The Internals of PostgreSQL

for database administrators and system developers

Chapter 2

Process and Memory Architecture

In this chapter, the process architecture and memory architecture in PostgreSQL are summarized to help to read the subsequent chapters. If you are already familiar with them, you may skip over this chapter.

2.1. Process Architecture

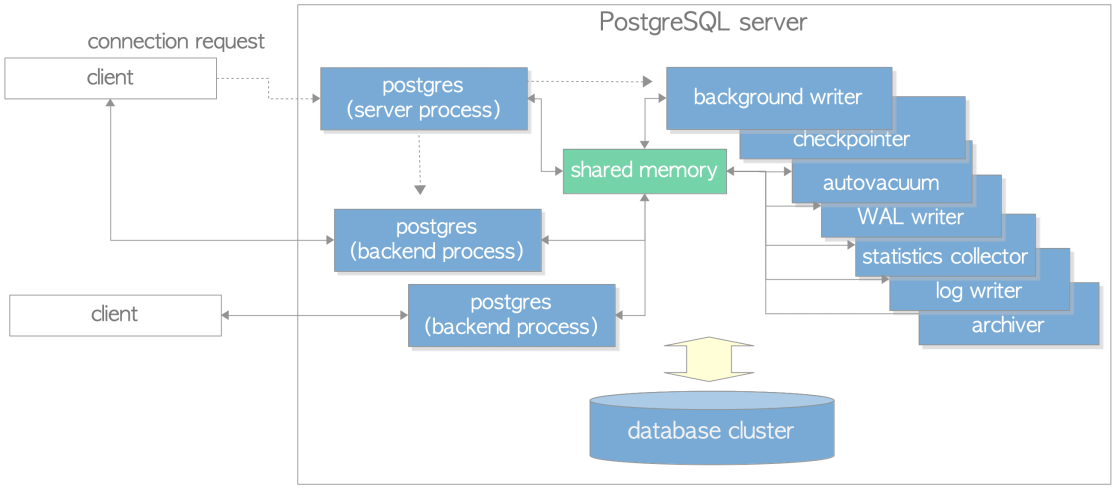
PostgreSQL is a client/server type relational database management system with the multi-process architecture and runs on a single host.

A collection of multiple processes cooperatively managing one database cluster is usually referred to as a *'PostgreSQL server'*, and it contains the following types of processes:

- A **postgres server process** is a parent of all processes related to a database cluster management.
- Each **backend process** handles all queries and statements issued by a connected client.
- Various **background processes** perform processes of each feature (e.g., VACUUM and CHECKPOINT processes) for database management.
- In the **replication associated processes**, they perform the streaming replication. The details are described in Chapter 11.
- In the **background worker process** supported from version 9.3, it can perform any processing implemented by users. As not going into detail here, refer to the official document.

In the following subsections, the details of the first three types of processes are described.

Fig. 2.1. An example of the process architecture in PostgreSQL.



This figure shows processes of a PostgreSQL server: a postgres server process, two backend processes, seven background processes, and two client processes. The database cluster, the shared memory, and two client processes are also illustrated.

2.1.1. Postgres Server Process

As already described above, a *postgres server process* is a parent of all in a PostgreSQL server. In the

earlier versions, it was called 'postmaster'.

By executing the pg_ctl utility with *start* option, a postgres server process starts up. Then, it allocates a shared memory area in memory, starts various background processes, starts replication associated processes and background worker processes if necessary, and waits for connection requests from clients. Whenever receiving a connection request from a client, it starts a backend process. (And then, the started backend process handles all queries issued by the connected client.)

A postgres server process listens to one network port, the default port is 5432. Although more than one PostgreSQL server can be run on the same host, each server should be set to listen to different port number in each other, e.g., 5432, 5433, etc.

2.1.2. Backend Processes

A backend process, which is also called *postgres*, is started by the postgres server process and handles all queries issued by one connected client. It communicates with the client by a single TCP connection, and terminates when the client gets disconnected.

As it is allowed to operate only one database, you have to specify a database you want to use explicitly when connecting to a PostgreSQL server.

PostgreSQL allows multiple clients to connect simultaneously; the configuration parameter *max_connections* controls the maximum number of the clients (default is 100).

