

Procedure for Evaluating Backup and Restoration Performance of PostgreSQL 8.1.4 by Using OSDL DBT-1

This document describes the procedure for evaluating the backup and restoration performance of PostgreSQL 8.1.4 by using DBT-1.

1. Overview

1.1 Environment Definition

Table 1 defines the environment for measurement.

Table 1 Measurement Environment

Hardware	HITACHI HA8000/110W HB
CPU	Intel Xeon HT OFF
Main memory	2GB
Hard disk	220GB
OS	RHEL4 2.6.9-11.Elsmpt
Software to be evaluated	PostgreSQL 8.1.4
Load tool	OSDL DBT-1 2.1

This measurement environment includes three hard disks, /dev/sda, /dev/sdb and /dev/sdc, as shown in Figure 1.

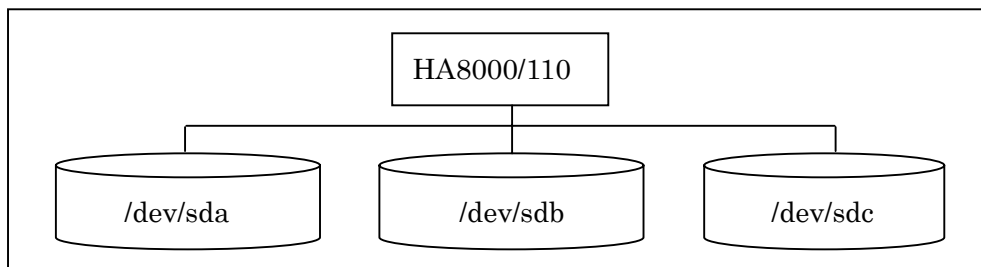


Figure 1 Disk Arrangement

Table 2 shows the purposes of these hard disks.

Table 2 Purposes of the Hard Disks in the Measurement Environment

Device name	Purpose	Mount point
/dev/sda	To store the OS and execution log.	/
/dev/sdb	To store the database to be backed up after DBT-1 execution	/data
/dev/sdc	To store backup file WAL	/data2

1.2 Databases Used

DBT-1 offers commands, datagen and build_db.sh, for database creation. We use these commands. Table 3 shows the arguments given to datagen.

Table 3 datagen Commands and Options for Measurement

Command and option	Database size determined by build_db.sh
datagen -d PGSQL -i 10000 -u 1000	6,727,304 KB (about 7 GB)
datagen -d PGSQL -i 10000 -u 8000	52,974,116 KB (about 53 GB)

We use the two databases shown in Table 3 for measurement.

1.3 Processing Flow for Measurement of Backup and Restoration Performance

There are two types of backup with PostgreSQL. One is off-line backup which is performed while no transactions are present. The other is on-line backup which is performed in the background during a transaction. A different restoration method is available for each type of backup.

Table 4 Backup and Restoration Methods

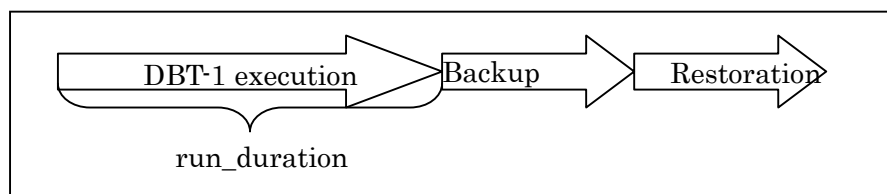
Backup method	Execution method and functionality	Execution during DBT-1 run	Restoration method and functionality
pg_dump	<p>pg_dump</p> <p>Backs up a database. One of output formats can be selected for the backup file depending on the option.</p> <p>pg_dump -Fp[Target DB name] > [Output filename]:</p>	Possible	<p>Restore only the valid area for each database regardless of whether the format is script or archive.</p> <p>To restore a script file, specify:</p> <p>psql -f [Backup filename][Target database name]</p>

	Dumps the database to the file in script format. pg_dump -Fc[Target DB name] > [Output filename]: Dumps the database to the file in custom archive format.		To restore an archive file, specify: pg_restore -d [Target database name] [Backup filename]
pg_dumpall	Dumps all the databases located in a cluster to one backup file in script format. Thus, Templates 0 and 1 as well as Postgres databases can be restored. This permits backing up information about the postgres user. To execute the backup, specify: pg_dumpall > [Output filename]	Impossible	Restore only the valid area in the databases. To execute the restoration, specify: psql -f [Backup filename] [Target database name]
tar czf	OS tar command which copies the entire cluster for backup. The z option is available to compress the files with gzip at the time of copying. If PITR is not specified, the postmaster should be stopped. To execute the backup, specify: tar czf [Output filename] [Data cluster directory name]	Possible only when PITR is specified	Restore also the invalid area because the file system is backed up. To execute the restoration, specify: tar xzf [Backup filename]

Backup with pg_dump or pg_dumpall involves issuing an SQL statement. This backup is impossible when PostgreSQL is inactive. Therefore, offline backup in the proper sense of the term cannot be executed with pg_dump or pg_dumpall. In this document, backup when no transactions are present is called offline backup, and backup during a transaction is called online backup.

