PGXC_CTL Primer

Configuring and operating Postgres-XC database cluster

May 7th, 2014

Koichi Suzuki

Change History:

May 7th, 2014: Initial version.
Oct 2nd, 2014: Added license condition.

1. Introduction

This document is an outline of Postgres-XC database cluster configuration and operation using pgxc_ctl. Pgxc_ctl will be found in the contrib module of postgres-xc distribution. This module helps to configure and operate your Postgres-XC database cluster easily. This saves much of the small but important things.

2. Plannings you Postgres-XC cluster

2.1 Postgres-XC overview

Before going into Postgres-XC configuration, please take a bit to consider if Postgres-XC cluster is useful to your application. Unlike any other PostgreSQL replication cluster, Postgres-XC is write-scalable PostgreSQL cluster. More servers you have, more throughput you get, both in read and write. Of course, you can handle bigger databsae if you have more servers and storages.

To achieve this, you should consider each table to be distributed (sharded) or replicated. If tables are updated very frequently, you should consider them to be distributed. If they are relatively static and are refered by distributed table or other replicated tables, you should consider them to be replicated. This mixture will make most of the update local to one of the datanode. Because each datanode operates in paralle, you can get both read and write scalability.

You can connect to any of the coordinators[1], each of which provides full transaction ACID capabiliy and transparent view to the database. That is, any update you make at any coordinator is visible to any other applications connected to any coordinators without delay. It behaves just like single database.

From your application, Postgres-XC cluster will be seen as shown in Figure 1.

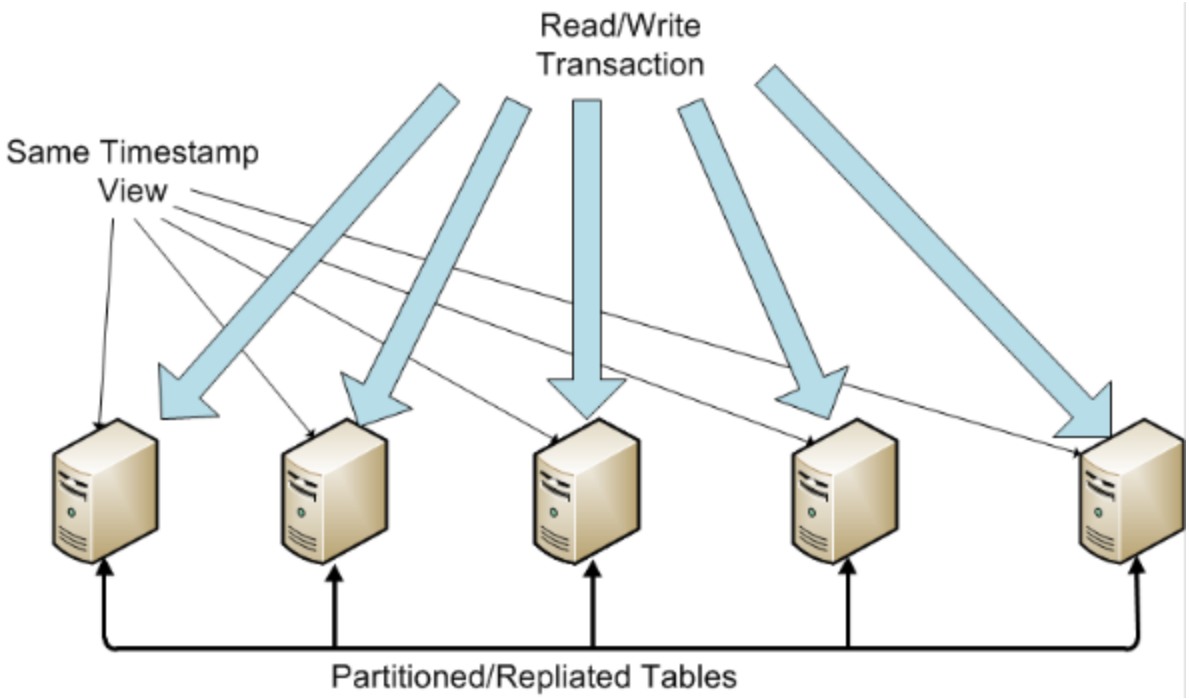

Figure1. How Postgres-XC is seen from your application.

While conventional PostgresSQL hot-standby configuration will be seen as in Figure 2.

[1] Coordinator is a component your application should connect to. XC has several other component such as datanode, GTM and GTM proxy. They will be described some more in datail in the next section.

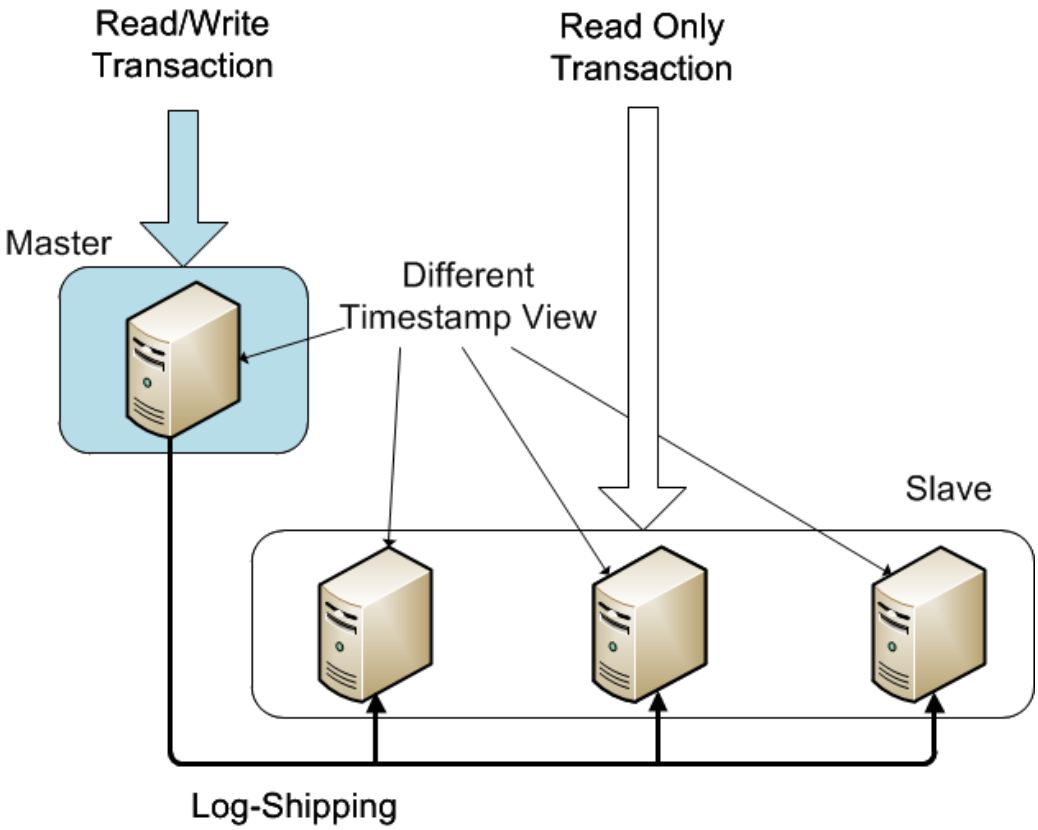

Figure 2. PostgreSQL Hot Standby

To provide these features, Postgres-XC assumes all the (master) nodes should be running normally while Postgres-XC runs. In other words, this architecture itself does not provide HA capability. To incorporate HA capability, you should consider to configure a slave for some of Postgres-XC components. This will be described in a later sections.

So far, Postgres-XC planner is best-tuned for OLTP applications. For data-warehouse applications, you may need separate patch which devides complexed query into smaller chunks which run in datanodes in parallel. StormDB developed such patch and is now is available from Translattice, www.translattice.com.

Postgres-XC is based upon shared-nothing architecture. You don't need any dedicated hardware to run it. You just need commodity INTEL server runnign Linux.

If you are sure that Postgres-XC helps your application, this is the time you should learn more about Postgres-XC. First of all, let's leary structure of Postgres-XC database cluster.

2.2 Suported Platform

As mentioned above. Postgres-XC runs only on INTEL 64bit Linux at present.
Postgres-XC community welcomes any effort to run it on other platforms.

2.3 Postgres-XC component

Overview of Postgres-XC component is shown in Figure 3.

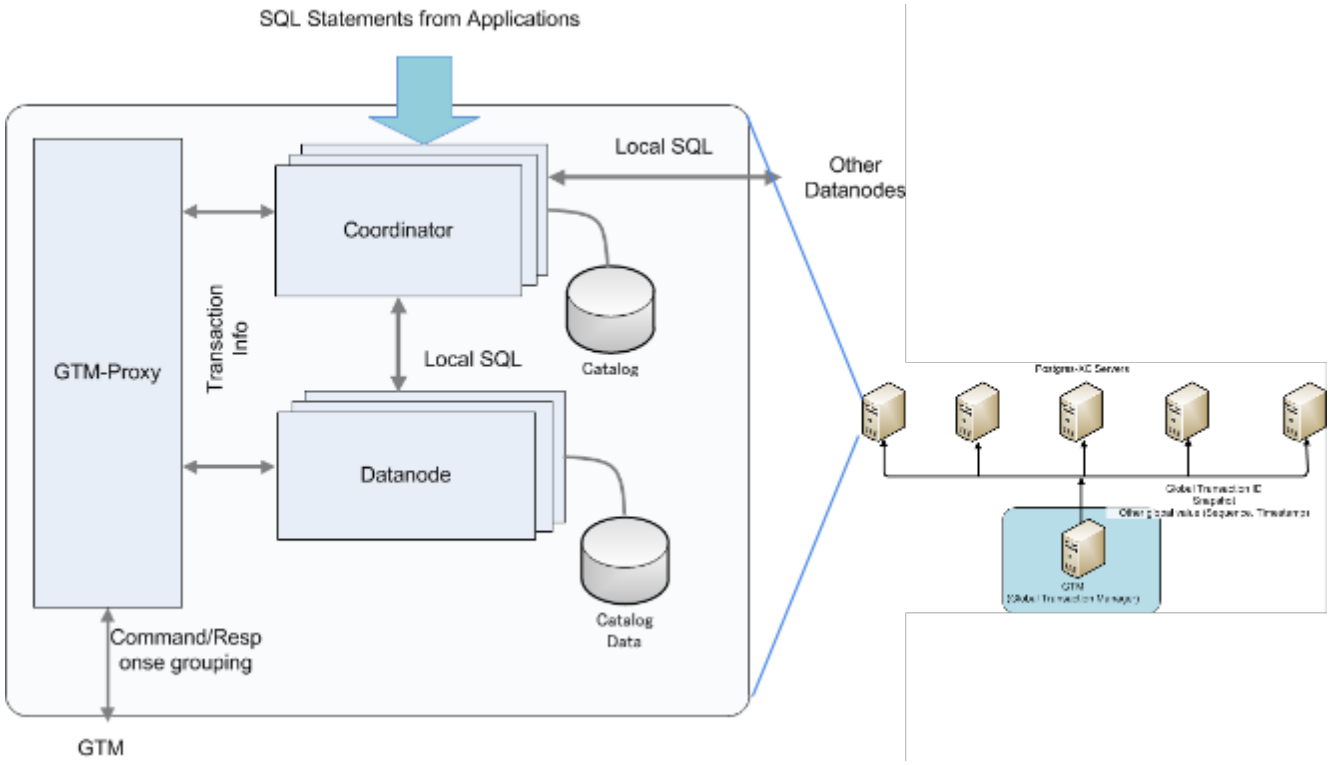

Figure 3. Postgres-XC component overview.

So far, Postgers-XC is composed of the following components.

1) Coordinator
   Connection point from your application. This component receives all the SQL statement from applications, make and optimize its execution plan by dividing it into smaller chunks if needed, ship them to datanodes as described next and combine all the result from them to return to applications.

2) Datanode
   Data storage. Each table data, distributed or replicated, will go to one or more of datanodes. This runs just like single PostgreSQL to execute local SQL statment shipped from coordinators.

