

Sharding with postgres_fdw

Postgres Open 2013
Chicago

Stephen Frost
sfrost@snowman.net

Stephen Frost

- PostgreSQL
 - Major Contributor, Committer
 - Implemented Roles in 8.3
 - Column-Level Privileges in 8.4
 - Contributions to PL/pgSQL, PostGIS
- Resonate, Inc.
 - Principal Database Engineer
 - Online Digital Media Company
 - We're Hiring! - techjobs@resonateinsights.com

Do you read . . .

- planet.postgresql.org

What is an FDW?

- First, SQL/MED
 - SQL/ Management of External Data
 - Standard to allow integration with external data
 - Foreign data can be nearly anything:
 - SQL Databases, CSV Files, Text Files,
 - NoSQL Databases, Cacheing systems, etc..
- Defines the notion of a 'FOREIGN TABLE'
 - Foreign tables are "views" to external data
 - No data is stored in the DB

What is an FDW? (part 2)

- FDWs are the back-end piece to support SQL/MED
- PostgreSQL provides a generic FDW API
- An FDW is a PG EXTENSION implementing the API
 - PG Extensions already exist for:
 - RDMS's: Oracle, MySQL, ODBC, JDBC
 - NoSQL's: CouchDB, Mongo, Redis
 - Files: CSV, Text, even JSON
 - "Other": Twitter, HTTP
- Our focus will be on (ab)using `postgres_fdw`

Basics of FDW connections

- Connecting to another actual RDBMS is complicated
 - CREATE FOREIGN SERVER
 - CREATE USER MAPPING
 - CREATE FOREIGN TABLE
- 'SERVER' provides a name and options to connect
- 'USER' maps the local user to the remote user
- 'TABLE' defines:
 - A local TABLE object, with columns, etc
 - A remote TABLE (through a FOREIGN SERVER)
- Connecting with a file FDW is simpler (no user map)

Using postgres_fdw

- CREATE EXTENSION postgres_fdw;

```
CREATE FOREIGN SERVER shard01  
FOREIGN DATA WRAPPER postgres_fdw  
OPTIONS (host 'shard01', dbname 'mydb', ...)
```

- All libpq options accepted except user/pw
- User/PW is specified in user mappings
- Cost options (fdw_startup_cost, fdw_tuple_cost)

Creating User Mappings

```
CREATE USER MAPPING FOR myrole  
SERVER shard01  
OPTIONS (user 'myrole', password 'abc123')
```

- Only takes user and password options
- User mappings are tied to servers
- User must exist on client and server
- Must use a password for non-superuser maps

Creating Foreign Tables

```
CREATE FOREIGN TABLE mytable_shard01 (  
  a int OPTIONS (column_name 'b'),  
  b int OPTIONS (column_name 'a'), ...  
SERVER shard01  
OPTIONS (table_name 'mytable');
```

- Can pick remote schema, remote table, and remote column
- These don't have to match the local system
- Very important for sharding

Remote Query Execution

- Each backend manages its own remote connections
- When a foreign table is queried:
 - PG opens a connection to the remote server
 - Starts a transaction on the remote server
 - A cursor is created for the query
 - WHERE clauses are pushed to remote server
 - Data is pulled through the remote cursor when rows are requested during query execution

More on Query Execution

- The remote transaction ends when the local transaction ends
 - Rolls back or commits based on local transaction
 - Rows inserted are not visible on remote until the local transaction completes
 - Be careful of 'idle in transaction' connections..
- Connections are kept after the foreign query
 - Re-used for later requests to the same server
 - No explicit limit on number of connections
 - Each connection uses up memory, of course.

Query costing with FDWs

- Approach to costing can be changed
- Options can be set at server or table level
- `fdw_startup_cost` and `fdw_tuple_cost`
- `use_remote_estimate` - false (default)
 - Looks up statistics for the table locally
 - Statistics updated with `ANALYZE`
- `use_remote_estimate` - true
 - Queries the remote server to determine cost info
 - Uses `EXPLAIN` on remote side
- `ANALYZE` your tables!

Sharding

- What is sharding?
 - Horizontal partitioning across servers
 - Break up large tables based on a key/range
 - Replicate small / common data across nodes
- Why sharding?
 - Allows (more) parallelization of work
 - Scales beyond a single server
- Challenges
 - Data consistency
 - Difficult to query against

Dealing with 32 shards

- Why 32?
 - Pre-sharding
 - Only 8 physical servers
 - Four clusters per node
 - Too many to manage manually
- Script everything
 - Building the clusters
 - User/role creation
 - Table creation, etc, etc..
- Use a CM System (Puppet, Chef, etc.)

