Author : Chris Drawater
Date : 15/01/2007
Version : 1.0

PostgreSQL 8.2.1 on Solaris 10 – Deployment Guidelines

Abstract

Advance planning enables PostgreSQL 8.2.1 to be quickly deployed in a basic but resilient and IO efficient manner.

Introduction

This paper documents how to deploy PostgreSQL 8.2 in a basic but both resilient and IO efficient manner.

It is based upon experience with the following configurations =>
PostgreSQL 8.2.1 on Solaris 10
using the PostgreSQL distributions =>
postgresql-base-8.2.1.tar.gz

Much of the contents is also applicable to Linux (SUSE) implementations.

Abbreviations & Definitions

OLTP → Online Transaction Processing (ie. no data trawling or MIS etc)
SAN → Storage Attached Network
LUN → virtual disc/partition from a SAN attached disc array
FS → Filesystem
HBA → host bus adapter (usually a Fibre Channel interfacew card - connects node to SAN for the purposes of this doc)
WAL → Write Ahead Log (ie TX log)

Please see Appendix 1 for PostgreSQL Background for Oracle DBAs

IO resilience

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

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

So the basic rules for resilience are as follows.

For non disc array 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 (and not sharing RAID groups)
- use separate disc/volume groups for recovery components and data (helps enforce the previous)
- use Host Adapter Multipathing drivers (such as *mpxio*) with 2 or more HBAs for access to array .
- ensure each LUN has 2 or more paths to the array

For all deployments →

- 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 a later section).

Whenever possible, ensure that every PostgreSQL component (including binaries etc) resides on resilient disc storage !

For manageability, keep the software distribution and binaries separate from the database objects. Likewise, keep the system catalogs and non-application data separate from the application specific data.

IO performance

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 WAL IO performance are as follows :

- keep WAL on dedicated spindles/LUNs (mirror/stripe in preference to RAID 5)
- keep WAL and arch WAL on separate spindles/LUNs (to reduce IO on WAL spindles).
- Keep WAL mirrors on separate discs/controllers/LUNs

And for the data IO :

- use PostgreSQL tablespaces to distribute data and thus IO across multiple FS
- use your Volume Manager (eg VxVM, Solaris Volume Manager) to stripe individual data FS across LUNS/discs *(alternatively RAID only if there is no underlying HW RAID and mirroring is not being used)*
- use 1 Mb chunks/units if unsure as to what chunk size to use.
-

with specifically for disc arrays :

- use a LUN:HBA mapping so that any stripe across the LUNs also stripes across the HBAs
- use 2Gbit HBAs for SAN access

Finally , mount FS with Direct IO options for WAL logs and optionally data.

Filesystems

A number of distinctive storage requirements can be identified =>

- Software tree (Binaries, Source, distr)
- Shared PG sys data
- WAL logs
- Arch WAL logs
- Application data
- Admin logfiles
- Backup directory - optional

For the purposes of this document , the following minimal set of FS (note, not simply directories) are suggested =>

[Disc Grp 1]

/opt/postgresql/8.2.1 *# default 4 Gb for software tree*
/var/opt/postgresql *# default 100 Mb*
/var/opt/postgresql/CLUST/admin *# admin logs etc*

/var/opt/postgresql/CLUST/sys *# default size 1Gb for shared sys data*
/var/opt/postgresql/CLUST/data01 *# application data + user DB sys catalogs*

[Disc Grp 2]

/var/opt/postgresql/CLUST/wal *# WAL location*
/var/opt/postgresql/CLUST/archwal *# archived WALs*
/var/opt/postgresql/CLUST/backup *# optional backup staging area for tape (with size > sum all other PG FS)*

where CLUST is your chosen name for the Postgres DB cluster

For enhanced IO distribution , a number of *.../data* FS (eg *data02*, *data03* etc) could be deployed.

If using UFS or VxFS filesystems consider using direct IO for the following FS =>

/var/opt/postgresql/CLUST/wal *# use directIO*
/var/opt/postgresql/CLUST/data01 *# use directIO if very write intensive*

With UFS, add the following options
forcedirectio, noatime
to the relevant FS mount directives in */etc/vfstab* .

Note that the PostgreSQL 8.1.1. release notes indicate that PostgreSQL uses O_DIRECT if available when using O_SYNC for the *wal_sync_method* .

Also create directories
/opt/postgresql/8.2.1/source *# source code*
/opt/postgresql/8.2.1/distr *# downloaded distribution*
/opt/postgresql/8.2.1/lib *# exe etc*

All FS & directories to be owned by user *postgres:pgdba* with 700 permissions

Installation Pre-requisites – OS accts, Compilers & IPC !

