HypoPG Documentation

Julien Rouhaud

Sep 10, 2023

Contents:

HypoPG is a PostgreSQL extension, adding support for *Hypothetical Indexes*. It's compatible with **PostgreSQL 9.2 and above**.

Note: This documentation is a work in progress. If you're looking for something and can't find it here, please report an issue so I can enhance the documentation.

CHAPTER 1

Hypothetical Indexes

A hypothetical, or virtual, index is an index that does not really exist, and therefore does not cost CPU, disk or any resource to create. They are useful to find out whether specific indexes can increase the performance for problematic queries, since you can discover if PostgreSQL will use these indexes or not without having to spend resources to create them.

CHAPTER 2

Installation

2.1 Requirements

- PostgreSQL 9.2+

2.2 Packages

Hypopg is available as a package on some GNU/Linux distributions:

- RHEL/Rocky Linux

  HypoPG is available as a package using the PGDG packages.

  Once the PGDG repository is setup, you just need to install the package. As root:

  ```
  yum install hypopg
  ```

- Debian / Ubuntu

  HypoPG is available as a package using the PGDG packages.

  Once the PGDG repository is setup, you just need to install the package. As root:

  ```
  apt install postgresql-XY-hypopg
  ```

  where XY is the major version for which you want to install hypopg.

- Archlinux

  Hypopg is available on the AUR repository.

  If you have **yaourt** setup, you can simply install the *hypopg-git* package with the following command:

  ```
  yaourt -S hypopg-git
  ```

Otherwise, look at the official documentation to manually install the package.

Note: Installing this package will use the current development version. If you want to install a specific version, please see the *Installation from sources* section.

2.3 Installation from sources

To install HypoPG from sources, you need the following extra requirements:

- PostgreSQL development packages

Note: On Debian/Ubuntu systems, the development packages are named *postgresql-server-dev-X*, X being the major version.

On RHEL/Centos systems, the development packages are named *postgresqlX-devel*, X being the major version.

- A C compiler and *make*
- *unzip*
- optionally the *wget* tool
- a user with *sudo* privilege, or a root access

Note: If you don't have *sudo* or if you user isn't authorized to issue command as root, you should do all the following commands as **root**.

First, you need to download HypoPG source code. If you want the development version, you can download it from here, or via command line:

```
wget https://github.com/HypoPG/hypopg/archive/master.zip
```

If you want a specific version, you can chose the version you want here and follow the related download link. For instance, if you want to install the version 1.0.0, you can download it from the command line with the following command:

```
wget https://github.com/HypoPG/hypopg/archive/1.0.0.zip
```

Then, you need to extract the downloaded archive with *unzip* and go to the extracted directory. For instance, if you downloaded the latest development version:

```
unzip master.zip
cd hypopg-master
```

You can now compile and install HypoPG. Simply run:

```
make
sudo make install
```

Note: If you were doing these commands as **root**, you don't need to use sudo. The last command should therefore be:

```
make install
```

If no errors occured, HypoPG is now available! If you need help on how to use it, please refer to the *Usage* section.

CHAPTER 3

Usage

3.1 Introduction

HypoPG is useful if you want to check if some index would help one or multiple queries. Therefore, you should already know what are the queries you need to optimize, and ideas on which indexes you want to try.

Also, the hypothetical indexes that HypoPG will create are not stored in any catalog, but in your connection private memory. Therefore, it won't bloat any table and won't impact any concurrent connection.

Also, since the hypothetical indexes doesn't really exists, HypoPG makes sure they will only be used using a simple EXPLAIN statement (without the ANALYZE option).

3.2 Install the extension

As any other extension, you have to install it on all the databases where you want to be able to use it. This is simply done executing the following query, connected on the database you want to install HypoPG with a user having enough privileges:

```
CREATE EXTENSION hypopg ;
```

HypoPG is now available. You can check easily if the extension is present using psql:

```
\dx
                    List of installed extensions
  Name   | Version |   Schema   |              Description
---------+---------+------------+-------------------------------------
 hypopg  | 1.1.0   | public     | Hypothetical indexes for PostgreSQL
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
(2 rows)
```

As you can see, hypopg version 1.1.0 is installed. If you need to check using plain SQL, please refer to the pg_extension table documentation.

3.3 Configuration

The following configuration parameters (GUCs) are available, and can be changed interactively:

hypopg.enabled: Default to `on`. Use this parameter to globally enable or disable HypoPG. When HypoPG is disabled, no hypothetical index will be used, but the defined hypothetical indexes won't be removed.

hypopg.use_real_oids: Default to `off`. By default, HypoPG won't use "real" object identifiers, but instead borrow ones from the ~ 14000 / 16384 (respectively the lowest unused oid less then FirstNormalObjectId and FirstNormalObjectId) range, which are reserved by PostgreSQL for future usage in future releases. This doesn't cause any problem, as the free range is dynamically computed the first time a connection uses HypoPG, and has the advantage to work on a standby server. But the drawback is that you can't have more than approximately 2500 hypothetical indexes at the same time, and creating a new hypothetical index will become very slow once more than the maximum number of objects has been created until `hypopg_reset()` is called.