3) GTM
   Provides distributed transaction control over all the coordinators and datanodes. In short, this is an implementation of distributed MVCC[2], distributed version of PostgreSQL's MVCC. This component also provides sequence.

[2] Multi-Version Concurrency Control.

4) GTM Proxy
   You may not configure this component. GTM Proxy groups up commands from coordinators and/or datanodes to save amount of network communication to GTM.

In document, each component may be called as “node”.

2.4 Whether distribute or repslicate tables?

This section describes how to design your database from Postgres-XC point of view. As described in earlier section, Postgres-XC achieves read and write scalability bo the combination of table distribution (sharding) and replication.

More stable tables should reside in all (or more than one) datanodes so that join operation with distributed table can be done locally at datanodes.

Frequently updated tables should be distributed (other words, sharded) among more than one datanodes so that updates can be performed in parallel. You should choose what column to use as a key to determine the location of each row (distribution key or distribution column). Distribution key could be its primary key. However, you should also be careful to use distribution key as join key as much as possible. For this purpose, you may want to add artificial column.

Following section describes how to determine what tables to distribute or replicated and what distribution key to choose in some of database benchmarks which will be helpful to your applications.

DBT-1

DBT-1 is a short-transaction application benchmark based on an online bookstore application. Tables can be grouped into following five groups:

1. Customer, including address, order and payment
2. Shopping cart. Some may say shopping cart can be grouped to the customer. Because the application may start to use shopping cart without customer assigned, shopping cart tables should be a separate group.
3. Book inventory.
4. Book list.
5. Author.

We separated book inventory from the book list because the former is updated very frequently and the latter is relatively stable.

Obviously, book list and author are relatively stable. While order/payment, shopping cart and book inventry can be updated thousands of times per second, the book list and author may be updated when new book is brought to the shop, may be onece or twice a day. In the shopping, customers refer to book/author information very frequently, in other words, they tend to be joined with first three groups of tables very frequently. Therefore, it is quite reasonable to define the first three groups of tables as distributed tables and to

define the rest of the table as replicated. In this way, most of the joins can be performed locally in a datanode where distributed table row is located.

This is very important step to determine your table distribution design. It is ideal if you application's joins are done with a same key. Because it does not happen often, you should be careful to choose what key to use to distribute tables. In DBT-1, shopping cart tables has to join with customer tables, such join has to be done across different datanodes (cross-node join or cross-node operation). Although this is very hard to avoid completely, you can minimize this chance.

Figure 4 shows table distribution design based on the above considerations.

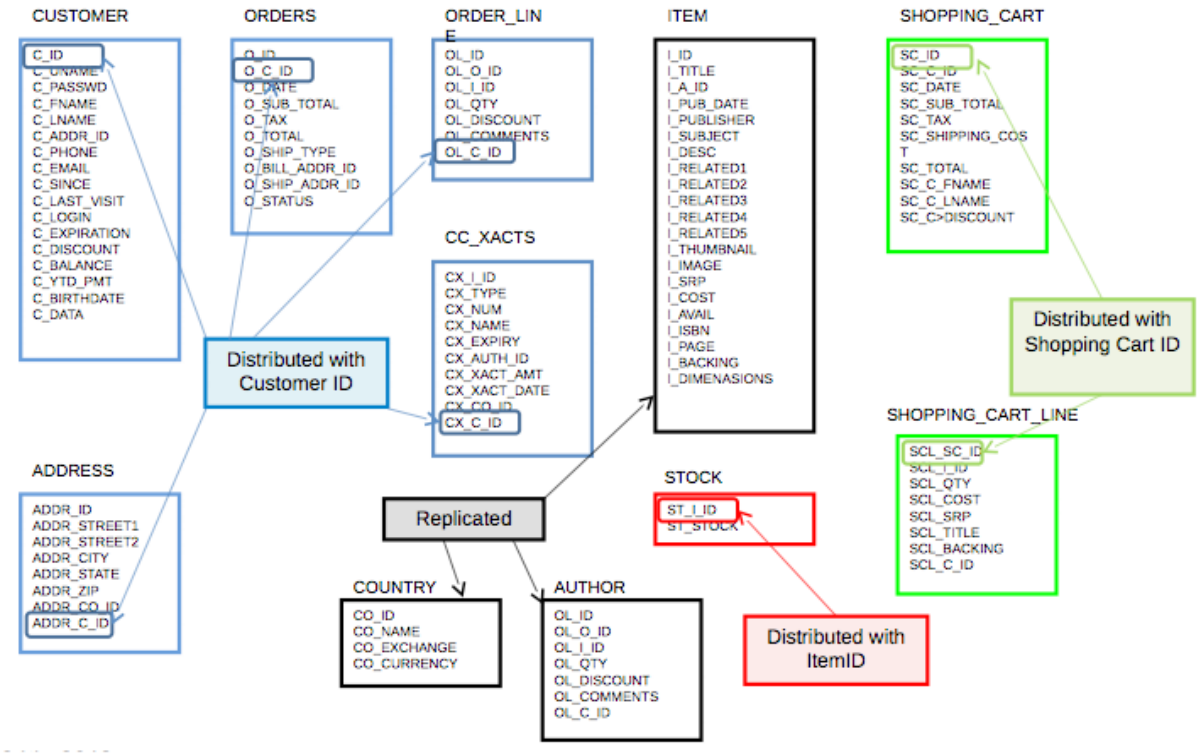

Figure 4. DBT-2 Tables Distribution Design.

2.5 Number of servers, master and slave

You should carefully plan how many servers to use to configure your Postgres-XC cluster. The minimum number of each components and recommended configuration is as follows:

GTM

You need one GTM at least. GTM is a crital component to run Postgres-XC cluster. It provideds cluster-wide transaction management. It also provides sequence feature. In this way, each coordinator/datanode can run in parallel maintaining global database consistency among the components.

You should consider to configure GTM in a separate server. Because of the nature of its feature, network workload of GTM is relatively high. On the other hand, its footprint to storage, CPU, and memory is relatively small. You may consider to use lower performance server for this.

If you are considering to configure HA feature, you should configure GTM slave as well. GTM slave's workload is similar to GTM and you may want to configure GTM slave in a separate server too.

GTM Proxy

GTM Proxy is not the MUST in Postgres-XC. It groups up requests and response between each transaction and GTM and reduces GTM network workload. If your application looks busy, it will be a good choice to configure GTM proxy. If GTM Proxy takes care of transaction running locally, interaction between GTM Proxy and transactions will be made locally, without NIC and it is also a good choice to configure GTM Proxy in all the servers where coordinators and/or datanodes are running.

Because GTM Proxy does not store any data, it does not need HA configuration. If something is wrong, you can simply configure another GTM Proxy, reconfigure coordinator/datanode and start it.

Coordinator and Datanode

In principle, Postgres-XC configuration is quite flexible. You can configure any number of coordinators and datanodes. If you configure different number of coordinator and datanode and configure them in different servers, you may have to be carefull about worload balance. Just for simplicity, we recommend to start with configuring both coordinator and datanode in a server, as well as GTM proxy.

You should have explicit reason not to do so.

Because Postgres-XC does not provide HA feature by its own, you need to configure slaves of coordinator and datanode to build HA capability. Simple configuration of coordinator/datanode slave is to configure it in a server where another coordinator/data master is configured. Figure 5 and Figure 6 illustrate such examples.

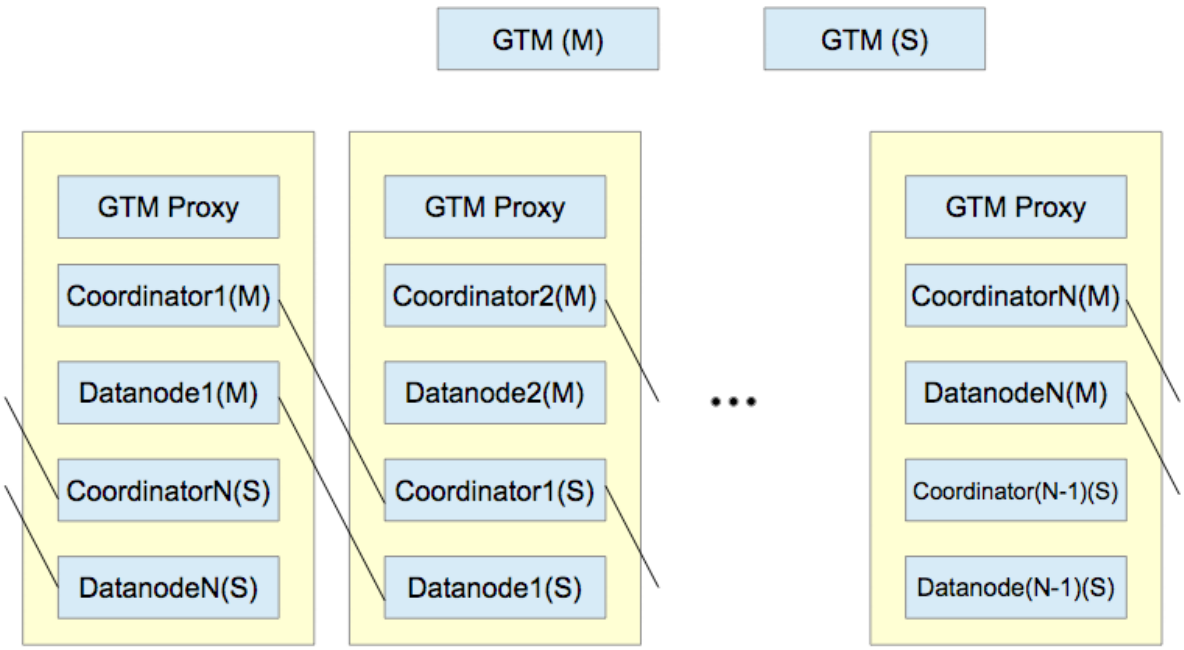

Figure 5. Postgres-XC HA configuration example (circular configuration)

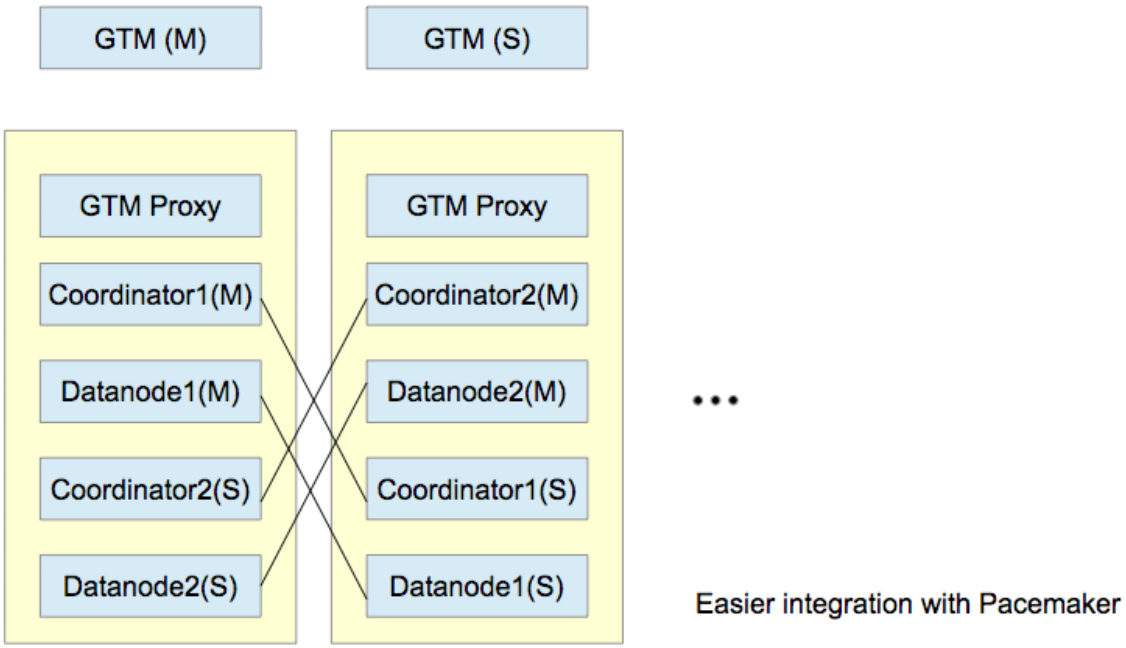

Figure 6. Postgres-XC HA configuraiton example (pair configuration)

2.6 Configuration overview

As mentioned in the reference manual (latest reference manuall will be found at Postgres-XC document page, you can configure each component manually. Configuring sigle PostgreSQL database takes much effort and you have to be very careful. Postgres-XC, in nature, is much much more.