●Offline Backup

We measure the performance for offline backup by using pg_dump -Fp, pg_dump -Fc, pg_dumpall and tar czf. We execute VACUUM FULL right before the backup with each command. We also measure the performance for backup when the invalid area in the data cluster to be backed up is eliminated from the file system. Figure 2 shows the processing flow for measurement without execution of VACUUM FULL. Figure 3 shows the processing flow for measurement with execution of VACUUM FULL.



finally to 50400 (10 + 4 hours). A data cluster of about 53 GB is generated with “datagen -d PGSQL -I 10000 -u 8000.” Online backup of this cluster took a little more than 2 hours with `pg_dump -Fp` and a little less than 4 hours with `tar czf`. Thus, the run duration after the start of backup is set to 2.5 hours or 4 hours because it should be longer than either of the online backup periods.

After the termination of DBT-1, the backup file created online is restored with `tar xzf`. Then, PITR is performed. PITR is not possible with `pg_dump -Fp`, and thus the processing with this command ends with the restoration shown in Figure 4.

2. Environment Setup

2.1 Installing Linux

Default installation of the distribution is selected. For this installation, follow the instructions in the installation manual for the distribution.

2.2 Environment Setup for DBT-1

In DBT-1, a socket is opened for each thread. Thus, if the upper limit on the number of files which can be opened simultaneously within one process is too low, database connection is disabled even though threads are generated. There is a need to raise the upper limit by adding definitions to the configuration file (`/etc/security/limits.conf`).

```
/etc/security/limits.conf
```

```
* soft nofile 4096  
* hard nofile 65536
```

*If “nofile” is not specified, add the two lines shown above. If it is specified, correct the values.

2.3 Installing PostgreSQL and Its Associated Driver

●Installing PostgreSQL

We install PostgreSQL from source. The procedure for the installation is not much

different from the ordinary installation procedure. For measurement, the name of a PostgreSQL super user is specified.

PostgreSQL to be used is version 8.1.4. Obtain the appropriate tarball file from the archive site for PostgreSQL.

Source URL: <http://www.postgresql.org/>

As the root user, create a pgsq user and pgsq group (Linux users who are to own PostgreSQL). Also, create a directory in which the PostgreSQL source code archive is to be expanded and specify the owner of the directory as the posql user.

```
# su - root
# groupadd pgsq
# useradd pgsq -g pgsq
# mkdir /usr/local/src/postgresql-8.1.4
# chown pgsq /usr/local/src/postgresql-8.1.4
```

As the pgsq user, expand the source code archive for PostgreSQL and move into the directory where the expanded PostgreSQL resides. Then, start installation. It is assumed that the source code archive for PostgreSQL is located in the /tmp directory.

```
# su - pgsq
$ cd /usr/local/src
$ tar xzvf /tmp/postgresql-8.1.4.tar.gz
$ cd ./postgresql-8.1.4
$ ./configure --prefix=/usr/local/pgsq
$ make all
$ su root
# make install
```

● Installing iODBC

To run DBT-1, we have to install the iODBC driver manager. To use applications with the ODBC driver on Linux, we have to install the ODBC driver manager and PostgreSQL ODBC driver which is a lower-level data access driver. We use iODBC as the ODBC driver manager.

Source URL: <http://www.iodbc.org/>

```
$ su - root
# mkdir /usr/local/src/libiodbc-3.51.2
# chown pgsq1 /usr/local/src/libiodbc-3.51.2
# su - pgsq1
$ cd /usr/local/src
$ tar xzf /tmp/libiodbc-3.51.2.tar.gz
$ cd libiodbc-3.51.2
$ ./configure
$ make
$ su root
# make install
```

●Installing the psqldb Driver

tarball for the psqldb driver should be downloaded to the /tmp directory. This driver is available from the site below.

