

Author : Chris Drawater
Date : May 2005
Version : 1.2

PostgreSQL 8.0.02 for J2EE applications on Solaris 10

Abstract

*Advance planning enables PostgreSQL 8 and its associated JDBC driver to be quickly deployed in a basic but resilient and IO efficient manner.
Minimal change is required to switch JDBC applications from Oracle to PostgreSQL.*

Document Status

This document is Copyright © 2005 by Chris Drawater.

This document is freely distributable under the license terms of the [GNU Free Documentation License](http://www.gnu.org/copyleft/fdl.html) (<http://www.gnu.org/copyleft/fdl.html>). It is provided for educational purposes only and is NOT supported.

Introduction

This paper documents how to deploy PostgreSQL 8 and its associated JDBC driver in a basic but both resilient and IO efficient manner. Guidance for switching from Oracle to PostgreSQL is also provided.

It is based upon experience with the following configurations =>

- PostgreSQL 8.0.2 on Solaris 10
- PostgreSQL JDBC driver on Windows 2000

using the PostgreSQL distributions =>

- postgresql-base-8.0.2.tar.gz*
- postgresql-8.0-311.jdbc3.jar*

Background for Oracle DBAs

For DBAs coming from an Oracle background, PostgreSQL has a number of familiar concepts including

- Checkpoints
- Tablespaces
- MVCC concurrency model
- Write ahead log (WAL)+ PITR
- Background DB writer
- Statistics based optimizer
- Recovery = Backup + archived WALs + current WALs

However, whereas 1 Oracle instance (set of processes) services 1 physical database, PostgreSQL differs in that

- 1 PostgreSQL “cluster” services n * physical DBs
- 1 cluster has tablespaces (accessible to all DBs)
- 1 cluster = 1 PostgreSQL instance = set of server processes etc (for all DBs) + 1 tuning config + 1 WAL
- User accts are cluster wide by default
- There is no undo or BI file – so to support MVCC, the “consistent read” data is held in the tables themselves and once obsolete needs to be cleansed out using the ‘vacuum’ utility.

The basic PostgreSQL deployment guidelines for Oracle aware DBAs are to =>

- Create only 1 DB per cluster
- Have 1 superuser per cluster
- Let only the superuser create the database
- Have one user to create/own the DB objects + n* endusers with appropriate read/write access
- Use only ANSI SQL datatypes and DDL.
- Wherever possible avoid DB specific SQL extensions to ensure cross-database portability

IO distribution & disc layouts

It is far better to start out with good disc layouts rather than reto-fix for a production database.

As with any DBMS, for resilience, the recovery components (eg. backups , WAL, archived WAL logs) should be kept on devices separate from the actual data.

So the basic rules for resilience are as follows.

For non disc array or JBOD systems =>

- keep recovery components separate from data on dedicated discs etc
- keep WAL and data on separate disc controllers
- mirror WAL across discs (preferably across controllers) for protection against WAL spindle loss

For SAN based disc arrays (eg HP XP12000) =>

- keep recovery components separate from data on dedicated LUNs etc
- use Host Adapter Multipathing drivers (such as *mpxio*) with 2 or more HBAs for access to SAN .

Deploy application data on mirrored/striped (ie RAID 1+0) or write-cache fronted RAID 5 storage. The WAL log IO should be configured to be *osync* for resilience (see basic tuning in later section).

Ensure that every PostgreSQL component on disc is resilient (duplexed) !

Recovery can be very stressful...

Moving onto IO performance, it is worth noting that WAL IO and general data IO access have different IO characteristics.

WAL → sequential access (write mostly)
Data → sequential scan, random access write/read

The basic rules for good IO performance

use tablespaces to distribute data and thus IO across spindles or disc array LUNs
keep WAL on dedicated spindles/LUNs (mirror/striped in preference to RAID 5)
keep WAL and arch WAL on separate spindles to reduce IO on WAL spindles.