If many clients such as WEB applications frequently repeat the connection and disconnection with a PostgreSQL server, it increases both costs of establishing connections and of creating backend processes because PostgreSQL has not implemented a native connection pooling feature. Such circumstance has a negative effect on the performance of database server. To deal with such a case, a pooling middleware (either pgbouncer or pgpool-II) is usually used.

2.1.3. Background Processes

Table 2.1 shows a list of background processes. In contrast to the postgres server and the backend process, it is impossible to explain each of the functions simply, because these functions depend on the individual specific features and PostgreSQL internals. Thus, in this chapter, only introductions

are made. Details will be described in the following chapters.

Table 2.1: background processes.

process	description	reference
background writer	In this process, dirty pages on the shared buffer pool are written to a persistent storage (e.g., HDD, SSD) on a regular basis gradually. (In version 9.1 or earlier, it was also responsible for checkpoint process.)	Section 8.6
checkpointer	In this process in version 9.2 or later, checkpoint process is performed.	Section 8.6, Section 9.7
autovacuum launcher	The autovacuum-worker processes are invoked for vacuum process periodically. (More precisely, it requests to create the autovacuum workers to the postgres server.)	Section 6.5
WAL writer	This process writes and flushes periodically the WAL data on the WAL buffer to persistent storage.	Section 9.9
statistics collector	In this process, statistics information such as for	

	pg_stat_activity and for pg_stat_database, etc. is collected.	
logging collector (logger)	This process writes error messages into log files.	
archiver	In this process, archiving logging is executed.	Section 9.10

The actual processes of a PostgreSQL server is shown here. In the following example, one postgres server process (pid is 9687), two backend processes (pids are 9697 and 9717) and the several background processes listed in Table 2.1 are running. See also Fig. 2.1.

```
postgres> pstree -p 9687
-+= 00001 root /sbin/launchd
 \-+- 09687 postgres /usr/local/pgsql/bin/postg
res -D /usr/local/pgsql/data
   |--= 09688 postgres postgres: logger proces
s
   |--= 09690 postgres postgres: checkpointer p
rocess
   |--= 09691 postgres postgres: writer process
   |--= 09692 postgres postgres: wal writer pro
cess
   |--= 09693 postgres postgres: autovacuum la
uncher process
```

```
  |--= 09694 postgres postgres: archiver proce
ss
  |--= 09695 postgres postgres: stats collector
 process
  |--= 09697 postgres postgres: postgres sam
pledb 192.168.1.100(54924) idle
  \-= 09717 postgres postgres: postgres sam
pledb 192.168.1.100(54964) idle in transaction
```