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 once installed, should be found in

/usr/sfw/bin

Create the Unix acct *postgres* in group *pgdba*
with a home directory of say */export/home/postgresql*
using
$ u*seradd utility*
or hack
/etc/group then */etc/passwd* then run *pwconv* and then *passwd postgres*

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

```
set shmsys:shminfo_shmmax=1300000000
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 usually 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/8.2.1/distr
$ gunzip postgresql-base-8.2.1.tar.gz

$ cd /opt/postgresql/8.2.1/source
$ tar -xvof /opt/postgresql/8.2.1/distr/postgresql-base-8.2.1.tar

Set Unix environment =>

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

Configure the build options =>

$ cd /opt/postgresql/8.2.1/source/postgresql-8.2.1
$ ./configure --prefix=/opt/postgresql/8.2.1 --with-pgport=5432 --without-readline CC=/usr/sfw/bin/gcc CFLAGS='-O3'

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

The *CFLAGS* flag is optional *(see gcc 3.4.4 optimize Options)*

And build =>

$ gmake
$ gmake install

Setup Unix environment

Add to the Unix environment, the following =>

```
LD_LIBRARY_PATH=/opt/postgresql/8.2.1/lib
PATH=/opt/postgresql/8.2.1/bin:$PATH
export PATH LD_LIBRARY_PATH

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

At this point, it's probably worth creating a *.profile* as per Appendix 2.

Create Database(Catalog) Cluster

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 the database storage area, and create the 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

Verify Server Startup & Shutdown

Check the startup of the PostgreSQL cluster =>

$ pg_ctl start -l /tmp/logfile # temp logfile until server tuned
$ pg_ctl stop

Now verify that you can log onto the system using the super user password =>

$ psql template1

Check the shutdown of the PostgreSQL cluster =>

$ pg_ctl stop

Configure PostgreSQL to accept Local OS Authentication

First take a backup of the client authentication file =>

$ cp pg_hba.conf pg_hba.conf.orig

FYI, on Linux it's fairly simple →

 # Linux only
local all postgres ident sameuser
local all all md5

but for Solaris, only the following works

Solaris
local all all md5

so for batch jobs, the prompt for password causes problems….

Re-verify server shutdown, startup & login.

Basic Tuning

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 = '*'
port = 5432

# security
#password_encryption = on

# data buffer cache
shared_buffers = 256MB                  # as 8Kb blocks

# free space
max_fsm_pages = 40000                   # check vacuum output

# log related
fsync = on                              # resilience
wal_sync_method = open_sync             # resilience
wal_buffers=32
commit_delay = 10000                    # group commit if works (in microseconds)
commit_siblings = 3

#archiving
archive_command = 'cp "%p" /var/opt/postgresql/CLUST/archwal/"%f"'

#temp
work_mem = 10MB                         # for sorts/hash prior to temp disk usage
temp_buffers=1MB                        # access to temporary tables

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

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

log_destination = 'stderr'          .
redirect_stderr = on
log_directory = '/var/opt/postgresql/CLUST/admin'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
log_rotation_age = 1d                   # 1 day


# vacuuming
autovacuum = on
```

stats_start_collector = on
stats_row_level = on

#transaction/locks
default_transaction_isolation = 'read committed'

 # default tablespace
default_tablespace = "ON A PER APPLICATION BASIS"

This is a basic 'first-cut' tuning which will need modification and enhancement with real application workloads.

Restart the servers =>

$ pg_ctl start

Create the Application Database

This requires the filesystems =>

```
/var/opt/postgresql/CLUST/wal          #  WAL location
/var/opt/postgresql/CLUST/archwal      # archived WALs
/var/opt/postgresql/CLUST/data01       #  application data + DB sys catalogs
```

plus maybe also =>

```
/var/opt/postgresql/CLUST/backup       # optional  backup 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/data01';
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
```

And verify =>

```
$ psql db9
```

Relocate WAL logs

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
```

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

User Accounts

Please see Reference 1.

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 *pg_hba.conf* will need to edited to allow access to our database server.

Add the following line =>

host db9 cxd 0.0.0.0/0 md5

where , for this example, database → db9, user → cxd, auth → md5

The below also works but is less secure (as the password is passed unencrypted over the network) =>

host db9 cxd 0.0.0.0/0 password

Concluding Remarks

At this stage, you should now have a working PostgreSQL 8.2 installation with the foundations laid for :

- a reasonably good level of resilience (recoverability)
- good IO distribution

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

Acknowledgements

Thanks to Elizabeth Hope for the Linux local OS authentication example.

References

1. Drawater (2007), PostgreSQL 8.2.1 – A User Management Example, v1.0

Appendix 1 - 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
- 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.
- There is no dedicated log writer process.
- User accounts (aka roles with logins) are cluster wide by default
- A user is not synonymous with a schema - 1 user can have n * schemas (collection of objects).

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* end users with appropriate read/write access
- Use only ANSI SQL datatypes and DDL.

Wherever possible, avoid DB specific SQL extensions so as to ensure cross-database portability

Appendix 2 – Example .profile

```
TMPDIR=/tmp
export TMPDIR

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

########################
# PostgreSQL 811 runtime
########################

LD_LIBRARY_PATH=/opt/postgresql/8.2.1/lib
PATH=/opt/postgresql/8.2.1/bin:$PATH
export PATH LD_LIBRARY_PATH

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