To avoid pitfalls you may encounter, Postgres-XC provides dedicated configuration and operation tool called pgxc_ctl[3]. The rest of this paper will be focused of Postgres-XC cluster confiugration and operation using this tool. Each internal steps will be provided in each section so that you can try manual configuration and operatoin.

[3] If you find any issues and bugs with pgxc_ctl, please report it to Postgres-XC general or bugs mailing list. Mailing list will be available at Mailing List tab in https://sourceforge.net/projects/postgres-xc/

3. Building Postgres-XC binary and pgxc_ctl tool

You can obtain Postgres-XC source tarball from its download site or you can clone Postgres-XC GIT repository to obtain its source code.

Pgxc_ctl source code comes with Postgres-XC release and Postgres-XC git repository. It is placed at contrib/pgxc_ctl directory. Whole directory looks like

```
[koichi@buildfarm:postgres-xc]$ ls -F
COPYRIGHT    HISTORY    README.git  config.log      configure.in  doc-xc/
GNUmakefile  Makefile   aclocal.m4  config.status*  contrib/       src/
GNUmakefile.in     README   config/     configure*      doc/
[koichi@buildfarm:postgres-xc]$ ls -F contrib
Makefile            file_fdw/               pg_test_fsync/         sepgsql/
README              fuzzystrmatch/          pg_test_timing/        spi/
adminpack/          hstore/                 pg_trgm/               sslinfo/
auth_delay/         intagg/                 pg_upgrade/            start-scripts/
auto_explain/       intarray/               pg_upgrade_support/    tablefunc/
btree_gin/          isn/                    pg_xlogdump/           tcn/
btree_gist/         lo/                     pgbench/               test_parser/
chkpass/            ltree/                  pgcrypto/              tsearch2/
citext/             oid2name/               pgrowlocks/            unaccent/
contrib-global.mk   pageinspect/            pgstattuple/           uuid-ossp/
cube/               passwordcheck/          pgxc_clean/            vacuumlo/
dblink/             pg_archivecleanup/      pgxc_ctl/              worker_spi/
dict_int/           pg_buffercache/         pgxc_ddl/              xml2/
dict_xsyn/          pg_freespacemap/        pgxc_monitor/
dummy_seclabel/     pg_standby/             postgres_fdw/
earthdistance/      pg_stat_statements/     seg/
[koichi@buildfarm:postgres-xc]$
```

First of all, you should run configure tool at the top level of Postgres-XC source tree. You may want to specify installation point and debug option as follows.

```
[koichi@buildfarm:postgres-xc]$ ./configure --prefix=/home/koichi/pgxc --enable-debug
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking which template to use... linux
checking whether to build with 64-bit integer date/time support... yes
checking whether NLS is wanted... no
checking for default port number... 5432
checking for block size... 8kB
checking for segment size... 1GB
checking for WAL block size... 8kB
checking for WAL segment size... 16MB

… (omitted) …

[koichi@buildfarm:postgres-xc]$
```

You can use all the configure options as you find in PostgreSQL. Here's a couple of issues you should notice.

1) Postgres-XC specific source line is distinguished by the simbol "PGXC" in C source code and its header files. Configure propagates this to each Makefile.

2) If you use CFLAGS option to specify options in gcc, optimization will be omitted. The default optimation option is -O2. If you specify other options in CFLAGS and would like to keep using -O2 option, please specify it explicitly.

Then, you can build Postgres-XC binary as follows:

```
[koichi@buildfarm:postgres-xc]$ make -j 8
make -C src all

… (omitted) …

All of PostgreSQL successfully made. Ready to install.
[koichi@buildfarm:postgres-xc]$
```

And then install.

```
[koichi@buildfarm:postgres-xc]$ make install
make -C src install

...

PostgreSQL installation complete.
[koichi@buildfarm:postgres-xc]$
```

Each contrib module will not be build automatically in this step. You should build them using separately. In this paper, you need pgxc_ctl and pgxc_monitor. Pgxc_monitor tells you if specified component is running or not.

```
[koichi@buildfarm:postgres-xc]$ cd contrib/pgxc_ctl
[koichi@buildfarm:pgxc_ctl]$ make
gcc -DPGXC -O2 …
… (omitted) …
[koichi@buildfarm:pgxc_ctl]$ make install
/bin/mkdir -p '/home/koichi/pgxc/bin'
/usr/bin/install -c  pgxc_ctl '/home/koichi/pgxc/bin'
[koichi@buildfarm:pgxc_ctl]$ cd ../pgxc_monitor
[koichi@buildfarm:pgxc_monitor]$ make
gcc -DPGXC -O2 -Wall …
… (omitted) …
[koichi@buildfarm:pgxc_monitor]$ make install
/bin/mkdir -p '/home/koichi/pgxc/bin'
/usr/bin/install -c  pgxc_monitor '/home/koichi/pgxc/bin'
[koichi@buildfarm:pgxc_monitor]$
```

4. What is pgxc_ctl

Pgxc_ctl is a command line tool to help your Postgres-XC cluster configuration, operation and management. Before you rul pgxc_ctl with your configuration, you should prepare pgxc_ctl resources, which will be described in the next section.

Pgxc_ctl prompts you to type its command. If typed like is not pgxc_ctl command, it just passes the line to your shell. Because of this, pgxc_ctl does not provide full shell capability such as variables.

You can specify one pgxc_ctl command as pgxc_ctl command arguments. In this case, pgxc_ctl will run specified command and exits. Examples will be shown later.

5. Writing your cluster configuration

5.1 Overview of pgxc_ctl configuration file and environment

Pgxc_ctl configuration file is in fact a bash shell script. That is, you can write any bash script which helps you to define your postgres-XC configuration. In later sections, you will find many of such examples.

Default name of the configuration file is `pgxc_ctl.conf`. You can specify other configuration file with `-c` option to pgxc_ctl command. The path is absolute of relative to pgxc_ctl directory as described in the next paragraph.

Pgxc_ctl assumes dedicated directory to store its log and other materials. The default directory is `$HOME/pgxc_ctl`. You can change this by specifiying `--home` option when you start pgxc_ctl.

Pgxc_ctl has some more options to cotrol its behavior such as log level and verbosity. You can specify this in `.pgxc_ctl` file placed in your home directory. Each line specifies option and its value such as

```
[koichi@buildfarm:~]$ cat .pgxc_ctl
xc_prompt 'PGXC$ '
#verbose y
#logMessage 'DEBUG3'
#printMessage 'DEBUG1'
#printLocation y
#logLocation y
#debug y
[koichi@buildfarm:~]$
```

`xc_prompt` is pgxc_ctl promt in a string (does not support serial number or other fancy staff as in bash). Verbose is `y` or `n`. `logMessage` is the level of the message goes to the log. You can specify …… . `printMessage` is the level of the message goes to the terminal you're running pgxc_ctl. `printLocation` is for debug to print location of pgxc_ctl source code with messages. Usually specify `n`. `Debug` also prints some more message for debugging. Usually, specify `n`.

This file is optional. All the default values will be taken if no environment file is found.

Pgxc_ctl log will be printed to the directory `pgxc_log` under pgxc_ctl directory unless you specify this explicity with `-L` option of pgxc_ctl.

5.2 Get configuration file template

First of all, you may need configuration file template to begin with. First you don't have pgxc_ctl directory. In this case run pgxc_ctl from your home directory like this.

```
[koichi@node01:~]$ pgxc_ctl prepare
Installing pgxc_ctl_bash script as /home/koichi/pgxc_ctl/pgxc_ctl_bash.
ERROR: File "/home/koichi/pgxc_ctl/pgxc_ctl.conf" not found or not a regular file.
No such file or directory
Installing pgxc_ctl_bash script as /home/koichi/pgxc_ctl/pgxc_ctl_bash.
Reading configuration using /home/koichi/pgxc_ctl/pgxc_ctl_bash --home
/home/koichi/pgxc_ctl --configuration /home/koichi/pgxc_ctl/pgxc_ctl.conf
Finished to read configuration.
   ******** PGXC_CTL START ***************

Current directory: /home/koichi/pgxc_ctl
[koichi@node01:~]$ ls pgxc_ctl
coordExtraConfig  pgxc_ctl.conf  pgxc_log/
[koichi@node01:~]$
```

You can specify pgxc_ctl command in pgxc_ctl command line. With several messages, your pgxc_ctl directory and configuration file are build.

You can specify configuration file name to build as:

```
[koichi@node01:~]$ pgxc_ctl prepare my_pgxc_ctl.conf
Installing pgxc_ctl_bash script as /home/koichi/pgxc_ctl/pgxc_ctl_bash.
ERROR: File "/home/koichi/pgxc_ctl/pgxc_ctl.conf" not found or not a regular file.
No such file or directory
Installing pgxc_ctl_bash script as /home/koichi/pgxc_ctl/pgxc_ctl_bash.
Reading configuration using /home/koichi/pgxc_ctl/pgxc_ctl_bash --home
/home/koichi/pgxc_ctl --configuration /home/koichi/pgxc_ctl/pgxc_ctl.conf
Finished to read configuration.
   ******** PGXC_CTL START ***************

Current directory: /home/koichi/pgxc_ctl
[koichi@node01:~]$ ls pgxc_ctl
coordExtraConfig  my_pgxc_ctl.conf  pgxc_log/
[koichi@node01:~]$
```

Please note that you don't have to make pgxc_ctl directory. If not found, pgxc_ctl will make this directory when it runs.

Later on, we use `$HOME/pgxc_ctl` and pgxc_ctl directory and `pgxc_ctl.conf` as configuration file, both the default.

5.3 How configuration file looks like

The next figure shows the outline of pgxc_ctl configuration file. Details of each portion will be described later, section by section. Again, because the configuration file is bash script, you can use bash capability to specify specific configuration. You will see how template configuration uses this.

```
#!/bin/bash
#
# Postgres-XC Configuration file for pgxc_ctl utility.
#
# Configuration file can be specified as -c option from pgxc_ctl command.   Default is
# $PGXC_CTL_HOME/pgxc_ctl.org.
#
# This is bash script so you can make any addition for your convenience to configure
# your Postgres-XC cluster.
#
# Please understand that pgxc_ctl provides only a subset of configuration which pgxc_ctl
# provide.  Here's several several assumptions/restrictions pgxc_ctl depends on.
#
...(omitted)...
# 8) Killing nodes may end up with IPC resource leak, such as semafor and shared memory.
#    Only listening port (socket) will be cleaned with clean command.
#
# 9) Backup and restore are not supported in pgxc_ctl at present.   This is a big task and
#    may need considerable resource.
#
#========================================================================================
#
#
# pgxcInstallDir variable is needed if you invoke "deploy" command from pgxc_ctl utility.
# If don't you don't need this variable.
pgxcInstallDir=$HOME/pgxc
#---- OVERALL -----------------------------------------------------------------------
#
pgxcOwner=koichi                # owner of the Postgres-XC databaseo cluster.  Here, we use this
                                # both as linus user and database user.  This must be
                                # the super user of each coordinator and datanode.
```

First lines are comments for the general description how the configuration file is composed. You may want to read this a bit carefully to avoid problems and pitfalls.

The configuration file's goal is to specify values of pre-defined variables.

