Author : Chris Drawater
Date : Feb 2013
Version : 1.1

PostgreSQL 9.2.3 - Deployment on Windows 2008 R2 (SSD based)

Abstract

A basic overview of deploying PostgreSQL on SSD based hardware running Windows 2008 R2.

Introduction

This paper documents the deployment of PostgreSQL 9.2.3 on Windows 2008 R2 running on SSD based hardware,

It is assumed that the reader has a rudimentary knowledge of PostgreSQL.

Abbreviations &Definitions

SSD →Solid State Disc
DW → Data Warehouse
FS →Filesystem
WAL → Write Ahead Log (ie TX log)

Products named herein this paper may be trademarks of their respective manufacturers and are hereby recognized.

All trademarks are recognized as the property of their respective owners.

For demonstrative purposes, 'db92' is used as the name of the database.

Hardware Configuration

HP DL360 with 36GB, 2 * E5640 (total 8 cores) with

- 8 * 240GB OCZ Vertex3 Max IOPS Edition Drives
- LSI MegaRAID 9265-8i 6G RAID Card

The 8 SSDs were configured as 1 logical volume RAID 6 (ie6+2).

Running commercial databases , this hardware configuration has seen to be capable of supporting the following performance :

- Parallel table scan rate →upto 2.7 GB/sec physical IO 15% cpu
- Insert as select →upto 1500 MB/sec , cpu upto 70%
- Tablespace creation IO writes → 1700MB/sec with <10% CPU once settled.

Filesystems

Normally ,it is recommended that a production PostgreSQLfilesystem layout utilize a number of isolated storage areas to balance (and maximize) IO throughput against resilience/recovery.

However for our SSD based system, we can get away with using a single RAID 6 based filesystem as per the previous section.

Our dedicated drive and FS for PostgreSQL is *D:*

Open a Command Prompt Window (as Administrator) and run the following →

```
D:
set PGDRIVE=d:
mkdir %PGDRIVE%\PostgreSQL
mkdir %PGDRIVE%\pgData01
mkdir %PGDRIVE%\pgWal01
mkdir %PGDRIVE%\pgSys01
mkdir %PGDRIVE%\pgTemp01
mkdir %PGDRIVE%\pgServerLogs
```

