east-1c running ec2-54-172-150-141.co... 54.172.... marco-keypair February 3, 2016 at 11:52:1...
citusdb-worker i-ec1e4165 m3.2xlarge us-east-1c running ec2-54-172-241-12.co... 54.172.... marco-keypair February 3, 2016 at 11:52:1...
citusdb-worker i-ef1e4166 m3.2xlarge us-east-1c running ec2-54-175-65-47.com... 54.175.... marco-keypair February 3, 2016 at 11:52:1...
citusdb-worker i-31e6aec8 m3.2xlarge us-east-1b running ec2-54-152-235-88.co... 54.152.... marco-keypair February 3, 2016 at 11:52:1...
citusdb-worker i-30e6aec9 m3.2xlarge us-east-1b running ec2-52-90-14-181.com... 52.90.1... marco-keypair February 3, 2016 at 11:52:1...
citusdb-worker i-33e6aeca m3.2xlarge us-east-1b running ec2-52-91-153-243.co... 52.91.1... marco-keypair February 3, 2016 at 11:52:1...
citusdb-worker i-32e6aecb m3.2xlarge us-east-1b running ec2-52-90-14-245.com... 52.90.1... marco-keypair February 3, 2016 at 11:52:1...
citusdb-worker i-04e6aefd m3.2xlarge us-east-1b running ec2-52-91-120-147.co... 52.91.1... marco-keypair February 3, 2016 at 11:52:1...
citusdb-worker i-831f400a m3.2xlarge us-east-1c running ec2-52-91-4-27.comput... 52.91.4... marco-keypair February 3, 2016 at 11:51:4...
Instance: i-371a51ce (citusdb-master)
Public DNS: ec2-52-91-184-115.compute-1.amazonaws.com
Feedback
English
© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

```
postgres=# \d events
                Table "public.events"
   Column   |            Type             | Modifiers
------------+-----------------------------+-----------
 id         | bigint                      |
 created_at | timestamp without time zone |
 type       | text                        |
 actor      | jsonb                       |
 repo       | jsonb                       |
 payload    | jsonb                       |
Indexes:
    "events_actor_idx" gin (actor jsonb_path_ops)
    "events_payload_idx" gin (payload jsonb_path_ops)
    "events_repo_idx" gin (repo jsonb_path_ops)
    "events_time_idx" btree (created_at)
    "events_type_idx" btree (type)

postgres=#
```

Pre-processing

Convert JSON data to table format:

```
CREATE TEMPORARY TABLE input (data jsonb);

COPY input FROM '$FILE' csv quote e'\x01' delimiter e'\x02’;

CREATE UNLOGGED TABLE $stage_table AS
SELECT (data->>'id'), (data->>'created_at'), (data->>'type'),
data->'actor', data->'repo', data->'payload' as payload FROM input;
```

Append to distributed table:

```
SELECT master_append_table_to_shard($shard, $stage_table, $node, $port);
```