5.4 Common configuration section

You will see common configuration section at the top. In this section, you define the directory where your Postgres-XC binaries are installes, and the set of servers where you're configurating Postgres-XC cluster.
The section looks like:

```
# pgxcInstallDir variable is needed if you invoke "deploy" command from pgxc_ctl utility.
# If don't you don't need this variable.
pgxcInstallDir=$HOME/pgxc
#---- OVERALL -----------------------------------------------------------------------------
#
pgxcOwner=koichi        # owner of the Postgres-XC databaseo cluster.  Here, we use this
                        # both as linus user and database user.  This must be
                        # the super user of each coordinator and datanode.
pgxcUser=$pgxcOwner     # OS user of Postgres-XC owner

tmpDir=/tmp                 # temporary dir used in XC servers
localTmpDir=$tmpDir         # temporary dir used here locally

configBackup=n                  # If you want config file backup, specify y to this value.
configBackupHost=pgxc-linker    # host to backup config file
configBackupDir=$HOME/pgxc      # Backup directory
configBackupFile=pgxc_ctl.bak   # Backup file name --> Need to synchronize when original changed.
```

pgxcInstallDir variable

First, you will see the variable `pgxcInstallDir`. This is the directory Postgres-XC binaries are installed locally. This value is the `--prefix` option value of `configure` utility used to build Postgres-XC binary from the source code. If you run `make` and `make install`, by specifying `--prefix` option as `$pgxcInstallDir` value, you will have `$pgxcInstallDir` like this:

```
[koichi@buildfarm:pgxc]$ pwd
/home/koichi/pgxc
[koichi@buildfarm:pgxc]$ ls -F
bin/  include/  lib/  share/
[koichi@buildfarm:pgxc]$
```

This is used to deploy these binaries to servers with `deploy` pgxc_ctl command. If you're installing binaries with other means, you don't have to worry about this variable value.

pgxcOwner variable

Second, you will see `pgxcOwner` variable. This variable specifies owner user of Postgres-XC database.

pgxcUser variable

Next, you will see `pgxcUser` variable. This variable specifies operating system user of each server you're running Postgres-XC. Pgxc_ctl uses ssh for the operation of

Postgres-XC component and assumes that key-based authentication is configured between the server pgxc_ctl is running and other servers where you run Postgres-XC components. Key-based authentication configuration is out of the scope of pgxc_ctl.

tmpDir variable

`tmpDir` variable specifies the work directory used in pgxc_ctl locally. Typical value can be `/tmp`. Depending upon your operating system, another value can be preferred. You may want to use `$HOME/tmp` or other user-specific directory for work.

localTmpDir variable

`localTmpDir` variable specifies work directory used in the servers where you're running Postgres-XC components. Pgxc_ctl uses the same work directory among all the servers.

configBackup

`configBackup` variable specifies if you're backing up configuation file. When you change Postgres-XC cluster configuration by adding/removing nodes or promoting slave to master, pgxc_ctl updates your configuration file by adding new lines to specify such changes. If you specify the value "`y`" to this variable, pgxc_ctl will backup this change to the file specified by the followng variables.

The template specifies "n" but specifies its backup configuration for your help.

configBackupHost

`configBackupHost` variable specifies what server you'd like to backup your pgxc_ctl configuration file. It will be a good idea to backup to different server so that you can take this and run pgxc_ctl at this server when pgxc_ctl server fails.

configBackupDir

`configBackupDir` variable specifies the directory where pgxc_ctl configuration file backup is stored. If you don't specify "`y`" to `configBackup` variable, you don't have to worry about this variable.

configBackupFile

`configBackupFile` variable specifies the file name of pgxc_ctl configuration backup. If you don't specify "`y`" to `configBackup` variable, you don't have to worry about this variable.

5.5 GTM master configuration

Following is GTM master section of pgxc_ctl configuration template. It looks very simple.

```
#---- GTM ------------------------------------------------------------------------------------

# GTM is mandatory.  You must have at least (and only) one GTM master in your Postgres-XC cluster.
# If GTM crashes and you need to reconfigure it, you can do it by pgxc_update_gtm command to update
# GTM master with others.   Of course, we provide pgxc_remove_gtm command to remove it.  This command
# will not stop the current GTM.  It is up to the operator.

#---- Overall -------
gtmName=gtm

#---- GTM Master -----------------------------------------------

#---- Overall ----
gtmMasterServer=node13
gtmMasterPort=20001
gtmMasterDir=$HOME/pgxc/nodes/gtm

#---- Configuration ---
gtmExtraConfig=none         # Will be added gtm.conf for both Master and Slave (done at initilization only)
gtmMasterSpecificExtraConfig=none   # Will be added to Master's gtm.conf (done at initialization only)
```

gtmName

`gtmName` variable defines the node name for GTM. GTM master and slave shares this. Because we have only one GTM master in the cluster, you may not have a chance to use this name in the cluster operation.

gtmMasterServer

`gtmMasterServer` variable is the server you are running GTM master.

gtmMasterPort

`gtmMasterPort` variable is TCP port number GTM uses to accept connections from GTM-Proxy or coordinator/datanode backend. You should assign unique port number in the host `$gtmMasterServer`.

gtmMasterDir

`gtmMasterDir` variable is the work directory for GTM master. Similar to PostgreSQL server, GTM needs dedicated work directory to store its configuration file, status, log and other information.

gtmExtraConfig and gtmMasterSpecificExtraConfig

In most cases, your GTM configuration is complete with above three configuration parameters. Pgxc_ctl takes other configuration variables and composes GTM master configuration file. If you want to specify extra configuration parameter to GTM master, you can use `gtmExtraConfig` and `gtmMasterSpecificExtraConfig` variable.

gtmExtraConfig variable specifies the file name where additional gtm.conf configuration lines are stored. Contents of these files will go to gtm.conf file of both master and slave. gtmMasterSpecificExtraConfig variable specifies the file name where gtm.conf configuration lines only for GTM master is stored.

Details of gtm.conf file will be found at http://postgres-xc.sourceforge.net/docs/1_2_1/app-gtm.html.

Defatul value of these variables are set to "none", which means "nothing". You can specify the value "none" for file names if you don't specify any.

pgxc_ctl specifies listen_addresses, port, nodename startup configuration parameters and you should not specify these configuration values in gtmExtraConfig or gtmMasterSpecificExtraConfig files.

If you'd like to specify contents of, for example, gtmExtraConfig file, you can do it by adding lines as shown below:

```
#---- Configuration ---
gtmExtraConfig=gtmExtraConfig       # Will be added gtm.conf for both Master and Slave (done at initilization only)
cat > $gtmExtraConfig << EOF
log_min_messages = DEBUG1
EOF
gtmMasterSpecificExtraConfig=none   # Will be added to Master's gtm.conf (done at initialization only)
```

Because the configuration file is a bash script, these additional lines will setup the file.

5.6 GTM slave configuration

GTM slave section of pgxc_ctl configuration template is as follows:

```
#---- GTM Slave -----------------------------------------------

# Because GTM is a key component to maintain database consistency, you may want to configure GTM slave
# for backup.

#---- Overall ------
gtmSlave=y                  # Specify y if you configure GTM Slave.   Otherwise, GTM slave will not be configured and
                            # all the following variables will be reset.
gtmSlaveServer=node12       # value none means GTM slave is not available.  Give none if you don't configure GTM Slave.
gtmSlavePort=20001          # Not used if you don't configure GTM slave.
gtmSlaveDir=$HOME/pgxc/nodes/gtm    # Not used if you don't configure GTM slave.
# Please note that when you have GTM failover, then there will be no slave available until you configure the slave
# again. (pgxc_add_gtm_slave function will handle it)

#---- Configuration ----
gtmSlaveSpecificExtraConfig=none # Will be added to Slave's gtm.conf (done at initialization only)
```

gtmSlave

This variable specifies if you use GTM slave. Specify "y" for this value if you are configuring GTM slave. Otherwise, skip this section.

gtmSlaveServer

Specify the server name you're running GTM slave.

gtmSlavePort

Specify the port number GTM slave accepts connections. This has to be unique in the server you specified in `gtmSlaveServer` variable.

ggtmSlaveDir

Specify the work directory for GTM slave. This has to be unique in the server you specified in `gtmSlaveServer` variable.

gtmSlaveSpecificExtraConfig

Specify the file name you put gtm.conf configuration file entries specifc to this GTM slave. For details of gtm.conf, please refer to http://postgres-xc.sourceforge.net/docs/1_2_1/app-gtm.html. You will find how to setup this file in the configuration file in section 5.5.

pgxc_ctl specifies `listen_addresses`, `port`, `nodename` and `startup` configuration parameters and you should not specify these configuration values in `gtmSlaveSpecificExtraConfig` file.

5.7 GTM proxy configuration

GTM Proxy is not mandatory for Postgres-XC configuration. Because it provides GTM slave promotion to the master without interpreting Postgres-XC cluster operation, you may want to configure this as well unless you're configuring Postgres-XC for the test locally.

As mentioned in sectopm 2.4, it's a good idea to configure a GTM proxy, a coordinator and a datanode in a server for load balancing these components and leverage local socket.

GTM proxy configuration section looks like this:

```
#---- GTM Proxy -----------------------------------------------------------------------------------------------------
# GTM proxy will be selected based upon which server each component runs on.
# When fails over to the slave, the slave inherits its master's gtm proxy.  It should be
# reconfigured based upon the new location.
#
# To do so, slave should be restarted.   So pg_ctl promote -> (edit postgresql.conf and recovery.conf) -> pg_ctl restart
#
# You don't have to configure GTM Proxy if you dont' configure GTM slave or you are happy if every component connects
# to GTM Master directly.  If you configure GTL slave, you must configure GTM proxy too.

#---- Shortcuts ------
gtmProxyDir=$HOME/pgxc/nodes/gtm_pxy

#---- Overall -------
gtmProxy=y              # Specify y if you conifugre at least one GTM proxy.   You may not configure gtm proxies
                        # only when you dont' configure GTM slaves.
                        # If you specify this value not to y, the following parameters will be set to default empty values.
                        # If we find there're no valid Proxy server names (means, every servers are specified
                        # as none), then gtmProxy value will be set to "n" and all the entries will be set to
                        # empty values.
gtmProxyNames=(gtm_pxy1 gtm_pxy2 gtm_pxy3 gtm_pxy4) # No used if it is not configured
gtmProxyServers=(node06 node07 node08 node09)           # Specify none if you dont' configure it.
gtmProxyPorts=(20001 20001 20001 20001)             # Not used if it is not configured.
gtmProxyDirs=($gtmProxyDir $gtmProxyDir $gtmProxyDir $gtmProxyDir)  # Not used if it is not configured.

#---- Configuration ----
gtmPxyExtraConfig=none      # Extra configuration parameter for gtm_proxy.  Coordinator section has an example.
gtmPxySpecificExtraConfig=(none none none none)
```

gtmProxyDir

This is a shortcut to specify same value for `gtmProxyDirs` array elements as described later.

gtmProxy

`gtmProxy` specifies if you are configuring GTM proxy. Specify "`y`" if you are configuring GTM proxy. Specify "`n`" otherwise.

gtmProxyNames

`gtmProxyNames` specifies names of GTM proxies. Because GTM proxies are configured in more than one server, each GTM proxy need to have unique name and is specified as an array. In this template, GTM proxy, coordinator and datanode are configured in four servers.

gtmProxyServers

`gtmProxyServers` specifies server for each GTM proxy. This is also an array. Specify servers for corresponding GTM proxy specified in `gtmProxyNames`.

gtmProxyPorts

gtmProxyPorts specifies port number of each GTM proxy. This is also an array like `gtmProxyNames`. Port number must be unique in each servers specified in `gtmProxyServers` parameter.

gtmProxyDirs

GTM proxy needs unique work directory. `gtmProxyDirs` parameter specifies this. In the template, work variable gtmProxyDir is used to assign the same value to each array element. You can use similar way for you convenience.

gtmPxyExtraConfig

Specify the file name which contain extra `gtm_proxy.conf` configuration lines. Content of this file will go to all the `gtm_proxy.conf` files you are configuring. Specify "`none`" if you are not using this feature.