Source URL: <http://pgfoundry.org/projects/psqldb/>

```
# su - root
# mkdir /usr/local/src/psqldb-08.00.0102
# chown pgsq1 /usr/local/src/psqldb-08.00.0102
# su - pgsq1
$ cd /usr/local/src
$ tar xzvf /tmp/psqldb-08.00.0102.tar.gz
$ cd psqldb-08.00.0102
$ ./configure
$ make
$ su root
# make install
```

To connect DBT-1 via ODBC to PostgreSQL, create an “.odbc.ini” file (which is a hidden file) in the pgsq1 user’s directory. This file is shown below.

```
[ODBC Data Sources]
DBT1=Postgres DBT1
[DBT1]
Servername=localhost
```

```
Database=DBT1
Driver=/usr/local/lib/psqlodbc.so
Port=5432
ReadOnly=No
```

2.4 Installing DBT-1

•Environment Setup for DBT-1

Capturing a log of DBT-1 activity involves using a command which requires root privilege. Thus, it is necessary to permit the pgsql user to utilize sudo. For this purpose, add the following line to the `/etc/sudoers` file as the root user:

```
pgsql ALL=(ALL) NOPASSWD: ALL
```

Also, specify the following in `.bashrc` of the pgsql user.

```
export PGDIR=/usr/local/pgsql
export PATH=$PATH:$PGDIR:$PGDIR/bin
export PGUSER=pgsql
export PGDATA=/data/pgsql/data
export SID1=DBT1
export LD_LIBRARY_PATH=/usr/local/lib:/usr/local/pgsql/lib
```

The PGDATA value is a variable indicating the location of the PostgreSQL data cluster. For measurement, the database shown in Table 3 should be created on the `/dev/sdb` disk. Thus, the directories at the `/data` level and below are specified.

•Installing DBT-1

As the pgsql user, install DBT-1 in the pgsql home directory.

First, download the source to the pgsql user's home directory. Then, expand tarball for the DBT-1 source with the following command:

```
$ tar xzvf dbt1-v2.1-PostgreSQL-ODBC-1.2.1.tar.gz
```

Next, compile the expanded source.

```
$ cd ~/dbt1-v2.1
```

```
$ ./configure --with-postgresql-odbc --without-sapdb
$ make
$ make install
```

3. Measurement Procedure

●Creating a Database for Measurement

Specify the directory which should contain data generated by DBT-1.

```
$ mkdir ~/data
$ chmod a+w ~/data
```

Create test-purpose data for DBT-1 by using the following commands:

```
$ cd ~/dbt1-v2.1/datagen
$ ./datagen -d PGSQL -i 10000 -u (1000or8000) -p /home/pgsql/data
```

We conduct measurement while changing the database size as described in Section 1.2. To change it, we modify the argument to the `-u` option for the `datagen` command. The other options are shown below.

- d: Database type (SAPDB or PGSQL)
- i: Total number of items to be generated
- u: Number of virtual users to be generated
- p: Actual data directory to be created

Load the created test-purpose data to the database for PostgreSQL.

```
# su - pgsq
$ mkdir -p /data/pgsql/data
$ initdb --no-locale --encoding =EUC_JP
$ cd /data
$ ./build_db.sh '-c tcp_socket=on ' 0 0
```

*Be sure to insert a blank space before and after the single quotation mark (') for execution.

This command creates a DBT1 database.

●Setting the DBT-1 Execution Parameters

The DBT-1 execution parameters are defined in `~/dbt1-v2.1/scripts/stats/dbt1.config`. For measurement, we change only the DBT-1 parameters listed in Table 5 below.

Table 5 DBT-1 Parameters Set for Measurement

Item	Settings
<code>#customers</code>	2880000 when 1000 is specified for <code>-u</code> for data generation with <code>datagen</code> . 23040000 when 8000 is specified for that option.
<code>#eu</code>	1000
<code>#run_duration</code> in seconds	10800 to 50400 depending on the measurement conditions

- Setting the PostgreSQL Execution Parameters

The parameters for PostgreSQL are defined in `$PGDATA/postgresql.conf`. `$PGDATA` is defined in `.bashrc` described in Section 2.4. We set the parameters as shown in Table 6 for measurement.

Table 6 PostgreSQL Parameters Set for Measurement

Item	Settings
<code>redirect_stderr</code>	on
<code>log_directory</code>	<code>pg_log</code>
<code>log_filename</code>	<code>postgresql-%Y-%m-%d_%H%M%S.log</code>
<code>log_truncate_on_rotation</code>	off
<code>log_rotation_age</code>	1440
<code>log_rotation_size</code>	10240
<code>log_line_prefix</code>	<code><%t></code>
<code>max_fsm_pages</code>	200000
<code>max_fsm_relations</code>	10000

To enable PITR, we should set the parameter shown in Table 7 in addition to those shown above.

Table 7 Parameter for PITR

Item	Settings
------	----------

archive_command	'cp -i %p /data2 /WAL/%f </dev/null'
-----------------	--------------------------------------

The archive_command parameter specifies the command for output to the WAL log. If this parameter is used, a directory (/data2/WAL in this example) for containing a copy should be created before the start of PostgreSQL. For measurement, we send output to a directory below /data2 on which the /dev/sdc disk is mounted.

