HighGo Postgres Server (HG-PGSQL) Release Notes

Version 1.1

December, 2019

HighGo Software Inc. ™
10318 Whalley Blvd #1 Surrey, BC, V3T 4H4, Canada
T: +1 (604) 781-6749 E: contact@highgo.ca W: www.highgo.ca

Table of Contents

1 Introduction

HighGo Software Canada (a subsidiary of HighGo Software Inc.) is pleased to announce HighGo Postgres Server 1.1 (HG-PGSQL 1.1). With this release, HighGo Software continues to innovate and build on top of open source based database solution (PostgreSQL 12.1) with aims to deliver a solution with high performance, reliability and ease of use for small, medium and large-scale enterprises.

PostgreSQL is an advanced and powerful open source object relational database with over 30 years of active development. HG-PGSQL 1.1 is the first release of HighGo Postgres Sever, which aims to provide additional functionality and value to its users. It is built on top of already robust, reliable and performant open source PostgreSQL 12.1 base. The focus of the first release is to enhance backup performance, maintain the ease of use and improve partitioning for better scalability. The goal of the HighGo team is to continuously enhance the product and provide tremendous value to our end-users in variety of areas.

PostgreSQL 12.1 is packed with many exciting new features and improvements. Partitioning functionality has been improved by introducing fast partition pruning along with additional partitioning improvements. There have been several general performance improvements as well, such as optimizations for b-tree and other indexes as well as automatic inlining of common table expression (CTE). New authentication features have been added as well including encryption of TCP/IP connections during GSSAPI authentication, support for SQL/JSON path language and several other exciting features.

HighGo Postgres Server 1.1 is built on top of PostgreSQL 12.1 and provides the following outstanding features:

- **Partitioning Enhancement**

Enhance the table partitioning syntax to allow specifying the parent table and partitions in the same SQL statement. This is an ease of use feature that will take away the hassle of creating partitions separately from the parent table. This allows the end-user to specify the partitions and sub-partitions using the native PostgreSQL syntax in the same statement as the parent table.

- **Shard Management**

Provide the capability of creating a sharded table by specifying parent table and foreign partitions in the same statement. This feature adds the ability to push-down partitions to the foreign server as part of the horizontal scaling / sharding effort. The user can specify the partitions along with the foreign server (shards) and use the push-down clause to automatically create partitions on the foreign servers. This is an exciting feature that automates the creation of partitions on the shards. This capability is added to **postgres_fdw** in this release and it can be extended to other databases.

- **Parallel Backup**

Greatly improve the performance of full base backup by adding parallelism to the backup operation. This feature adds parallelism to base backup by spawning a configurable number of parallel workers to perform the backup. This feature enables the user to use more system resources by spawning multiple workers to perform backup and greatly reduce to time required.

Please visit the link below to download HighGo Postgres Server 1.1:

https://yum.highgo.ca/

2 Installation and Documentation

Highgo Postgres Server v1.1 supports RPM installation on the following platforms

- CentOS (x86_64) 6.x
- CentOS (x86_64) 7.x

Installation and quick start documentation is available on Highgo Software website. Visit:

https://www.highgo.ca/products/highgo-postgresql-server

3 HighGo Postgres Server Main Features

HighGo Postgres Sever 1.1 comes with the following outstanding features that are unique to HG-PGSQL and includes the integration of all the features that are part of PostgreSQL 12.1 release.

This section includes a brief overview of the new features included in the HG-PGSQL release. Please refer to comprehensive user guide in the link below for more detailed information on the features:

https://www.highgo.ca/products/highgo-postgresql-server

3.1 Partitioning Enhancements

HG-PGSQL 1.1 enhances the table partitioning syntax by allowing the user to specify parent table, partitions and sub-partitions in the same SQL statement. This takes away the hassle of creating parent table, partitions and their associated artifacts separately.

This feature is particularly useful in real world scenarios where a user is creating complex partitions and sub-partitions. The CREATE TABLE clause example below shows that it is very easy to create a complex partition table having partitions and sub-partitions declared in a single SQL statement using the new syntax

```
CREATE TABLE table_name (..)
PARTITION BY { RANGE | LIST | HASH } (..)
(
    list of partitions
);
```

The new syntax supports all the partition types. It is important to note that while creating partitions and sub-partitions, the user is supposed to use the same PG syntax.