RAID or stripe data across discs/LUNs in 1 Mb chunks/units if unsure as what chunk size to use.

For manageability, keep the software distr and binaries separate from the database objects.

Likewise, keep the system catalogs and non-application data separate from the application specific data.

5 distinct storage requirements can be identified =>

Software tree (Binaries, Source, distr)
Shared PG sys data
WAL logs
Arch WAL logs
Application data

For the purposes of this document, the following minimal set of FS are suggested =>

```
/opt/postgresql/8.0.2           # default 4 Gb for software tree  
/var/opt/postgresql         # default 100 Mb  
/var/opt/postgresql/CLUST/sys # default size 1Gb for shared sys data  
/var/opt/postgresql/CLUST/wal # WAL location # mirrored/striped  
/var/opt/postgresql/CLUST/archwal # archived WALs  
/var/opt/postgresql/CLUST/data # application data + DB sys catalogs # RAID 5
```

where CLUST is your chosen name for the Postgres DB cluster

For enhanced IO distribution, a number of *.../data* FS (eg *data01*, *data02* etc) could be deployed.

Pre-requisites !

The GNU compiler and make software utilities (available on the Solaris 10 installation CDs) =>

```
gcc (compiler) ($ gcc --version => 3.4.3 )  
gmake (GNU make)
```

are required and should be found in

```
/usr/sfw/bin
```

Create the Unix acct

```
postgres  
in group dba  
with a home directory of say /export/home/postgresql  
using
```

```
$ useradd utility
```

or hack

```
/etc/group then /etc/passwd then run pwconv and then passwd postgres
```

Assuming the following FS have been created =>

```

/opt/postgresql/8.0.2      # default 4 Gb for the PostgreSQL software tree
/var/opt/postgresql      # default 100 Mb

create directories
/opt/postgresql/8.0.2/source # source code
/opt/postgresql/8.0.2/distr  # downloaded distribution
all owned by user postgres:dba with 700 permissions

```

To ensure, there are enough IPC resources to use PostgreSQL, edit */etc/system* and add the following lines =>

```

set shmsys:shminfo_shmmax=130000000
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=200
set shmsys:shminfo_shmseg=20
set semsys:seminfo_semmns=800
set semsys:seminfo_semmni=70
set semsys:seminfo_semmsl=270 # defaults to 25

set rlim_fd_cur=1024 # per process file descriptor soft limit
set rlim_fd_max=4096 # per process file descriptor hard limit

```

Then on the console (log in as root) =>

```

$ init 0
{a} ok boot -r

```

Download Source

Download the source codes from <http://www.postgresql.org> (and if downloaded via Windows, remember to ftp in binary mode) =>

```

Distributions often available include =>
postgresql-XXX.tar.gz => full source distribution.
postgresql-base-XXX.tar.gz => Server and the essential client interfaces
postgresql-opt-XXX.tar.gz => C++, JDBC, ODBC, Perl, Python, and Tcl interfaces, as well as multibyte support
postgresql-docs-XXX.tar.gz => html docs
postgresql-test-XXX.tar.gz => regression test

```

For a working, basic PostgreSQL installation supporting JDBC applications, simply use the 'base' distribution.

Create Binaries

Unpack Source =>

```

$ cd /opt/postgresql/8.0.2/distr
$ gunzip postgresql-base-8.0.2.tar.gz

$ cd /opt/postgresql/8.0.2/source
$ tar -xvof /opt/postgresql/8.0.2/distr/postgresql-base-8.0.2.tar

```

Set Unix environment =>

```
TMPDIR=/tmp
PATH=/usr/bin:/usr/ucb:/etc/./usr/sfw/bin:/usr/local/bin:/usr/ccs/bin:$PATH
export PATH TMPDIR
```