2.2. Memory Architecture

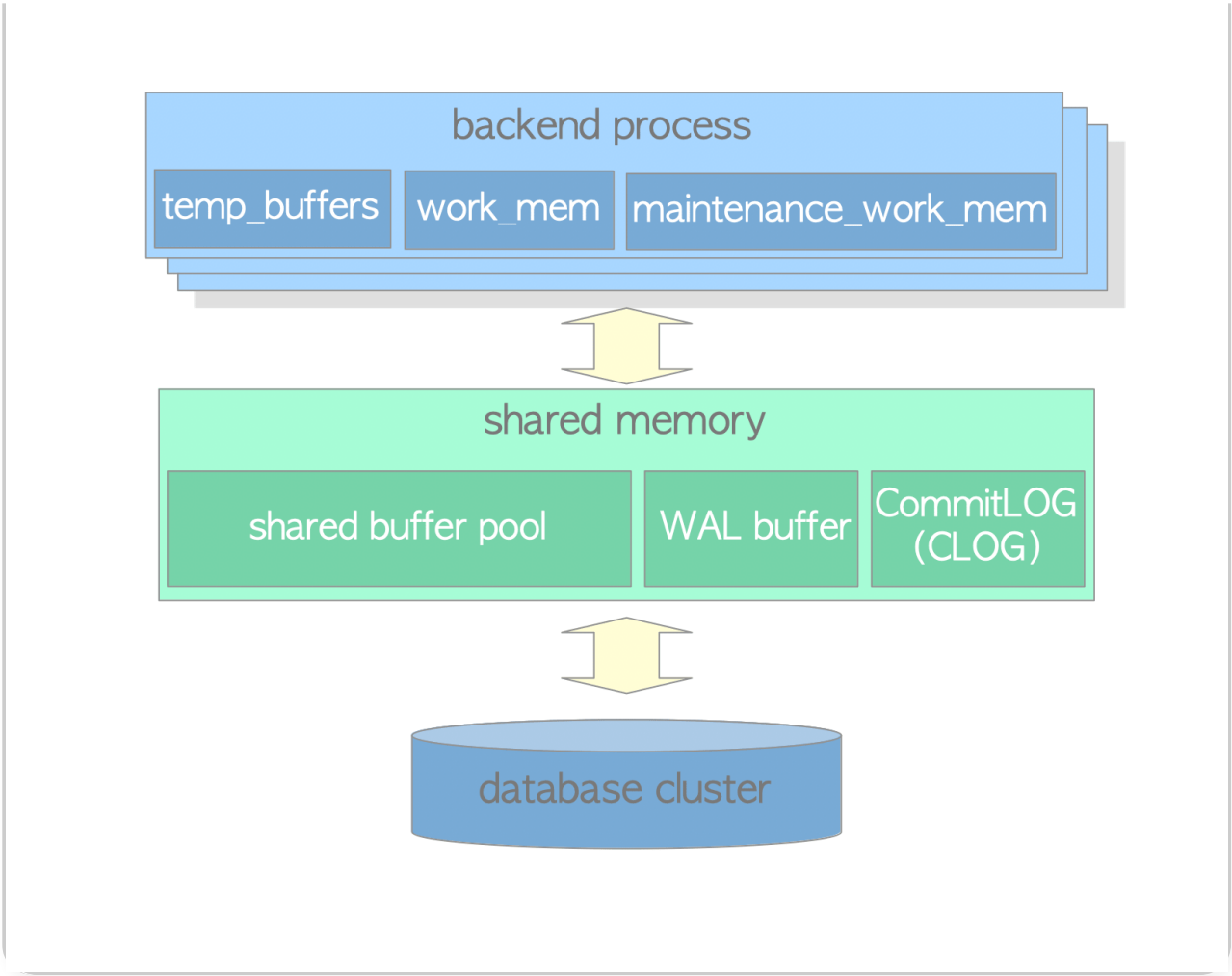
Memory architecture in PostgreSQL can be classified into two broad categories:

- Local memory area – allocated by each backend process for its own use.

  Shared memory area – used by all processes of a PostgreSQL server.

In the following subsections, those are briefly descibed.

Fig. 2.2. Memory architecture in PostgreSQL.



2.2.1. Local Memory Area

Each backend process allocates a local memory area for query processing; each area is divided into several sub-areas – whose sizes are either fixed or variable. Table 2.2 shows a list of the major sub-areas. The details will be described in the following chapters.

Table 2.2: Local memory area

sub-area	description	reference
work_mem	Executor uses this	Chapter 3

	area for sorting tuples by ORDER BY and DISTINCT operations, and for joining tables by merge-join and hash-join operations.	
maintenance_work_mem	Some kinds of maintenance operations (e.g., VACUUM, REINDEX) use this area.	Section 6.1
temp_buffers	Executor uses this area for storing temporary tables.	

2.2.2. Shared Memory Area

A shared memory area is allocated by a PostgreSQL server when it starts up. This area is also divided into several fix sized sub-areas. Table 2.3 shows a list of the major sub-areas. The details will be described in the following chapters.

Table 2.3: Shared memory area

sub-area	description	reference
shared buffer pool	PostgreSQL loads pages within tables and indexes from a persistent storage to here, and operates them directly.	Chapter 8
WAL buffer	To ensure that no data has been lost by server failures, PostgreSQL supports the WAL mechanism. WAL data (also referred to as XLOG records) are transaction log in PostgreSQL; and WAL buffer is a buffering area of the WAL data before writing to a persistent storage.	Chapter 9
commit log	Commit Log(CLOG) keeps the states of all transactions (e.g., in_progress,committed,aborted) for Concurrency Control (CC) mechanism.	Section 5.4

In addition to them, PostgreSQL allocates several areas as shown below:

- Sub-areas for the various access control mechanisms. (e.g., semaphores, lightweight locks, shared and exclusive locks, etc)
- Sub-areas for the various background processes, such as checkpointer and autovacuum.

- Sub-areas for transaction processing such as save-point and two-phase-commit.

and others.