Details if gtm_proxy.conf file will be found at http://postgres-xc.sourceforge.net/docs/1_2_1/app-gtm-proxy.html.

Out of `gtm_proxy.conf` configuration, `listen_addresses`, `worker_threads` and `gtm_connect_retry_interval` will be set by pgxc_ctl and you can change them with `gtmPxyExtraConfig` and `gtmPxySpecificExtraConfig`.

pgxc_ctl will also setup nodename, port, gtm_host, and gtm_port. They comes at the last of gtm_proxy.conf so specifying them in `gtmPxyExtraConfig` or `gtmPxySpecificExtraConfig` will not work.

gtmPxySpecificExtraConfig

You can specify extra configuration for each GTM proxy with this parameter. Specify file name which contains extra gtm_proxy.conf lines for each GTM proxy as an element of this array. Specify "`none`" element value if you don't use this.

5.8 Coordinator master configuration

If you became familiar with GTM proxy configuration, you will find coordinator and datanode configuration is quite similar to it. Yes, it is and with just a few addition.

Coordinator master configuration section looks as follows. Please be careful that coordinator slave configuration is at the middle of this configuration, which will be explained in the next section.

```
#---- Coordinators ----------------------------------------------------------------------------------------------

#---- shortcuts ----------
coordMasterDir=$HOME/pgxc/nodes/coord
coordSlaveDir=$HOME/pgxc/nodes/coord_slave
coordArchLogDir=$HOME/pgxc/nodes/coord_archlog

#---- Overall ------------
coordNames=(coord1 coord2 coord3 coord4)        # Master and slave use the same name
coordPorts=(20004 20005 20004 20005)            # Master and slave use the same port
poolerPorts=(20010 20011 20010 20011)           # Master and slave use the same pooler port
coordPgHbaEntries=(192.168.1.0/24)              # Assumes that all the coordinator (master/slave) accepts
... (Omitted) ...

#---- Master -------------
coordMasterServers=(node06 node07 node08 node09)        # none means this master is not available
coordMasterDirs=($coordMasterDir $coordMasterDir $coordMasterDir $coordMasterDir)
coordMaxWALsernder=5    # max_wal_senders: needed to configure slave. If zero value is specified,
                        # it is expected to supply this parameter explicitly by external files
                        # specified in the following.   If you don't configure slaves, leave this value to zero.
coordMaxWALSenders=($coordMaxWALsernder $coordMaxWALsernder $coordMaxWALsernder $coordMaxWALsernder)
                        # max_wal_senders configuration for each coordinator.

#---- Slave -------------
... (omitted) ...

#---- Configuration files---
# Need these when you'd like setup specific non-default configuration
# These files will go to corresponding files for the master.
# You may supply your bash script to setup extra config lines and extra pg_hba.conf entries
# Or you may supply these files manually.
coordExtraConfig=coordExtraConfig   # Extra configuration file for coordinators.
                        # This file will be added to all the coordinators'
                        # postgresql.conf
# Please note that the following sets up minimum parameters which you may want to change.
# You can put your postgresql.conf lines here.
cat > $coordExtraConfig <<EOF
#================================================
# Added to all the coordinator postgresql.conf
# Original: $coordExtraConfig
log_destination = 'stderr'
logging_collector = on
log_directory = 'pg_log'
listen_addresses = '*'
max_connections = 100
EOF

# Additional Configuration file for specific coordinator master.
# You can define each setting by similar means as above.
coordSpecificExtraConfig=(none none none none)
coordExtraPgHba=none    # Extra entry for pg_hba.conf.  This file will be added to all the coordinators' pg_hba.conf
coordSpecificExtraPgHba=(none none none none)
```

First three variable settings for `coordMasterDir`, `coordSlaveDir` and `coordArchDir` are shortcuts to specify the same value to each array element. You can write any script for your convenience.

coordNames

Specify each coordinator name in this array element.

coordMasterDirs

Specify working directory for each coordinator in this array element. In this template, `coordMasterDir` variable is used to assign the same value to all the elements.

coordPorts

Specify the port number which each coordinator uses to accept connection from application or other coordinators. This value must be unique in the server specified in `coordMasterServers` variable and `coordSlaveServers` variable if you are configuring coordinator slaves.

This template is based upon circular HA configuration where each coordinator slave runs at the next server and master and slave uses the same port. Please note that each coordinator is assigned different port to meet this configuration.

poolerPorts

Coordinator implements connection pooler internally to pool connection to other coordinators and datanodes. This variable specifies port number which the pooler uses internally. The value must be unique in the server specified in `coordMasterServers` variable and `coordSlaveServers` variable if you are configuring coordinator slaves,

coordPgHbaEntries

This is a shortcut of configuring `pg_hba.conf` file of each coordinator. Each element specified in this array will be converted into "`host all` xxx `trust`" format to go to pg_hba.conf where xxx is the value of the element. If you don't like to have such setups, you should use `coordExtraPgHba` or `coordSpecificExtraPgHba` variable.

coordMasterServers

This array specifis what server each coordinator runs.

coordMasterDirs

This array specifies work directory of each coordinator. Please note that this template uses variable `coordMasterDir` to assign the same value to each array element.

coordMaxWalSenders

This array specifies max_wal_sender configuration parameter value for each coordinator. If you are configuring coordinator slave, this value must be positive.

coordExtraConfig and coordSpecificExtraConfig

`coordExtraConfig` specifies the file name which contains postgres.conf configuration entries for all the coordinators. The following lines are the script to set up the file.

Just like GTM proxy, you can specify `postgres.conf` file entry for each coordinator with `coordSpecificExtraConfig` array. Specify "`none`" for the element value if you don't use it.

pgxc_ctl will set up `port`, `pooler_port`, `gtm_host`, and `gtm_port` configuration at the last part of coordinator's `postgresql.conf` file. Reconfiguring these parameters in `coordExtraConfig` and `coordSpecifcExtraConig` will not work.

If you are configuring coordinator slave, pgxc_ctl will configure `wal_level`, `archive_mode`, `archive_command`, and `max_wal_senders` as well at the last part. Reconfiguring these parameters in `coordExtraConfig` and `coordSpecificExtraConfig` will not work either in this case.

coordExtraPgHba and coordSpecificExtraPgHba

`coordExtraPgHba` specifies the file name which contains lines to go to `pg_hba.conf` file of all the coordinators.

Each element of `coordSpecificExtraPgHba` array specifies the file name which contains lines of `pg_hba.conf` file for each coordinator.

5.9 Coordinator slave configuration

Please note that pgxc_ctl configures coordinator to use the same port as their masters.

Configuration sectio for coordinator slave looks like this:

```
#---- Slave -------------
coordSlave=y            # Specify y if you configure at least one coordiantor slave.  Otherwise, the following
                        # configuration parameters will be set to empty values.
                        # If no effective server names are found (that is, every servers are specified as none),
                        # then coordSlave value will be set to n and all the following values will be set to
                        # empty values.
coordSlaveSync=y        # Specify to connect with synchronized mode.
coordSlaveServers=(node07 node08 node09 node06)         # none means this slave is not available
coordSlaveDirs=($coordSlaveDir $coordSlaveDir $coordSlaveDir $coordSlaveDir)
coordArchLogDirs=($coordArchLogDir $coordArchLogDir $coordArchLogDir $coordArchLogDir)
```

coordSlave

Specify "y" if you are configuring coordinator slaves. Otherwise, specify "n".

coordSlaveSync

Specify "y" if you use synchronous wal shipping for the slave. At present, you should specify "y" because asynchronous wal shipping could lose some transactions at promote which may make cluster inconsistent.

coordSlaveServers

Specify which servers each coordinator slave runs.

coordSlaveDirs

Specify work directory for each coordiantor slave.

coordArchLogDirs

Specify a directory to receive WAL archive for each coordinator slave.

5.10 Datanode master configuration

Datanode master and slave configuration is very similar to coordinator master and slave configuration. One major difference is that datanodes does not have pooler.

Datanode master configuration section is as follows:

```
#---- Datanodes ---------------------------------------------------------------------------------------------------

#---- Shortcuts --------------
datanodeMasterDir=$HOME/pgxc/nodes/dn_master
datanodeSlaveDir=$HOME/pgxc/nodes/dn_slave
datanodeArchLogDir=$HOME/pgxc/nodes/datanode_archlog

#---- Overall ---------------
#primaryDatanode=datanode1              # Primary Node.
# At present, xc has a problem to issue ALTER NODE against the primay node.  Until it is fixed, the test will be done
# without this feature.
primaryDatanode=datanode1               # Primary Node.
datanodeNames=(datanode1 datanode2 datanode3 datanode4)
datanodePorts=(20008 20009 20008 20009) # Master and slave use the same port!
datanodePgHbaEntries=(192.168.1.0/24)   # Assumes that all the coordinator (master/slave) accepts
                                        # the same connection
                                        # This list sets up pg_hba.conf for $pgxcOwner user.
                                        # If you'd like to setup other entries, supply them
                                        # through extra configuration files specified below.
# Note: The above parameter is extracted as "host all all 0.0.0.0/0 trust".  If you don't want
# such setups, specify the value () to this variable and suplly what you want using datanodeExtraPgHba
# and/or datanodeSpecificExtraPgHba variables.

#---- Master ----------------
datanodeMasterServers=(node06 node07 node08 node09) # none means this master is not available.
                                                    # This means that there should be the master but is down.
                                                    # The cluster is not operational until the master is
                                                    # recovered and ready to run.
datanodeMasterDirs=($datanodeMasterDir $datanodeMasterDir $datanodeMasterDir $datanodeMasterDir)
datanodeMaxWalSender=5                              # max_wal_senders: needed to configure slave. If zero value is
                                                    # specified, it is expected this parameter is explicitly supplied
                                                    # by external configuration files.
                                                    # If you don't configure slaves, leave this value zero.
datanodeMaxWALSenders=($datanodeMaxWalSender $datanodeMaxWalSender $datanodeMaxWalSender $datanodeMaxWalSender)
                    # max_wal_senders configuration for each datanode

#---- Slave -----------------

... (Omitted) ...

# ---- Configuration files ---
# You may supply your bash script to setup extra config lines and extra pg_hba.conf entries here.
# These files will go to corresponding files for the master.
# Or you may supply these files manually.
datanodeExtraConfig=none    # Extra configuration file for datanodes.  This file will be added to all the
                            # datanodes' postgresql.conf
datanodeSpecificExtraConfig=(none none none none)
datanodeExtraPgHba=none     # Extra entry for pg_hba.conf.  This file will be added to all the datanodes' postgresql.conf
datanodeSpecificExtraPgHba=(none none none none)
```

Similar to coordinator, slave configuration is placed in the middle, which will be described in the next section.

`datanodeMasterDir`, `datanodeSlaveDir`, `datanodeArchiLogDir` are shortcuts used in the following configuration.

primaryDatanode

This configuration is unique to the datanode, specifying primary datanode name. Primary datanode is the datanode where replicated table update takes place first. This is how to maintain replicated table consisitent. In the future release of Postgres-XC, primary datanode may be determined automatically and this parameter may become obsolete.

datanodeNames

This array specifies name of the datanodes. Node name of primaryDatanode has to be specified in one of the element.

datanodePorts

Specifies the port number which datanode postmaster uses to accept connections.
Master and slave of each datanode uses the same port number and this number has to be unique in the servers running datanode master or slave, if configured.

datanodePgHbaEntries

Shortcut to specify pg_hba.conf file of each datanode. Please see CoordPgHbaEntires for details.

datanodeMasterServers

This array specifies server name where each datanode master runs.

datanodeMasterDirs

This array specifies the directory for each datanode master. This hast to be unique in the server where the coordinator master is running.

datanodeMaxWalSenders

This array specifies max_wal_senders configuration for each datanode's postgresql.conf.
If you are configuring datanode slave, this value has to be positive.

datanodeExtraConfig

Specify the file name which contains extra lines for postgresql.conf file of all the datanodes. Specify "none" if you are not using this.

datanodeSpecificExtraConfig

This array specifies the file name which contains extra lines for postgresql.conf file of each corresponding datanode.

datanodeExtraPgHba

Specify the file name which contains additional lines for pg_hba.conf file of all the datanodes. Specify "none" if you are not using this.

datanodeSpecificExtraPgHba

This array specifies the file name which contains extra lines for pg_hba.conf of each corresponding datanode.