Sharding suggestions

- Still partition on shards
 - Smaller tables, smaller indexes
 - Use inheritance and CHECK constraints
 - Foreign tables can use parent tables
- Break up sequence spaces
 - Define a range for each shard
 - Put constraints to ensure correct sequence used
 - Consider one global sequence approach

FDW Challenges

- Not parallelized!
 - Queries against foreign tables are done serially
 - Transactions commit with the head node
- What is pushed down and what isn't?
 - Conditionals
 - Only built-in data types, operators, functions
 - Joins aren't (yet...)
- Not able to call remote functions directly
- Foreign Tables are one-to-one
- Inserts go to *all* columns (can't have defaults..)

Parallelizing

- Need an independent "job starting" process
 - cron
 - pgAgent
 - Daemon w/ LISTEN/NOTIFY
- Use triggers on remote tables to NOTIFY
- View / Manage jobs through the head node
- Custom background worker...?

Working through FDWs

- Use lots of views
 - Script building them
 - UNION ALL is your friend
 - Add constants/conditionals to view's query
 - Use DO-INSTEAD rules for updates
 - Put them on foreign system too for joins, etc
- Get friendly with triggers
 - Use them to run remote procedures
 - Remember that everything is serial!
- Bottlenecks, network latency can be a factor

View Example

```
CREATE FOREIGN TABLE workflow.jobs_shard1
  ( workflow_name text, name text, state text )
  SERVER shard1 OPTIONS (schema_name 'workflow', table_name 'jobs');
...

CREATE FOREIGN TABLE workflow.workflow_shard1
  ( name text, state text )
  SERVER shard1 OPTIONS (schema_name 'workflow', table_name 'workflow');
...
```

```
CREATE VIEW workflow.workflow AS
SELECT 'shard1'::text AS shard, * FROM workflow_shard.workflow_shard1 UNION ALL
SELECT 'shard2'::text AS shard, * FROM workflow_shard.workflow_shard2 UNION ALL
SELECT 'shard3'::text AS shard, * FROM workflow_shard.workflow_shard3 ...
```

```
CREATE VIEW workflow.jobs AS
SELECT 'shard1'::text AS shard, * FROM workflow_shard.jobs_shard1 UNION ALL
SELECT 'shard2'::text AS shard, * FROM workflow_shard.jobs_shard2 UNION ALL
SELECT 'shard3'::text AS shard, * FROM workflow_shard.jobs_shard3 ...
```

What 's PG do?

- Let's try a join..

```
EXPLAIN SELECT * FROM workflow
                JOIN jobs
                ON (workflow.shard = jobs.shard and workflow.name = workflow_name);
```

QUERY PLAN

```
-----
Merge Join (cost=13235.29..41555.08 rows=1878610 width=224)
  Merge Cond: (((shard1)::text) = (shard1)::text) AND (workflow_shard1.name = jobs_shard1.workflow_name)
    -> Sort (cost=6367.75..6410.79 rows=17216 width=128)
        Sort Key: (shard1)::text, jobs_shard1.workflow_name
        -> Append (cost=100.00..4036.48 rows=17216 width=128)
            -> Foreign Scan on jobs_shard1 (cost=100.00..126.14 rows=538 width=128)
            -> Foreign Scan on jobs_shard2 (cost=100.00..126.14 rows=538 width=128)
            -> Foreign Scan on jobs_shard3 (cost=100.00..126.14 rows=538 width=128)
            -> Foreign Scan on jobs_shard4 (cost=100.00..126.14 rows=538 width=128)
            .....
        -> Materialize (cost=6867.54..6976.66 rows=21824 width=96)
            -> Sort (cost=6867.54..6922.10 rows=21824 width=96)
                Sort Key: (shard1)::text, workflow_shard1.name
                -> Append (cost=100.00..4174.72 rows=21824 width=96)
                    -> Foreign Scan on workflow_shard1 (cost=100.00..130.46 rows=682 width=96)
                    -> Foreign Scan on workflow_shard2 (cost=100.00..130.46 rows=682 width=96)
                    -> Foreign Scan on workflow_shard3 (cost=100.00..130.46 rows=682 width=96)
                    .....
    (73 rows)
```