Configure the build options =>

```
$ cd /opt/postgresql/8.0.2/source/postgresql-8.0.2
$ ./configure --prefix=/opt/postgresql/8.0.2 --with-pgport=5432 --without-readline
CC=/usr/sfw/bin/gcc
```

Note => *--enable-thread-safety option failed*

And build =>

```
$ make
$ make install
```

On an Ultra 5 workstation, this gives 32 bit executables

Setup Unix environment

Add to environment =>

```
LD_LIBRARY_PATH=/opt/postgresql/8.0.2/lib
PATH=/opt/postgresql/8.0.2/bin:$PATH
export PATH LD_LIBRARY_PATH
```

Create Database(Catalog) Cluster

Add to Unix environment =>

```
PGDATA=/var/opt/postgresql/CLUST/sys          # PG sys data , used by all DBs
export PGDATA
```

Assuming the following FS has been created =>

```
/var/opt/postgresql/CLUST/sys          # default size 1Gb
```

where CLUST is your chosen name for the Postgres DB cluster,
initialize database storage area, create shared catalogs and template database *template1* =>

```
$ initdb -E UNICODE -A password          -W
# DBs have default Unicode char set, user basic passwords, prompt for super user password
```

Startup, Shutdown and basic tuning of servers

Check servers start/shutdown =>

```
$ pg_ctl start -l /tmp/logfile
$ pg_ctl stop
```

Next, tune the PostgreSQL instance by editing the configuration file *\$PGDATA/postgresql.conf*.

First take a safety copy =>

```
$ cd $PGDATA
$ cp postgresql.conf postgresql.conf.orig
```

then make the following (or similar changes) to *postgresql.conf* =>

```
# listener
listen_addresses = 'localhost'
port = 5432

# data buffer cache
shared_buffers = 10000          # each 8Kb so depends upon memory available

# checkpoints
checkpoint_segments = 3        # default
checkpoint_timeout = 300       # default
checkpoint_warning = 30        # default – logs warning if ckpt interval < 30s

# log related
fsync = true                   # resilience
wal_sync_method = open_sync    # resilience
commit_delay = 10              # group commit if works
archive_command = 'cp "%p" /var/opt/postgresql/CLUST/archwal/"%f"'

# server error log
log_line_prefix = '%t : '      # timestamp
log_min_duration_statement = 1000 # log any SQL taking more than 1000ms
log_min_messages = info

#transaction/locks
default_transaction_isolation = 'read committed'
```

Restart the servers =>

```
$ pg_ctl start -l /tmp/logfile
```

Create the Database

This requires the FS =>

```
/var/opt/postgresql/CLUST/wal          # WAL location
/var/opt/postgresql/CLUST/archwal      # archived WALs
/var/opt/postgresql/CLUST/data         # application data + DB sys catalogs
```

plus maybe also =>

```
/var/opt/postgresql/CLUST/backup      # optional for data and config files etc as staging
area for tape
```

Create the clusterwide tablespaces (in this example, a single tablespace named 'appdata') =>

```
$ psql template1
....
```

```

template1=# CREATE TABLESPACE appdata LOCATION '/var/opt/postgresql/CLUST/data';
template1=# SELECT spcname FROM pg_tablespace;
  spcname
-----
pg_default
pg_global
appdata
(3 rows)

```

and add to the server config =>

```
default_tablespace = 'appdata'
```

Next, create the database itself (eg name = db9, unicode char set) =>

```

$ createdb -D appdata -E UNICODE -e db9      # appdata = default TABLESPACE
$ createlang -d db9 plpgsql                  # install 'Oracle PL/SQL like' language

```

WAL logs are stored in the directory `pg_xlog` under the data directory. Shut the server down & move the directory `pg_xlog` to `/var/opt/postgresql/CLUST/wal` and create a symbolic link from the original location in the main data directory to the new path.

```

$ pg_ctl stop
$ cd $PGDATA
$ mv pg_xlog /var/opt/postgresql/CLUST/wal
$ ls /var/opt/postgresql/CLUST/wal
$ ln -s /var/opt/postgresql/CLUST/wal/pg_xlog $PGDATA/pg_xlog # soft link as across FS
$ pg_ctl start -l /tmp/logfile

```

Assuming all is now working OK, shutdown PostgreSQL & backup up all the PostgreSQL related FS above... just in case...!