Can do this on the worker nodes.

```
     shard      |    shardminvalue    |    shardmaxvalue
----------------+---------------------+---------------------
 events_102516 | 2015-01-01 00:00:00 | 2015-01-01 23:59:59
 events_102515 | 2015-01-02 00:00:01 | 2015-01-02 23:59:59
 events_102514 | 2015-01-03 00:00:00 | 2015-01-03 23:59:59
 events_102513 | 2015-01-04 00:00:00 | 2015-01-04 23:59:59
 events_102512 | 2015-01-05 00:00:00 | 2015-01-05 23:59:59
 events_102511 | 2015-01-06 00:00:00 | 2015-01-06 23:59:59
 events_102510 | 2015-01-07 00:00:00 | 2015-01-07 23:59:59
 events_102509 | 2015-01-08 00:00:00 | 2015-01-08 23:59:58
 events_102508 | 2015-01-09 00:00:00 | 2015-01-09 23:59:59
 events_102507 | 2015-01-10 00:00:00 | 2015-01-10 23:59:59
 events_102506 | 2015-01-11 00:00:00 | 2015-01-11 23:59:59
 events_102505 | 2015-01-12 00:00:00 | 2015-01-12 23:59:59
 events_102504 | 2015-01-13 00:00:00 | 2015-01-13 23:59:59
 events_102503 | 2015-01-14 00:00:00 | 2015-01-14 23:59:59
 events_102502 | 2015-01-15 00:00:00 | 2015-01-15 23:59:59
 events_102501 | 2015-01-16 00:00:00 | 2015-01-16 23:59:59
 events_102500 | 2015-01-17 00:00:00 | 2015-01-17 23:59:59
 events_102499 | 2015-01-18 00:00:01 | 2015-01-18 23:59:59
 events_102498 | 2015-01-19 00:00:00 | 2015-01-19 23:59:59
 events_102497 | 2015-01-20 00:00:00 | 2015-01-20 23:59:59
 events_102496 | 2015-01-21 00:00:00 | 2015-01-21 23:59:59
 events_102495 | 2015-01-22 00:00:00 | 2015-01-22 23:59:59
 events_102493 | 2015-01-23 00:00:00 | 2015-01-23 23:59:59
 events_102494 | 2015-01-24 00:00:00 | 2015-01-24 23:59:59
 events_102492 | 2015-01-25 00:00:00 | 2015-01-25 23:59:59
 events_102491 | 2015-01-26 00:00:00 | 2015-01-26 23:59:59
 events_102490 | 2015-01-27 00:00:00 | 2015-01-27 23:59:57
 events_102489 | 2015-01-28 00:00:00 | 2015-01-28 23:59:59
 events_102488 | 2015-01-29 00:00:00 | 2015-01-29 23:59:59
 events_102487 | 2015-01-30 00:00:00 | 2015-01-30 23:59:59
 events_102486 | 2015-01-31 00:00:00 | 2015-01-31 23:59:59
 events_102485 | 2015-02-01 00:00:00 | 2015-02-01 23:59:58
 events_102484 | 2015-02-02 00:00:00 | 2015-02-02 23:59:59
 events_102483 | 2015-02-03 00:00:00 | 2015-02-03 23:59:59
 events_102482 | 2015-02-04 00:00:00 | 2015-02-04 23:59:59
 events_102481 | 2015-02-05 00:00:00 | 2015-02-05 23:59:59
 events_102480 | 2015-02-06 00:00:00 | 2015-02-06 23:59:59
--More--
```

```
postgres=# SELECT date_trunc('day', created_at) AS day,
       repeat('#', (count(*)/10000)::int)
FROM   events
WHERE  type = 'PushEvent'
       AND created_at >= date '2016-01-01'
GROUP  BY day
ORDER  BY day;
         day          |                      repeat
---------------------+-------------------------------------------------
 2016-01-01 00:00:00 | ####################
 2016-01-02 00:00:00 | ########################
 2016-01-03 00:00:00 | ##########################
 2016-01-04 00:00:00 | #####################################
 2016-01-05 00:00:00 | #######################################
 2016-01-06 00:00:00 | ########################################
 2016-01-07 00:00:00 | ########################################
 2016-01-08 00:00:00 | #######################################
 2016-01-09 00:00:00 | #############################
 2016-01-10 00:00:00 | ##############################
 2016-01-11 00:00:00 | ##########################################
 2016-01-12 00:00:00 | ###########################################
 2016-01-13 00:00:00 | ############################################
 2016-01-14 00:00:00 | ############################################
 2016-01-15 00:00:00 | #########################################
 2016-01-16 00:00:00 | ###############################
 2016-01-17 00:00:00 | ################################
 2016-01-18 00:00:00 | ##########################################
 2016-01-19 00:00:00 | ###########################################
 2016-01-20 00:00:00 | ############################################
 2016-01-21 00:00:00 | ############################################
 2016-01-22 00:00:00 | ##########################################
 2016-01-23 00:00:00 | ################################
 2016-01-24 00:00:00 | #################################
 2016-01-25 00:00:00 | ############################################
 2016-01-26 00:00:00 | ##############################################
 2016-01-27 00:00:00 | ###############################################
 2016-01-28 00:00:00 | #########################################
 2016-01-29 00:00:00 | #########################################
 2016-01-30 00:00:00 | #################################
 2016-01-31 00:00:00 | ###################################
 2016-02-01 00:00:00 | ###########################################
 2016-02-02 00:00:00 | #############################################
 2016-02-03 00:00:00 | ##############
(34 rows)

Time: 944.937 ms
postgres=#
```