Playing with views

- Looking at one shard..

```
EXPLAIN SELECT * FROM workflow
              JOIN jobs
              ON (workflow.shard = jobs.shard and workflow.name = workflow_name)
              WHERE workflow.shard = 'shard1';
```

QUERY PLAN

```
-----
Hash Join (cost=232.86..329.41 rows=1835 width=224)
  Hash Cond: (workflow_shard1.name = jobs_shard1.workflow_name)
  -> Append (cost=100.00..130.46 rows=682 width=96)
    -> Foreign Scan on workflow_shard1 (cost=100.00..130.46 rows=682 width=96)
  -> Hash (cost=126.14..126.14 rows=538 width=128)
    -> Append (cost=100.00..126.14 rows=538 width=128)
      -> Foreign Scan on jobs_shard1 (cost=100.00..126.14 rows=538 width=128)
(7 rows)
```

- Much better, but means you have to remember...
- Still works through prepared queries

Verbose

- Shows the query to be sent

```
EXPLAIN (verbose) SELECT * FROM workflow
      JOIN jobs
      ON (workflow.shard = jobs.shard and workflow.name = workflow_name)
      WHERE workflow.shard = 'shard1';
```

QUERY PLAN

```
-----
Hash Join (cost=232.86..329.41 rows=1835 width=224)
  Output: ('shard1'::text), workflow_shard1.name, workflow_shard1.state,
         ('shard1'::text), jobs_shard1.workflow_name, jobs_shard1.name, jobs_shard1.state
  Hash Cond: (workflow_shard1.name = jobs_shard1.workflow_name)
  -> Append (cost=100.00..130.46 rows=682 width=96)
    -> Foreign Scan on workflow_shard.workflow_shard1 (cost=100.00..130.46 rows=682 width=96)
        Output: 'shard1'::text, workflow_shard1.name, workflow_shard1.state
        Remote SQL: SELECT name, state FROM workflow.workflow
    -> Hash (cost=126.14..126.14 rows=538 width=128)
        Output: ('shard1'::text), jobs_shard1.workflow_name, jobs_shard1.name, jobs_shard1.state
        -> Append (cost=100.00..126.14 rows=538 width=128)
            -> Foreign Scan on workflow_shard.jobs_shard1 (cost=100.00..126.14 rows=538 width=128)
                Output: 'shard1'::text, jobs_shard1.workflow_name, jobs_shard1.name, jobs_shard1.state
                Remote SQL: SELECT workflow_name, name, state FROM workflow.jobs

(13 rows)
```

Firing a remote procedure

- Have to set it up as an INSERT trigger
- Arguments and result end up being columns

On the shards:

```
CREATE TABLE fire_func ( id bigint, a int, b int, result int );
CREATE FUNCTION add_two () RETURNS trigger AS $_$
    begin new.result = new.a + new.b; return new; end; $_$
LANGUAGE plpgsql;
CREATE TRIGGER add_two_trig BEFORE INSERT ON fire_func FOR EACH ROW EXECUTE PROCEDURE add_two();
```

On the head node:

```
CREATE FOREIGN TABLE fire_func_shard1 ( id bigint, a int, b int, result int )
    SERVER shard1 OPTIONS (schema_name 'workflow', table_name 'fire_func');
CREATE FOREIGN TABLE fire_func_shard2 ( id bigint, a int, b int, result int )
    SERVER shard2 OPTIONS (schema_name 'workflow', table_name 'fire_func');
...
```

```
=# insert into fire_func_shard2 (id, a, b) values (100, 1, 2) returning id, a, b, result;
 id | a | b | result
-----+-----+-----+-----
 100 | 1 | 2 |      3
(1 row)
```

Managing foreign tables

- Scripts, ideally generalized
 - Generating foreign tables
 - Building views
- Use a schema migration system
 - Roll-your-own
 - External options (Sqitch, etc)
- Use "foreign schemas"

Improvements for FDWs

- Parallelize work
 - Make Append() send all FDW queries at once
 - Use a round-robin approach to pulling data
 - Buffer results
- Better user management
 - Credential proxying
 - Automatic user maps
 - Trusted inter-server connections

More idle thoughts

- Make UNION ALL views updatable
- Inheritance for foreign tables
- Auto-discover foreign table definition
- Join push-down
- Scripting the server/user map/table creation
- Building views over the foreign tables
- How views are implemented / run by PG
- Build system to trigger actions off of a table update
- Managing workflows, external processes
- *REAL PARTITIONING*

Thank you!

Stephen Frost
sfrost@snowman.net
[@net_snow](#)