User Accounts

Create 1 * power user to create/own/control the tables (*using psql*) =>

```
$ psql template1
    create user cxd with password 'abc';
    grant create on tablespace appdata to cxd;
```

Do not create any more superusers or users that can create databases!

Now create n* enduser accts to work against the data =>

```
$psql template1
    CREATE GROUP endusers;
    create user enduser1 with password 'xyz';
    ALTER GROUP endusers ADD USER enduser1;
```

```
$ psql db9 cxd
    grant select. on <table>. to group endusers;
```

JDBC driver

A pure Java (Type 4) JDBC driver implementation can be downloaded from <http://jdbc.postgresql.org/>

Assuming the use of the SDK 1.4 or 1.5, download *postgresql-8.0-311.jdbc3.jar*

and include this in your application CLASSPATH.

(If moving JAR files between different hardware types, always ftp in BIN mode).

Configure PostgreSQL to accept JDBC Connections

To allow the postmaster listener to accept TCP/IP connections from client nodes running the JDBC applications, edit the server configuration file and change

```
listen_addresses = '*'           # * = any IP interface
```

Alternatively, this parameter can specify only selected IP interfaces (see documentation).

In addition, the client authentication file will need to be edited to allow access to our database server.

First take a backup of the file =>

```
$ cp pg_hba.conf pg_hba.conf.orig
```

Add the following line =>

```
host db9 cxd 0.0.0.0/0 password
```

where , for this example, database → db9, user → cxd, auth → password

Switching JDBC applications from Oracle to PostgreSQL

The URL used to connect to the PostgreSQL server should be of the form

```
jdbc:postgresql://host:port/database
```

If used, replace the line (used to load the JDBC driver)

```
Class.forName ("oracle.jdbc.driver.OracleDriver");
```

with

```
Class.forName("org.postgresql.Driver");
```

Remove any Oracle JDBC extensions, such as

```
((OracleConnection)con2).setDefaultRowPrefetch(50);
```

Instead, the row pre-fetch must be specified at an individual *Statement* level =>

```
eg.    PreparedStatement pi = con1.prepareStatement("select....");  
        pi.setFetchSize(50);
```

If not set, the default fetch size = 0;

Likewise, any non ANSI SQL extensions will need changing.

For example sequence numbers

```
Oracle => online_id.nextval
```

should be replaced by

```
PostgreSQL => nextval('online_id')
```

Oracle 'hints' embedded within SQL statements are ignored by PostgreSQL.

Now test your application!

Concluding Remarks

At this stage, you should now have a working PostgreSQL database fronted by a JDBC based application, and the foundations will have been laid for :

- A reasonably level of resilience (recoverability)

- A good starting IO distribution

The next step is to tune the system under load... and that's another doc...

Chris Drawater has been working with RDBMSs since 1987 and the JDBC API since late 1996, and can be contacted at chris.drawater@three.co.uk or drawater@compuserve.com .

Appendix 1 – Example .profile

```
TMPDIR=/tmp
export TMPDIR
```

```
PATH=/usr/bin:/usr/ucb:/etc/..:/usr/sfw/bin:/usr/local/bin:/usr/ccs/bin:$PATH
export PATH
```

```
#####
# PostgreSQL 8.0.2 runtime
#####
```

```
LD_LIBRARY_PATH=/opt/postgresql/8.0.2/lib
PATH=/opt/postgresql/8.0.2/bin:$PATH
export PATH LD_LIBRARY_PATH
```

```
PGDATA=/var/opt/postgresql/CLUST/sys
export PGDATA
```