If those drawbacks are problematic, you can enable this parameter. HypoPG will then ask for a real object identifier, which will need to obtain more locks and won't work on a standby, but will allow to use the full range of object identifiers.

Note that switching this parameter doesn't require to reset the entries, both can coexist at the same time.

3.4 Supported access methods

The following access methods are supported:

- btree
- brin
- hash (requires PostgreSQL 10 or above)
- bloom (requires the bloom extension to be installed)

3.5 Create a hypothetical index

Note: Using HypoPG require some knowledge on the **EXPLAIN** command. If you need more information about this command, you can check the official documentation. There are also a lot of very good resources available.

For clarity, let's see how it works with a very simple test case:

```
CREATE TABLE hypo (id integer, val text) ;
INSERT INTO hypo SELECT i, 'line ' || i FROM generate_series(1, 100000) i ;
VACUUM ANALYZE hypo ;
```

This table doesn't have any index. Let's assume we want to check if an index would help a simple query. First, let's see how it behaves:

```
EXPLAIN SELECT val FROM hypo WHERE id = 1;
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on hypo  (cost=0.00..1791.00 rows=1 width=14)
   Filter: (id = 1)
(2 rows)
```

A plain sequential scan is used, since no index exists on the table. A simple btree index on the **id** column should help this query. Let's check with HypoPG. The function **hypopg_create_index()** will accept any standard **CREATE INDEX** statement(s) (any other statement passed to this function will be ignored), and create a hypothetical index for each:

```
SELECT * FROM hypopg_create_index('CREATE INDEX ON hypo (id)') ;
 indexrelid |      indexname
------------+----------------------
      18284 | <18284>btree_hypo_id
(1 row)
```

The function returns two columns:

- the object identifier of the hypothetical index
- the generated hypothetical index name

We can run the EXPLAIN again to see if PostgreSQL would use this index:

```
EXPLAIN SELECT val FROM hypo WHERE id = 1;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using <18284>btree_hypo_id on hypo  (cost=0.04..8.06 rows=1 width=10)
   Index Cond: (id = 1)
(2 rows)
```

Yes, PostgreSQL would use such an index. Just to be sure, let's check that the hypothetical index won't be used to acually run the query:

```
EXPLAIN ANALYZE SELECT val FROM hypo WHERE id = 1;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------
↪-------------
 Seq Scan on hypo  (cost=0.00..1791.00 rows=1 width=10) (actual time=0.046..46.390␣
↪rows=1 loops=1)
   Filter: (id = 1)
   Rows Removed by Filter: 99999
 Planning time: 0.160 ms
 Execution time: 46.460 ms
(5 rows)
```

That's all you need to create hypothetical indexes and see if PostgreSQL would use such indexes.

3.6 Manipulate hypothetical indexes

Some other convenience functions and views are available:

- **hypopg_list_indexes**: view that lists all hypothetical indexes that have been created

```
SELECT * FROM hypopg_list_indexes ;
 indexrelid |      index_name       | schema_name | table_name | am_name
------------+-----------------------+-------------+------------+---------
      18284 | <18284>btree_hypo_id  | public      | hypo       | btree
(1 row)
```

- **hypopg**(): function that lists all hypothetical indexes that have been created with the same format as **pg_index**

```
SELECT * FROM hypopg() ;
      indexname       | indexrelid | indrelid | innatts | indisunique | indkey |␣
↪indcollation | indclass | indoption | indexprs | indpred | amid
----------------------+------------+----------+---------+-------------+--------+------
↪--------+----------+-----------+----------+---------+------
 <18284>btree_hypo_id |      13543 |    18122 |       1 | f           | 1      | 0    ␣
↪        | 1978     | <NULL>    | <NULL>   | <NULL>  |  403
(1 row)
```

- **hypopg_get_indexdef(oid)**: function that lists the CREATE INDEX statement that would recreate a stored hypothetical index

```
SELECT index_name, hypopg_get_indexdef(indexrelid) FROM hypopg_list_indexes ;
      index_name       |             hypopg_get_indexdef
-----------------------+-------------------------------------------------
 <18284>btree_hypo_id  | CREATE INDEX ON public.hypo USING btree (id)
(1 row)
```

- **hypopg_relation_size(oid)**: function that estimates how big a hypothetical index would be:

```
SELECT index_name, pg_size_pretty(hypopg_relation_size(indexrelid))
  FROM hypopg_list_indexes ;
      index_name       | pg_size_pretty
-----------------------+----------------
 <18284>btree_hypo_id  | 2544 kB
(1 row)
```

- **hypopg_drop_index(oid)**: function that removes the given hypothetical index
- **hypopg_reset()**: function that removes all hypothetical indexes

3.7 Hypothetically hide existing indexes

You can hide both existing and hypothetical indexes hypothetically. If you want to test it as described in the documentation, you should first use **hypopg_reset()** to clear the effects of any other hypothetical indexes.