5.10 Datanode slave configuration

Similar to coordinators, datanode slave uses the same port number as its master.

Datanode slave configuration section looks like:

```
#---- Slave -----------------
datanodeSlave=y         # Specify y if you configure at least one coordiantor slave.  Otherwise, the following
                        # configuration parameters will be set to empty values.
                        # If no effective server names are found (that is, every servers are specified as none),
                        # then datanodeSlave value will be set to n and all the following values will be set to
                        # empty values.
datanodeSlaveServers=(node07 node08 node09 node06)  # value none means this slave is not available
datanodeSlaveSync=y     # If datanode slave is connected in synchronized mode
datanodeSlaveDirs=($datanodeSlaveDir $datanodeSlaveDir $datanodeSlaveDir $datanodeSlaveDir)
datanodeArchLogDirs=( $datanodeArchLogDir $datanodeArchLogDir $datanodeArchLogDir $datanodeArchLogDir )
```

datanodeSlave

Specify “y” if you are configuring datanode slaves. Otherwise, specify “n”.

datanodeSlaveServers

This array specifies the server where each datanode slave is running.

datanodeSlaveSync

Specify if you are using synchronous wal shipping. To maintain database consistency, please specify just “y” here to avoid a chance to lose transactions at promotion.

datanodeSlaveDirs

This array specifies the directory for each datanode.

datanodeArchLogDirs

This array specifies the directory to receive each datanode’s archive WAL.

6. Initialize Postgres-XC cluster

This chapter describes how to initialize your Postgres-XC cluster.

When you obtain pgxc_ctl configuration file tempalte with pgxc_ctl prepare command, you have built `$HOME/pgxc_ctl` direcoty and your `pgxc_ctl.conf` file at this directory.

You have designed your Postgres-XC configuration and edited `pgxc_ctl.conf` file.

You have configured ssh connection from the computer you are running pgxc_ctl to each server you are running one or more Postgres-XC components.

Now you are ready to initialize your Postgres-XC cluster with pgxc_ctl.

6.1 Invoke pgxc_ctl

Now invoke `pgxc_ctl`. If pgxc_ctl does not find any error in your configuration, it will print a promt asking for a command.

You may find errors in the configuration. Then, edit the configuration file and start again.

6.2 Deploy Postgrs-XC binaries to servers

You should deploy all Postgres-XC binaries to all the servers you are running Postgres-XC components. If you have installed this by binary package or manually, you can skip this section.

If you are deploying binaries, then type `deploy all` and return. Pgxc_ctl will look for servers where at least one Postgres-XC component runs and copy binaries to their installation directory specified as pgxcInstallDir configuration variable.

Please note that deploy all does not take care of PATH environment in your shell. You should do this manually.

6.3 Initialize the cluster

Type `init all` and the return. Pgxc_ctl will do everything needed to configure and start up your Postgres-XC database cluster.

Pgxc_ctl provides more step-by-step initialization. This is for the test and does not provide cluster configuration using `CREATE NODE` statement. It is more convenient to use `init all` command.

If there's something wrong, you will obtain an error. Don't worry. If you need any correction to your cofiguration file and do it over from the scratch, you should do the following.

1) Issue `kill all` command againt pgxc_ctl command prompt to kill all the processes at servers. It it doesn't work, then you should kill all the process of gtm, gtm_proxy and postgres manually by visiting each server.

2) Issue `clean all` command against pgxc_ctl to clean up all the working directories.
3) Fix the issue in the configuration file or in the settings pgxc_ctl assumes.
4) If you need to have additional servers to be involved and deployed Postgres-XC binaries using `deploy all` command, issue `deploy` *`newserver`* command agains pgxc_ctl prompt, which deploys Postgres-XC binaries to *`newserver`*. Otherwiser, install Postgres-XC binary in your way.
5) Start his step from the beginning.

6.4 What `init all` does

Init all does plenty of work inside to initialize each component and configure them to work together.

A) Initialize GTM master

1) Kill gtm process if exists, remove the work directory if exists and then create it.
2) Run `initgtm` utility to initialize gtm environment.
3) Configure `gtm.conf` file for the master.
4) Setup GTM to start with appropriate GXID value.
5) Start GTM master

B) Initialize GTM slave if configured

1) Kill gtm process if exists, remove work directory if exists and then create it.
2) Run `initgtm` to initialize gtm environment.
3) Configure `gtm.conf` file for the slave.
4) Start GTM slave.

C) Initialize GTM proxies if configured

The following steps are done for each gtm_proxy in parallel.

1) Kill gtm_process if exists, remove work directory if exists and them create it.
2) Run `initgtm` to initialize gtm_proxy environment.
3) Configure `gtm_proxy.conf` file.
4) Start GTM proxy.

D) Initialize coordinator masters

The following steps are done for each coordinator master in parallel.

1) Initialize the work directory.
2) Run `initdb` to initialize a coordinator.
3) Configure `postgresql.conf` file.
4) If coordinator slave is configured, add wal shipping configuration to `postgresql.conf` file.
5) Start coordinator master.

E) Initialize coordinator slaves if configured

The following steps are done for each coordinator slave in parallel.

1) Initialize the work directory.
2) Run `pg_basebackup` utility to build the base backup.
3) Configure `recovery.conf`.
4) Add `postgresql.conf` configuration entries to run as the slave.
5) Start coordinator slave.

F) Initialize datanode masters

The following steps are done for each datanode master in parallel.

1) Initialize the work directory.
2) Run `initdb` to initialize a datanode.
3) Configure `postgresql.conf` file.
4) If datanode slave is configured, add wal shipping configuration to `postgresql.conf` file.
5) Start datanode master.

G) Initialize datanode slaves if configured

The following steps are done for each datanode slave in parallel.

1) Initialize the work directory
2) Run `pg_basebackup` utility to build the base backup.
3) Configure `recovery.conf`.
4) Add `postgresql.conf` configuration entries to run as the slave.
5) Start datanode slave.

H) Node configuration

1) Run CREATE NODE and ALTER NODE statement at each coordinator to finalize the node configuration and make each coordinator ready to accept connections.

7. Build your database

When you are successfull in `init all` pgxc_ctl command, you are ready to run psql or other utilities. Most of PostgreSQL utilities are ported to Postgres-XC.

They accept `-h` and `-p` command line option to specify what coordinator to connect. As an alternative, pgxc_ctl provides two built-in commands, Createdb and Psql.

They choose one of the available coordinator and connect to it. You can specify what coordinator to connect with `-` followed by a coordinator name to connect to, not host name or port number.

So you can create your own database by issuing `Createdb` *`newdb`* against pgxc_ctl prompt, or pgxc_ctl command argument.

8. Run your SQL statements

Pgxc_ctl provides Psql built-in command which invokes psql against specified coordinator. You can specify the coordinator name after '-' argument like

$ Psql - coord1

Where, coord1 is the coordinator name. If you don't specify coordinator name, pgxc_ctl will choose one. You can specify any other psql command options too.

Then you can issue any coordinator Postgres-XC SQL statements.

9. Writing applications

Postgres-XC's libpq interface is binary compatible with PostgreSQL so you can write your application with the same manner as PostgreSQL. Because of the clustering nature, there are several SQL statements which Postgres-XC does not support. Also, there are several SQL statements specific to Postgres-XC. For details, please refer to Postgres-XC document at http://postgres-xc.sourceforge.net/docs/1_1/ or http://postgres-xc.sourceforge.net/docs/1_2_1/

10. Backing up Postgres-XC cluster

10.1 pg_dump and pg_dumpall

As in the case of PostgreSQL, pg_dump and pg_dumpall are the basic backup tool of Postgres-XC. You can connect to one of the coordinators using -h and -p option (sorry, pgxc_ctl does not provide buildt-in command such as Pg_dump/Pg_dumpall so far). This is almost the same as PostgreSQL.

Backup is consisitent and can be restored using psql or pg_restore.

10.2 WAL-shipping backup

You can configure Postgres-XC coordinator and datanode to enable WAL-shipping backup manually. At present, pgxc_ctl does not support this feature. This paper does not provide any further description on it so far.

Pgxc_ctl provides master/slave configuration and failover of each node. Please use this feature now.

11. Recovery from the backup

11.1 Recovery with pg_dump/pg_dumpall

You can restore the database from the backup you made using pg_dump or pg_dumpall. First, re-initialize your cluster and then apply the dump using psql (when the backup was taken in text format) or pg_restore.

11.2 Recovery from WAL shipping archive

For the same reason as 10.2, this is out of the scope of this paper.

12. Node failover

If you configure slave for GTM, coordinator or datanode and one of hem fails, you can promote the slave and swithch over the master.

Pgxc_ctl provides only manuall promotion, not automatic failover. The background is as follows:

1) Automatic failover should be integrated with other resource failover, such as server hardware, network, storage and other software resource such as web server and application server.
2) 1) very widely depend upon individual system integration/configuration and it may not be adequate to provide automatic failover system just within database system.

The following sections will describe pgxc_ctl command interface to promote slaves.

12.1 GTM slave promotion

When GTM master does not work and you are running GTM slave, you can promote GTM slave to the master. Here is how to do at pgxc_ctl.

You have configured GTM Proxy

With GTM Proxy, you can promote GTM slave without stopping Postgres-XC cluster. If live transactions needs to communicate with GTM while GTM master is out, they will be aborted but you don't have to restart nodes.

First, issue `failover gtm` command at pgxc_ctl command prompt like:

```
PGXC$ failover gtm
```

Then, you issue reconnect gtm_proxy all command like:

```
PGXC$ reconnect gtm_proxy all
```

Then, all gtm_proxies will connect to the new master just promoted.

Through this step, the following will be done:

1) Run `gtm_ctl promote` command at gtm slave.
2) Configure `gtm.conf` of the promoted gtm so that it starts as the master next time.
3) Update your configuration file to reflect these changes. Backup it if specified.

Please note that these commands does not stop old GTM master.

You have not configured GTM Proxy

Pgxc_ctl does not provide a convenient way to deal with this situation. You have to do the following manually.

1) Run `gtm_ctl` promote command at gtm slave.
2) Edit `postgresql.conf` file so that they connect to the new gtm master.
3) Restart all the coordinators and datanodes.

12.2 Coordinator slave promotion

If any coordinator fails and it has a slave running, you can promote it. To do this, you should invoke `failover coordinator` command like:

```
PGXC$ failover coordinator coodname
```

where ***coordname*** is the coordinator name to promote.

Pgxc_ctl will do the following:

1) Because coordinator slave is running at a different server for the master, determine which gtm_proxy promoting coordinator should connect.
2) Unregister the coordinator from GTM.
3) Promote the slave using `pg_ctl promote`.
4) Edit `postgresql.conf` file to reflect the change in target gtm_proxy. If gtm_proxy is not configured in the server, gtm will be chosen.
5) Issue `pg_ctl restart` to reflect these changes.
6) Update pgxc_ctl configuration file and backup it if specified.
7) Issue `ALTER NODE` statement and `pgxc_pool_reload()` function at all the coordinators to reflect this change.

Please note that all the other coordinator masters should be running to handle `ALTER NODE` statement.

12.3 Datanode slave promotion

If any datanode fails and it has a slave running, you can promote it. To do this, you should invoke `failover datanode` command like:

```
PGXC$ failover datanode datanodename
```

where ***datanodename*** is the datanode name to promote.

Pgxc_ctl will do the following:

1) Because datanode slave is running at a different server for the master, determine which gtm_proxy promoting datampde should connect.
2) Unregister the datanode from GTM.
3) Promote the slave using `pg_ctl promote`.
4) Edit `postgresql.conf` file to reflect the change in target gtm_proxy. If gtm_proxy is not configured in the server, gtm will be chosen.
5) Issue `pg_ctl restart` to reflect these changes.
6) Update pgxc_ctl configuration file and backup it if specified.
7) Issue `ALTER NODE` statement and `pgxc_pool_reload()` function at all the coordinators to reflect this change.