using distributed table

~950ms

```
postgres=# SELECT date_trunc('day', created_at) AS day,
       repeat('#', (count(*)/10000)::int)
FROM   events_local
WHERE  type = 'PushEvent'
       AND created_at >= date '2016-01-01'
GROUP  BY day
ORDER  BY day;
         day          |                      repeat
----------------------+------------------------------------------------
 2016-01-01 00:00:00 | ####################
 2016-01-02 00:00:00 | ########################
 2016-01-03 00:00:00 | ##########################
 2016-01-04 00:00:00 | #####################################
 2016-01-05 00:00:00 | #######################################
 2016-01-06 00:00:00 | ########################################
 2016-01-07 00:00:00 | ########################################
 2016-01-08 00:00:00 | #######################################
 2016-01-09 00:00:00 | #############################
 2016-01-10 00:00:00 | ##############################
 2016-01-11 00:00:00 | ##########################################
 2016-01-12 00:00:00 | ###########################################
 2016-01-13 00:00:00 | ############################################
 2016-01-14 00:00:00 | ############################################
 2016-01-15 00:00:00 | #########################################
 2016-01-16 00:00:00 | ###############################
 2016-01-17 00:00:00 | ################################
 2016-01-18 00:00:00 | ##########################################
 2016-01-19 00:00:00 | ###########################################
 2016-01-20 00:00:00 | ############################################
 2016-01-21 00:00:00 | ############################################
 2016-01-22 00:00:00 | ##########################################
 2016-01-23 00:00:00 | ################################
 2016-01-24 00:00:00 | #################################
 2016-01-25 00:00:00 | ############################################
 2016-01-26 00:00:00 | ##############################################
 2016-01-27 00:00:00 | ###############################################
 2016-01-28 00:00:00 | #########################################
 2016-01-29 00:00:00 | #########################################
 2016-01-30 00:00:00 | #################################
 2016-01-31 00:00:00 | ###################################
 2016-02-01 00:00:00 | ###########################################
 2016-02-02 00:00:00 | #############################################
 2016-02-03 00:00:00 | ##############
(34 rows)

Time: 111275.136 ms
postgres=#
```

using local table

~2 minutes

```
postgres=# SELECT jsonb_array_elements(payload->'commits')->'author'->>'name' AS name,
       count(*)
FROM   events
WHERE  repo @> '{"name":"postgres/postgres"}'
GROUP  BY name
ORDER  BY 2 DESC;
        name        | count
--------------------+-------
 Tom Lane           |  1691
 Robert Haas        |   339
 Heikki Linnakangas |   312
 Andres Freund      |   306
 Alvaro Herrera     |   306
 Noah Misch         |   254
 Bruce Momjian      |   229
 Peter Eisentraut   |   228
 Stephen Frost      |   137
 Andrew Dunstan     |   122
 Fujii Masao        |   107
 Michael Meskes     |    65
 Magnus Hagander    |    57
 Joe Conway         |    41
 Simon Riggs        |    40
 Teodor Sigaev      |    39
 Kevin Grittner     |    38
 Tatsuo Ishii       |    23
 Greg Stark         |    14
 Jeff Davis         |     4
(20 rows)

Time: 266.441 ms
postgres=#
```