After editing postgresql.conf, restart PostgreSQL with the following commands:

```
$ pg_ctl stop
$ pg_ctl start
```

- Installing pgstattuple

To investigate the physical table size and valid and invalid areas for PostgreSQL, install pgstattuple from within the contrib module for PostgreSQL. To do this, execute the following as the root user:

```
$ su - root
# cd /usr/local/src/postgresql-8.1.4/contrib/pgstattuple
# make
# make install
# exit
$ cd /usr/local/pgsql/share/contrib
$ psql -f pgstattuple.sql DBT1
```

Then, execute the command below during the start of PostgreSQL. This enables checking the status of each table.

```
$ psql -x -c "select * from pgstattuple('[テーブル名]');" DBT1
```

For [Table name], enter the name of a table which is in the DBT1 database. This database includes the following tables:

- address
- addrid
- author
- cc_xacts
- country

- custid
- customer
- item
- order_line
- orders
- scid
- shopping_cart
- shopping_cart_line

●Starting DBT-1

Execute DBT-1 by using the following commands:

```
$ cd ~/dbt1-v2.1/scripts/stats  
$ ./run_dbt1.sh /home/pgsql/[directory]
```

The directory specified in `run_dbt1.sh` is automatically created. When DBT-1 terminates, a file called “BT” is created immediately below the specified directory. There is an item called “total errors” at the end of the line in this BT file. If the value of this item is “0,” DBT-1 processing was successfully performed.

●VACUUM

When DBT-1 execution updates the database, an invalid data area occurs in PostgreSQL. Our measurement involves determining the performance for backup without an invalid area. Thus, delete the invalid area by executing `VACUUM FULL` right before the backup processing. The command used is as follows.

```
$ psql -c "vacuum full VERBOSE;" DBT1
```

This command executes `VACUUM FULL` for the entire DBT-1 database. The `VERBOSE` option in the `-c` option displays details about the `VACUUM` execution.

●Backup and Restoration

After the DBT-1 processing, measure the performance for backup and restoration. We use the `time` command for measuring the processing time. The `time` command executes

a command by giving an argument to the command. Right after the command execution, the time command displays the time taken for that execution.

Table 8 shows the methods for offline backup and their associated restore commands.

Table 8 Commands for Backup and Restoration

Backup method	Backup command	Restore command
pg_dumpall	pg_dumpall > [Backup filename]	psql -f [Backup filename] DBT1
pg_dump -F p	pg_dump -F p DBT1 > [Backup filename]	psql -f [Backup filename] DBT1
pg_dump -F c	pg_dump -F c DBT1 > [Backup filename]	pg_restore -d DBT1 [Backup filename]
tar czf	tar czf [Backup filename] ./data	tar xzf [Backup filename]

*pg_dump methods delete the DBT-1 directory between the backup and restore time, and redefine an empty DBT-1 directory.

*tar czf stops PostgreSQL before backup and issues a command via the directory located one level above \$PGDATA. tar czf also deletes \$PGDATA before restoration.

For measurement, the backup file is output to the /dev/sdc disk.

The procedure for online backup using pg_dump -Fp is identical to that for offline backup using the same method. However, tar czf requires PITR settings described in XXX.

To perform backup for PITR with tar czf, follow the procedure below.

```
$ cd $PGDATA/..
$ tar czf [Backup filename] ./data
$ psql -c "SELECT pg_stop_backup();" DBT1
```

After the completion of backup and termination of DBT-1, stop PostgreSQL and perform restoration. To perform it, follow the procedure below.

```
$ cd $PGDATA
$ mv pg_xlog [Arbitrary directory]
$ cd $PGDATA/..
$ rm -rf ./data
```

```
$ tar xzf [Backup filename]
```

The `pg_xlog` directory still contains a WAL log which has not been archived with the command specified in `archive_command`. Thus, it is necessary to save the WAL log before deleting `$PGDATA` for restoration.

Lastly, we need to perform PITR. To do this, place `recovery.conf` in `$PGDATA` while PostgreSQL is inactive. There is a `recovery.conf.sample` file in `/usr/local/pgsql/share/recovery.conf.sample`. Copy this file to `$PGDATA` and rename the copy. Then, open the copy with an editor and correct the line where `restore_command` is defined to the following:

```
restore_command = 'cp /data2/ WAL/%f %p'
```

Move the WAL log, which was saved immediately before restoration, back to `$PGDATA` and start PostgreSQL. PITR recovery will start.

```
$ mv [Saved pg_xlog] $PGDATA  
$ pg_ctl start
```

Check if “<2006-11-25 00:24:14 JST>LOG: archive recovery complete“ appears in the PostgreSQL execution log (below `$PGDATA/pg_log`). If so, PITR successfully terminated.