Please note that all the coordinator masters should be running to handle `ALTER NODE` statement.

13. Adding nodes

Pgxc_ctl provides series of command to add nodes. While adding a node, you don't have to stop the whole Postgres-XC cluster but some node may need restart. This chapter describes the basics of each node addition.

13.1 Adding GTM slave

If you did not configure GTM slave or you don't have GTM slave because original GTM slave has been promoted to the master, you can add GTM slave to your Postgres-XC cluster.

Pgxc_ctl provides `add gtm slave` command for this purpose. The syntax of the command is as follows:

```
PGXC$ add gtm slave name host port dir
```

name, ***host***, ***port***, and ***dir*** are the node name, host where GTM slave runs, port assigned to GTM slave to accept connections and its working directory, respectively.

Adding GTM slave does not affect active transactions.

When adding GTM slave, pgxc_ctl does the following:

1) Update pgxc_ctl configuration file and backup if specified.
2) Initialize gtm slave and start it. See 6.4 B) for details.

13.2 What about GTM master?

GTM master is Postgres-XC's vital component and it has to be configured and running. Pgxc_ctl does not provide a means to "add" GTM master. To move GTM master to other server, run gtm slave at the target server and promote it.

13.3 Adding a GTM proxy

If you are adding coordinator or datanode at a server where gtm_proxy is not configured, you may want to add gtm_proxy at this server.

You can do this by issuing `add gtm_proxy` command like:

```
PGXC$ add gtm_proxy name host port dir
```

name, ***host***, ***port***, and ***dir*** are the node name, host where the GTM proxy runs, port assigned to GTM proxy to accept connections and its working directory, respectively.

If you have not installed Postgres-XC binary to the server, you should do it as described in 6.2.

When adding GTM proxy, pgxc_ctl will do the following:

1) Update pgxc_ctl configuration file and backup it if specified.
2) Configure GTM proxy and start it. See 6.4 C) for details.

13.4 Adding a coordinator master

If you have not installed Postgres-XC binary to the server, you should do it as described in 6.2.

If you are adding a coordinator master at a server where GTM proxy is not configured, you may want to configure it first, as described in 13.3.

Adding a coordinator master in pgxc_ctl is simple. Just invoke add coordinator master command like:

```
PGXC$ add coordinator master name host port pooler dir
```

name, ***host***, ***port***, ***pooler***, ***dir*** are the node name, host where the new coordinator master runs, port assigned to the coordinator to accept connections, port assigned to coordinator connection pooler, and its working directory, respectively.

When adding a coordinator master, pgxc_ctl will do the following:

1) Update pgxc_ctl configuration file and back up it if specified.
2) Initialize the working directory and run `initdb` to for initial configuration of the new coordinator master.
3) Determine GTM proxy or GTM to use and update new coordinator master's `postgresql.conf` file.
4) Edit `pg_hba.conf` file to accept minimum connection specified in `coordPgHbaEntries` variable. See 5.8 for details.
5) Choose an active coordinator and issue `pgxc_lock_for_backup()` to block DDL issued to all the active coordinators.
6) Choose an active coordinator and issue `pg_dumpall` to dump all the catalog information to be imported to the new coordinator master.
7) Start the new coordinator master with `-Z restoremode` and import the catalog exported at the step 6).
8) Stop the new coordinator and start it with `-Z coordinator` option as a coordinator.
9) Issue `CREATE NODE` or `ALTER NODE` and then `pgxc_pool_reload()` at all the coordinators to reflect the change.
10) Close the session opened in the step 5) to release DDL lock.

13.5 Adding a coordinator slave

Please consider to install Postgres-XC binaries and configure GTM proxy as described in 13.4.

You can add a coordinator slave just as follows:

```
PGXC$ add coordinator slave name host dir archDir
```

name, ***host***, ***dir*** and ***archDir*** are the node name, host where the new coordinator slave runs, its working directory and the directory to receive WAL archive from its master, respectively.

When adding a coordinator slave, pgxc_ctl will do the following:

1) Initialize the working directory and archive WAL directory.
2) Reconfigure the master's `postgresql.conf` file to begin WAL shipping.
3) Reconfigure the master's `pg_hba.conf` file to accept WAL shipping connection from the new slave.
4) Update pgxc_ctl configuration file and backup it if specified.
5) Restart the master to relfect changes done in 2) and 3).
6) Run `pg_basebackup` to build the master's base backup at the slave's work directory to start with.
7) Update the slave's `postgresql.conf` to run as a slave.
8) Configure the slave's `recovery.conf` file to connect to the master for log shipping.
9) Start the slave.

13.6 Adding a datanode master

Adding a datanode master is similar to adding a coordinator master as described in 13.4. Please consider to install Postgres-XC binaries and GTM proxy if needed, as described in 13.4.

To add a datanode master, you can issue add datanode master command as follows:

```
PGXC$ add datanode master name host port dir
```

name, ***host***, ***port***, and ***dir*** are the ndoe name, host where the new datanode master runs, port number used to accept connections, and the working directory, respectively.

Please note that adding a datanode master does not redistribute the table data automatically because you can specify a set of nodes to distribute or replicate each table. To redistribute tables, use ALTER TABLE statement as described in http://postgres-xc.sourceforge.net/docs/1_2_1/sql-altertable.html and http://postgres-xc.sourceforge.net/docs/1_1/sql-altertable.html.

When adding a datanode master, pgxc_ctl will do the following:

1) Update pgxc_ctl configuration file and back up it if specified.
2) Initialize the working directory and run `initdb` to for initial configuration of the new datanode master.
3) Determine GTM proxy or GTM to use and update new datanode master's `postgresql.conf` file.
4) Edit `pg_hba.conf` file to accept minimum connection specified in `datanodePgHbaEntries` variable. See 5.10 for details.
5) Choose an active coordinator and issue `pgxc_lock_for_backup()` to block DDL issued to all the active coordinators[4].
6) Choose an active datanode and issue `pg_dumpall` to dump all the catalog information to be imported to the new coordinator master.
7) Start the new datanode master with `-Z restoremode` and import the catalog exported at the step 6).
8) Stop the new datanode and start it with `-Z datanode` option as a datanode.
9) Issue `CREATE NODE` and `pgxc_pool_reload()` at all the coordinators to reflect the change.
10) Close the session opened in the step 5) to release DDL lock.

[4] In the current release, pgxc_lock_for_backup() is targetted to a datanode master and does not propagate to other nodes. It should have targeted to a coordinator. Fix will be committed and available at the next minor release.

13.7 Adding a datanode slave

Please note that the master datanode must be configured and running to add a datanode slave. Please also consider to install Postgres-XC binaries and configure GTM proxy if needed, as described in 13.4.

Adding datanode slave is quite similar to adding coordinator slave. You can do this as follows:

```
PGXC$ add datanode slave name host dir archDir
```

name, ***host***, ***dir*** and ***archDir*** are the node name, host where the new datanode slave runs, its working directory and the directory to receive WAL archive from its master, respectively.

When adding a datanode slave, pgxc_ctl will do the following:

1) Initialize the working directory and archive WAL directory.
2) Reconfigure the master's `postgresql.conf` file to begin WAL shipping.
3) Reconfigure the master's `pg_hba.conf` file to accept WAL shipping connection from the new slave.
4) Update pgxc_ctl configuration file and backup it if specified.
5) Restart the master to relfect changes done in 2) and 3).
6) Run `pg_basebackup` to build the master's base backup at the slave's work directory to start with.
7) Update the slave's `postgresql.conf` to run as a slave.
8) Configure the slave's `recovery.conf` file to connect to the master for log shipping.
9) Start the slave.

14. Removing nodes

As mentioned, GTM master is a vital Postgres-XC component and it is not allowed to remove it. GTM master has to be running when Postgres-XC cluster is running.

14.1 Removing GTM slave

You should stop GTM slave before removing. Pgxc_ctl provides command to do this:

```
PGXC$ stop gtm slave
```

Then, you can remove GTM slave by:

```
PGXC$ remove gtm slave
```

To remove gtm slave, pgxc_ctl does the following:

1) Update pgxc_ctl configuration file and back up it if specified.

14.2 Removing GTM proxy

Before you remove a gtm proxy, you should stop it. Pgxc_ctl provides a command to do as follows:

```
PGXC$ stop gtm_proxy name
```

where ***name*** is gtm_proxy name to stop.

Then, you can remove the gtm_proxy as follows:

```
PGXC$ remove gtm_proxy name
```

Please note that you should configure coordinators and datanodes connecting to this gtmm proxy and restart them. It is advised that you can remove a gtm proxy if no coordinators or datanodes are connected to it any longer.

14.3 Removing coordinator master

Because a coordinator does not store uesr data, it is not halmful to remove a coordinator master. Please do not issue DDL while you are removing coordinator master, or such DDL could be propagated to the removeing coordinator.

Pgxc_ctl does not care if the removing coordinator master is running. If it is runnig, pgxc_ctl will stop it.

The command to remove a coordinator master is as follows:

```
PGXC$ remove coordinator master name
```

where ***name*** is the coordinator node name to remove.

Pgxc_ctl will do the following to remove a coordinator master.

1) Remove the slave of the removing coordinator master if configured. See the next section for details.
2) Issue `DROP NODE` statement at all the other coordinator to remove the coordinator from all the other coordinators.
3) Stop the coordinator master if running.
4) Update pgxc_ctl configuration file and back up it if specified.

14.4 Removing a coordinator slave

You can remove a coordinator slave by following pgxc_ctl command.

```
PGXC$ remove coordinator slave name
```

where name is coordinator name to remove.

Pgxc_ctl will do the following to remove a coordinator slave.

1) If the coordinator slave is running, stop it.
2) Update the master's configuration to disable log shipping.
3) Restart the master.
4) Update pgxc_ctl configuration file and back up it if specified.

14.5 Removing a datanode master

Before you remove a datanode master, please be sure that the removing datanode does not contain any user data. You can check this by ussing `\d+ pattern` command to psql. Issue `ALTER TABLE` statement to each table to remove the datanode from its replication or distribution nodes.

You can remove a datanode master with the command:

`PGXC$` **`remove datanode master`** ***`name`***

where ***`name`*** is the datanode node name to remove.

Pgxc_ctl will do the following to remove a datanode master.

1) If slave is configured for this mater, remove it. See 14.6 for details.
2) Issue `DROP NODE` statement in all the coordinators to remove this datanode.
3) Update pgxc_ctl configuration file and back up it if specified.

14.6 Removing a datanode slave

Removing a datanode slave is quite similar to removing a coordinator slave. You can do this by the following pgxc_ctl command:

```
PGXC$ remove datanode slave name
```

where ***name*** is the datanode name to remove.

Pgxc_ctl will do the following to remove a datanode slave.

1) If the coordinator slave is running, stop it.
2) Update the master's configuration to disable log shipping.
3) Restart the master.
4) Update pgxc_ctl configuration file and back up it if specified.

15. Star Schema (appendix)

As described in 2.4, Postgres-XC architecture is build to leverage a database design which consists of few but big tables updated frequently and smaller but many tables which are very stable. This structure is known as star schema. This section describes about star schema and how Postgres-XC leverages it.