As a simple case, let's consider two indexes:

```
SELECT hypopg_reset();
CREATE INDEX ON hypo(id);
CREATE INDEX ON hypo(id, val);
```

```
EXPLAIN SELECT * FROM hypo WHERE id = 1;
                                  QUERY PLAN
-------------------------------------------------------------------------------
Index Only Scan using hypo_id_val_idx on hypo  (cost=0.29..8.30 rows=1 width=13)
Index Cond: (id = 1)
(2 rows)
```

The query plan is using the **hypo_id_val_idx** index now.

- **hypopg_hide_index(oid)**: function that allows you to hide an index in the EXPLAIN output by using its OID. It returns *true* if the index was successfully hidden, and *false* otherwise.

```
SELECT hypopg_hide_index('hypo_id_val_idx'::REGCLASS);
 hypopg_hide_index
-------------------
t
(1 row)

EXPLAIN SELECT * FROM hypo WHERE id = 1;
                              QUERY PLAN
-----------------------------------------------------------------------
Index Scan using hypo_id_idx on hypo  (cost=0.29..8.30 rows=1 width=13)
Index Cond: (id = 1)
(2 rows)
```

As an example, let's assume that the query plan is currently using the **hypo_id_val_idx** index. To continue testing, use the **hypopg_hide_index(oid)** function to hide another index.

```
SELECT hypopg_hide_index('hypo_id_idx'::REGCLASS);
 hypopg_hide_index
-------------------
t
(1 row)

EXPLAIN SELECT * FROM hypo WHERE id = 1;
                     QUERY PLAN
-----------------------------------------------------
Seq Scan on hypo  (cost=0.00..180.00 rows=1 width=13)
Filter: (id = 1)
(2 rows)
```

- **hypopg_unhide_index(oid)**: function that restore a previously hidden index in the EXPLAIN output by using its OID. It returns *true* if the index was successfully restored, and *false* otherwise.

```
SELECT hypopg_unhide_index('hypo_id_idx'::regclass);
 hypopg_unhide_index
---------------------
t
(1 row)

EXPLAIN SELECT * FROM hypo WHERE id = 1;
                              QUERY PLAN
-----------------------------------------------------------------------
Index Scan using hypo_id_idx on hypo  (cost=0.29..8.30 rows=1 width=13)
Index Cond: (id = 1)
(2 rows)
```

- **hypopg_unhide_all_index()**: function that restore all hidden indexes and returns void.
- **hypopg_hidden_indexes()**: function that returns a list of OIDs for all hidden indexes.

```
SELECT * FROM hypopg_hidden_indexes();
 indexid
---------
526604
(1 rows)
```

- **hypopg_hidden_indexes**: view that returns a formatted list of all hidden indexes.

```
SELECT * FROM hypopg_hidden_indexes;
  indexrelid |      index_name       | schema_name | table_name | am_name | is_hypo
-------------+-----------------------+-------------+------------+---------+---------
      526604 | hypo_id_val_idx       | public      | hypo       | btree   | f
(1 rows)
```

Note: Hypothetical indexes can be hidden as well.

```
SELECT hypopg_create_index('CREATE INDEX ON hypo(id)');
    hypopg_create_index
-------------------------------
(12659,<12659>btree_hypo_id)
(1 row)

EXPLAIN SELECT * FROM hypo WHERE id = 1;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
Index Scan using "<12659>btree_hypo_id" on hypo  (cost=0.04..8.05 rows=1 width=13)
Index Cond: (id = 1)
(2 rows)
```

Now that the hypothetical index is being used, we can try hiding it to see the change:

```
SELECT hypopg_hide_index(12659);
 hypopg_hide_index
-------------------
t
(1 row)

EXPLAIN SELECT * FROM hypo WHERE id = 1;
                            QUERY PLAN
------------------------------------------------------------------------
Index Scan using hypo_id_idx on hypo  (cost=0.29..8.30 rows=1 width=13)
Index Cond: (id = 1)
(2 rows)

SELECT * FROM hypopg_hidden_indexes;
  indexrelid |      index_name       | schema_name | table_name | am_name | is_hypo
-------------+-----------------------+-------------+------------+---------+---------
       12659 | <12659>btree_hypo_id  | public      | hypo       | btree   | t
      526604 | hypo_id_val_idx       | public      | hypo       | btree   | f
(2 rows)
```

Note: If a hypothetical index has been hidden, it will be automatically unhidden when it is deleted using **hypopg_drop_index(oid)** or **hypopg_reset()**.

```
SELECT hypopg_drop_index(12659);

SELECT * FROM hypopg_hidden_indexes;
  indexrelid |      index_name       | schema_name | table_name | am_name | is_hypo
-------------+-----------------------+-------------+------------+---------+---------
      526604 | hypo_id_val_idx       | public      | hypo       | btree   | f
(2 rows)
```

CHAPTER 4

Contributing

HypoPG is an open source project, distributed under the PostgreSQL licence.

4.1 Talk

If you have suggestions, feature request or just want to say hi you can join the **#hypopg** IRC channel on freenode.

4.2 Bug reports

If you've found a bug, please report it on the HypoPG bug-tracker on Github.

4.3 Hacking

If you want to fix a bug, enhance the documentation or develop new features, feel free to clone the git repository on Github.