The Following example of creating a list partition table shows that PG syntax is used while specifying the partitions.

```
CREATE TABLE region_sales
(
    item_number INT,
    store_name VARCHAR(30),
    region_code VARCHAR(2),
    date date
)
PARTITION BY LIST (region_code)
(
    PARTITION p_florida for VALUES in (10),
    PARTITION p_hawaii for VALUES in (20),
    PARTITION p_others DEFAULT
);
```

3.2 Shard Management

Building on the horizontal scalability effort, HG-PGSQL 1.1 supports the creation of a sharded table where the partitions or sub-partitions are automatically created on foreign servers (i.e. shards). The community has been putting effort in horizontal scalability and added number of important features like join pushdown, aggregate pushdown, sort pushdown … etc in last few releases of PostgreSQL.

The shard management feature of HG-PGSQL is part of the same effort to allow automatic creation of partitions on foreign servers (i.e. shards), without which the user currently has to manually create the partitions on the shards, which is very cumbersome and error-prone task

```
CREATE TABLE partitioned_tbl(a int)
    PARTITION BY RANGE(a)
    (
        PARTITION
            public.pushed_down_part FOR VALUES FROM (1) TO (10)
            TABLESPACE test_space
            SERVER shard_server WITH PUSHDOWN,
        PARTITION
            public.not_pushed_down_part FOR VALUES FROM (10) TO (20)
            SERVER shard_server,
        PARTITION public.local_part DEFAULT
    );
```

The above example of creating a sharded table shows the syntax of partition creation on the foreign server (i.e. shards).

Please note the line "`SERVER shard_server WITH PUSHDOWN`" containing the clause `<WITH PUSHDOWN>` indicates that the partition is to be automatically created in the foreign server called "shard_server". Without the `<WITH PUSHDOWN>` clause, the partition will not be created on the foreign server.

This feature is only implemented in **postgres_fdw** extension in HG-PGSQL 1.1 and it can be extended to other FDW's to create partitions on other foreign servers. This is a useful feature designed to greatly simply the creation of sharded tables.

HG-PGSQL also supports dropping foreign tables right from the local database server, by using the new `<INCLUDE REMOTE>` clause. The example below will drop both the local and the foreign table named "`tft1_f`"

```
$ DROP FOREIGN TABLE tf1_f INCLUDE REMOTE;

NOTICE: Drop remote table 'public.tf1_f' on foreign server 'S1'
DROP FOREIGN TABLE
```

3.2.1 Restrictions

Constaints are not Created at Foreign Server:
The end-user can specify constraints for partitions using existing PG syntax as shown below. The constraints will get created on the parent node and won't get pushdown to the remote servers. This behavior is intended and as a workaround, user can create the constraints on the remote servers manually.

```
CREATE TABLE partitioned_tbl(a int)
    PARTITION BY RANGE(a)
    (
        PARTITION
            public.pushed_down_part
            (
                CONSTRAINT p_con1 CHECK (a < 10)
            ) FOR VALUES FROM (1) TO (10)
            TABLESPACE test_space
            SERVER shard_server WITH PUSHDOWN,
        PARTITION
            public.not_pushed_down_part
            (
                CONSTRAINT p_con2 CHECK (a > 10)
            ) FOR VALUES FROM (10) TO (20)
            SERVER shard_server,
        PARTITION public.local_part DEFAULT
    );
```

It is possible that some of these restrictions are handled in later HG-PSQL release.

3.3 Parallel Backup

HG-PGSQL 1.1 is packed with a much-anticipated performance upgrade by introducing parallelism to full database backup operation. The process of taking a full backup of a large database can be very time consuming and can potentially slow down the online database operations. The parallel backup feature adds parallelism to backup operation by spawning multiple worker processes that will perform the required backup operation in parallel and thus reduce the time it would need.