Star schema is found in many data warehouse and OLTP applications. Star schema consists of a few and big “fact” tables and many “dimention” tables. For example, sales database may include “sales fact” as a fact table and “product dimention” and “store dimention” table. Fact tables are big in size and updated frequently. On the other hand, dimention tables are small in size and more stable compared with fact tables. Figure 7 shows typical star schema taken from http://support.sas.com/documentation/cdl/en/spdsug/64018/HTML/default/viewer.htm#n0mlj75x9c4dtzn1ves84e1op3jt.htm

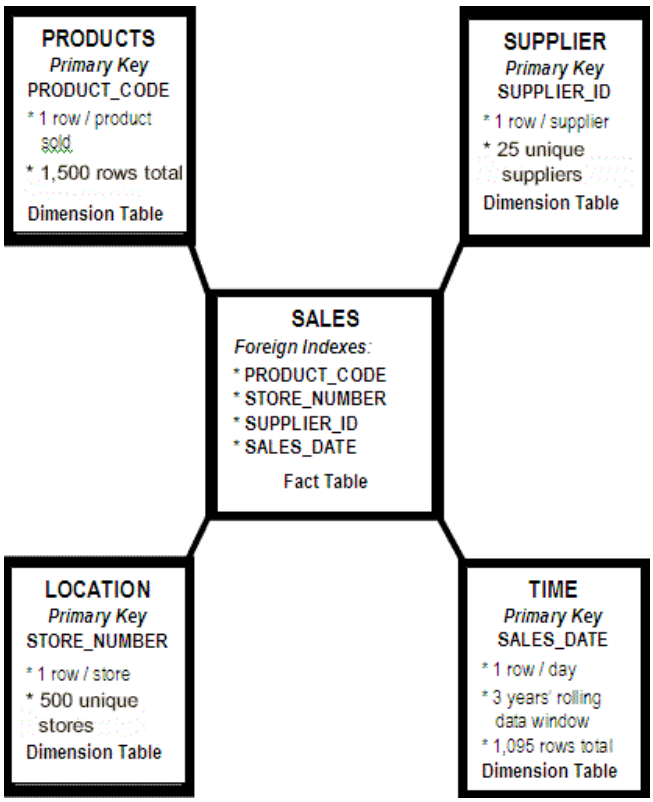

Figure 7 Typical star schema
(See above for the source)

Postgres-XC architecture is build to leverage star schema characteristics. Usually, if there’s more than one fact tables, they tend to share candidate keys. In Postgres-XC, it is desirable to shard fact tables using one of such common candidate key. In this way, we cay shard one (or few) big table into smaller pieces and store them in different server

(datanode). The key used to determine what datanode each row goes is called “distribution key”.

Then updates by multiple transactions can be distributed among datanodes and they can be done in parallel. With more datanode, we can run more updates to fact tables in parallel. This is basically the background that Postgres-XC provides write scalability. Figure 8 illustrates this.

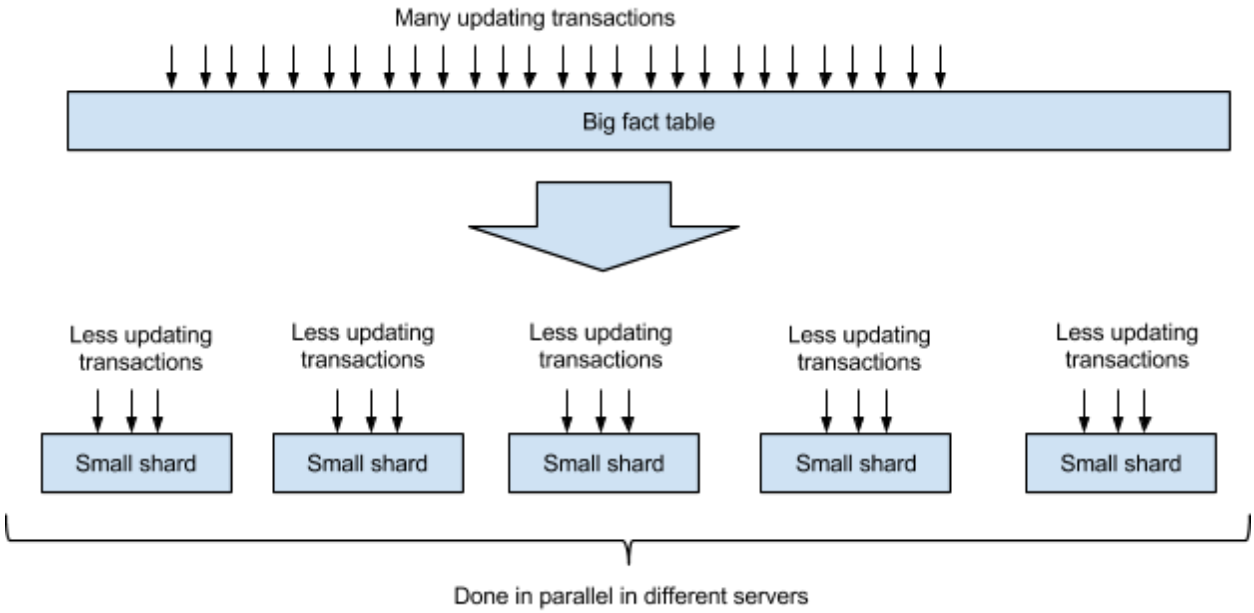

Figure 8 Write scalability in Postgres-XC

We replicate all the dimention tables to all the datanodes. Because most of the joins are done between a fact table and dimention tables, or among fact tables with distribution key involved, we can perform big join as a union of joins between each shard and replicated tables locally in each datanode in parallel. This is how Postgres-XC provides read scalability.

If a statement has additional predicates in WHERE clause to locate a datanode where the target rows are stored, and most of OLTP queries are, then Postgres-XC can select only a few of datanode to perform such a query.

Figure 9 and Figure 10 illustrate this.

Please note that updating dimention tables does not scale. Each replica of a dimention table (replicated table, in XC) has to be updated separately. Although each separate update statement are performed in parallel, we should not expect write scalability in this case.

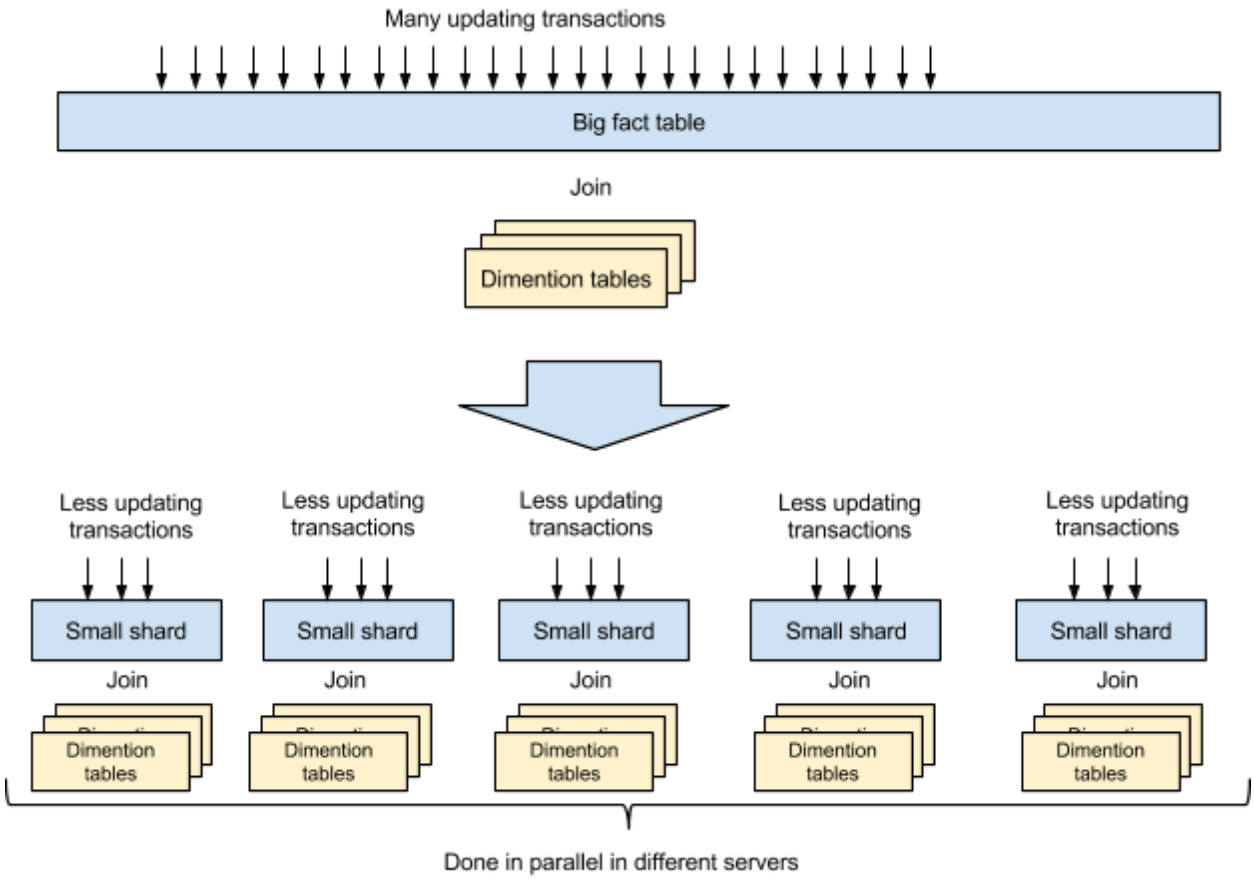

Figure 9 Decompose big statement into smaller shards.

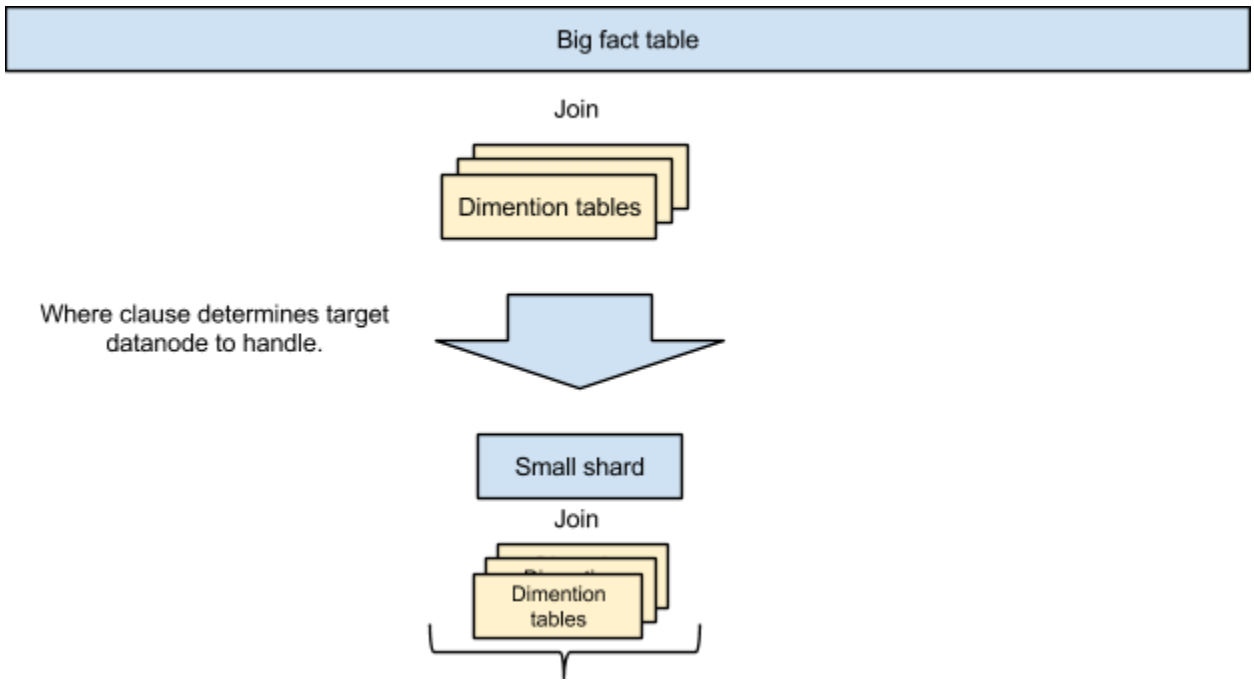

Figure 10 Statement can be optimized more if WHERE clause determines the target

There could be exceptional case where an application needs a join between fact tables without distribution key involved. In this case, Postgres-XC pushes down as many

operation as possible to each datanode but does final join operation at the top level (coordinator).

In other words, if an application cannot utillize this start schema, you should be very careful to design the table distribution to use Postgres-XC’s distributed query processing.