using GIN index

~270ms

```
postgres=# SELECT date_trunc('month', created_at) AS month,
       count(*)
FROM   events
WHERE  type = 'PushEvent'
GROUP  BY month
ORDER  BY month;
        month        |  count
---------------------+----------
 2015-01-01 00:00:00 |  7028554
 2015-02-01 00:00:00 |  7242981
 2015-03-01 00:00:00 |  8616370
 2015-04-01 00:00:00 |  8510359
 2015-05-01 00:00:00 |  8499153
 2015-06-01 00:00:00 |  8208818
 2015-07-01 00:00:00 |  8050175
 2015-08-01 00:00:00 |  8198493
 2015-09-01 00:00:00 |  9246127
 2015-10-01 00:00:00 |  9990779
 2015-11-01 00:00:00 | 10314029
 2015-12-01 00:00:00 | 10482194
 2016-01-01 00:00:00 | 11403552
 2016-02-01 00:00:00 |  1041518
(14 rows)

Time: 2863.495 ms
postgres=#
```

scan >100m PushEvents

2.9 seconds

```
postgres=# SELECT count(*) FROM events WHERE created_at >= current_timestamp - interval '1' month;
  count
----------
 22745687
(1 row)

Time: 1492.376 ms
postgres=# SELECT count(*) FROM events WHERE created_at >= current_timestamp - interval '2' month;
  count
----------
 41907532
(1 row)

Time: 1612.991 ms
postgres=# SELECT count(*) FROM events WHERE created_at >= current_timestamp - interval '3' month;
  count
----------
 63325883
(1 row)

Time: 1722.621 ms
postgres=# SELECT count(*) FROM events WHERE created_at >= current_timestamp - interval '4' month;
  count
----------
 84091873
(1 row)

Time: 1995.270 ms
postgres=# SELECT count(*) FROM events WHERE created_at >= current_timestamp - interval '5' month;
   count
-----------
 102527425
(1 row)

Time: 2287.337 ms
postgres=#
```

1 month -> 31 cores, 1.5 seconds

3 months -> 80 cores, 1.7 seconds

5 months -> 80 cores, 2.3 seconds

Comparison to other systems

Distributed database spectrum:

Postgres-XL		Citus	Greenplum
OLTP	NoSQL + Analytics	Real-time Analytics	Data warehouse

Summary

CitusDB: an Extension for Scaling out PostgreSQL

Discussed different ways of (transparently) sharding your database:

- Append-partitioning — Best for bulk-loading, partitioning by time
- Hash-partitioning — Best for INSERT, partitioning by ID

Parallel execution logic:

- Translate a single distributed query into multiple local queries using logical planning
- Push down computation to workers based on commutativity

Questions?

Marco - marco@citusdata.com

General - engage@citusdata.com

https://www.citusdata.com/blog

```
postgres=# SELECT payload->'commits'->0->'author'->>'name' AS name, date_trunc('month', created_at) AS month,
       repeat('#', count(*)::int)
FROM   events
WHERE  repo @> '{"name":"postgres/postgres"}' AND payload->'commits'->0->'author' @> '{"name":"Andres Freund"}'
GROUP  BY name, month
ORDER  BY month ASC;
     name      |        month        |                    repeat
---------------+---------------------+-----------------------------------------------
 Andres Freund | 2015-01-01 00:00:00 | #############################################
 Andres Freund | 2015-02-01 00:00:00 | #########################
 Andres Freund | 2015-03-01 00:00:00 | #########
 Andres Freund | 2015-04-01 00:00:00 | ###########
 Andres Freund | 2015-05-01 00:00:00 | #############
 Andres Freund | 2015-06-01 00:00:00 | ##############
 Andres Freund | 2015-07-01 00:00:00 | ###################
 Andres Freund | 2015-08-01 00:00:00 | ######################################
 Andres Freund | 2015-09-01 00:00:00 | ##################
 Andres Freund | 2015-10-01 00:00:00 | ######################
 Andres Freund | 2015-11-01 00:00:00 | #######
 Andres Freund | 2015-12-01 00:00:00 | #######################
(12 rows)

Time: 470.435 ms
postgres=#
```

Citus Data team outing :)