This feature is implemented in **pg_basebackup** front end tool and the number of parallel workers can be specified as command line arguments as **pg_basebackup** is executed by the user. The desired number of workers depends on the size of the target database and system specification such as "limits" on unix-based systems. The user should carefully evaluate the system and select an appropriate number of parallel workers such that the backup process is optimized while not affecting other aspects of the system in terms of resource allocation.

```
$ pg_basebackup --help
 pg_basebackup takes a base backup of a running PostgreSQL server.

 Usage:
 pg_basebackup [OPTION]…
```

```
Options controlling the output:
-j, --jobs=NUM         use this many parallel jobs to backup
```

The above shows that --jobs (-j) argument can be used to set the number of parallel workers to do the backup operation in parallel. Both **pg_basebackup** and its spawned workers perform backup operation in file level and the backup files are divided across the workers during parallel operation.

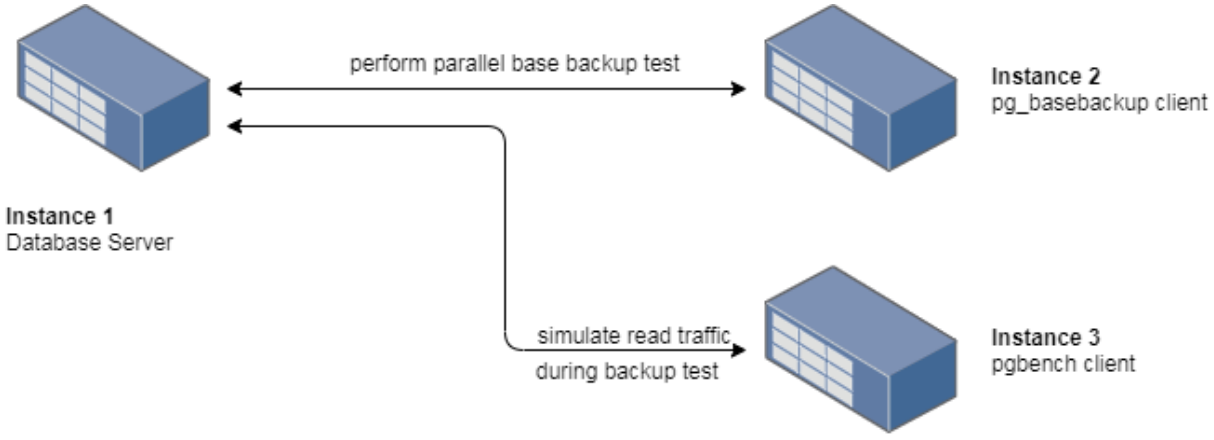
3.3.1 Parallel Backup Benchmark

The Parallel full backup feature of HG-PGSQL 1.1 was carried out in order to demonstrate the performance gain in comparison to base backup. The benchmarking was done using instances on AWS cloud (specification shown below) and both base backup and parallel backup tests were done on the same environment and configuration in order to perform an apple to apple comparison. The test was done against a moderate sized database and each test was repeated 3 times to take a median from the three results.

The benchmarking script performs full base backup 3 times and performs parallel backup 3 times on each different number of threads in order to observe the trend as we increase the number of threads. The results for the full base backup and parallel backup are shown below.

AWS Instances
3 AWS instances having instance type t2.xlarge were used in this benchmarking exercise. As shown in the diagram below, instance 1 was running the database server, instance 2 was running the **pg_basebackup** client and instance 3 was running the **pgbench** client that is generating read load while the backup is in progress.

All instances are using AWS volume type io1, which provides about 16000 maximum IOP rating to get better throughput from the storage devices. The goal of parallel backup is to saturate the network and get maximum processing out of the network and disk I/O by running parallel threads performing the backup operations. The parallel threads are running while the database server is under load from several clients competing for resources to mimic a real-world scenario.



Benchmark Results

The detailed parallel backup benchmark report and results can be found in the white paper below published by the HighGo Team:

https://www.highgo.ca/products/highgo-postgresql-server

4 Report a Problem

To report any issue you are having, please contact HighGo Software technical support staff:

Email: contact@highgo.ca
Phone: +1 (604) 781-6749