Download Software

The Windows 64 bit PostgreSQL installer

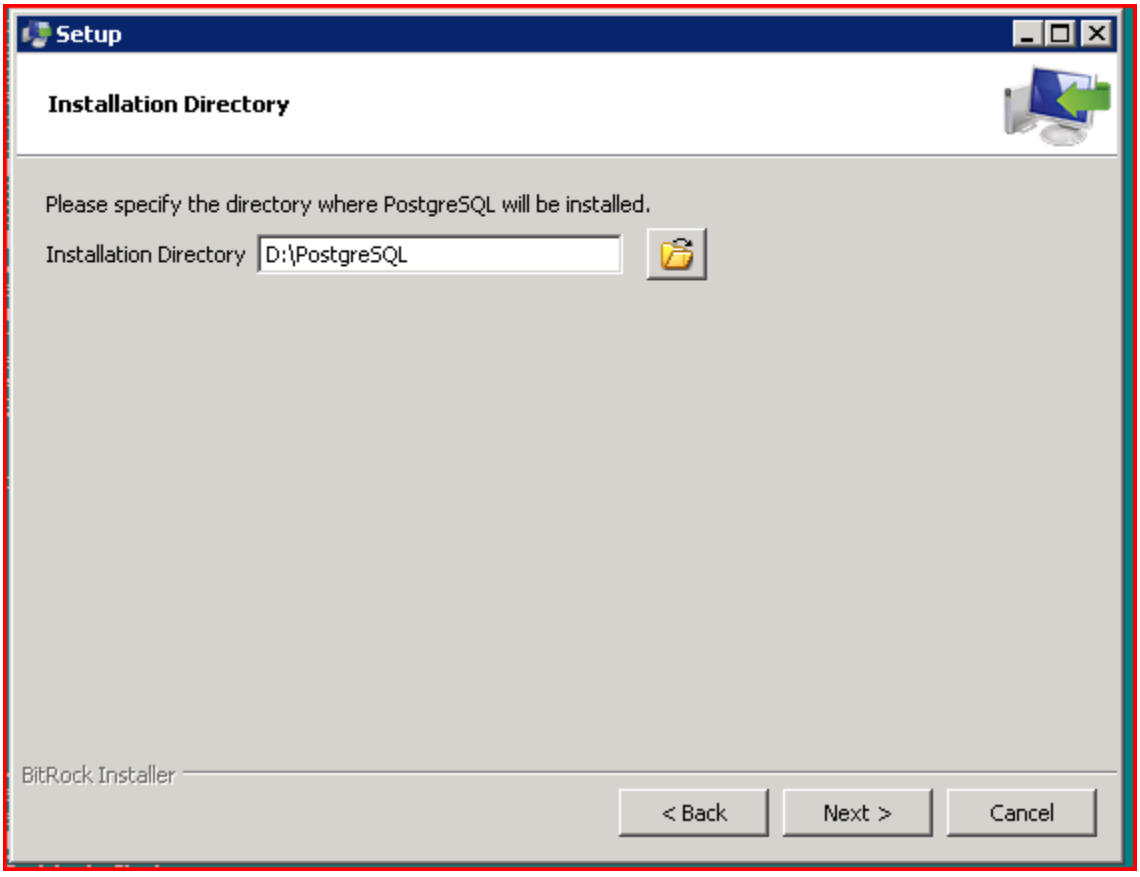
postgresql-9.2.3-1-windows-x64

can be downloaded from http://www.enterprisedb.com/products-services-training/pgdownload

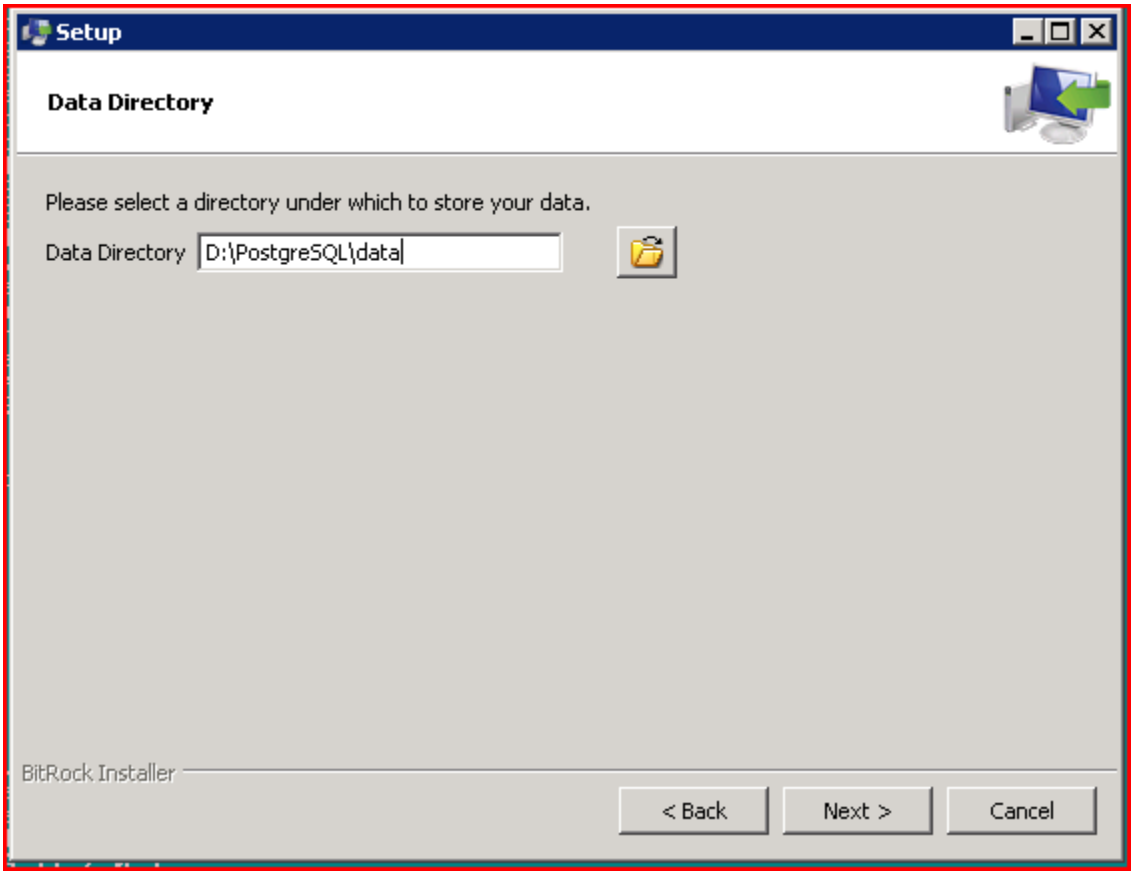
Install thePostgreSQL software using the GUI installer

Run in a DOS Window (opened as Administrator)

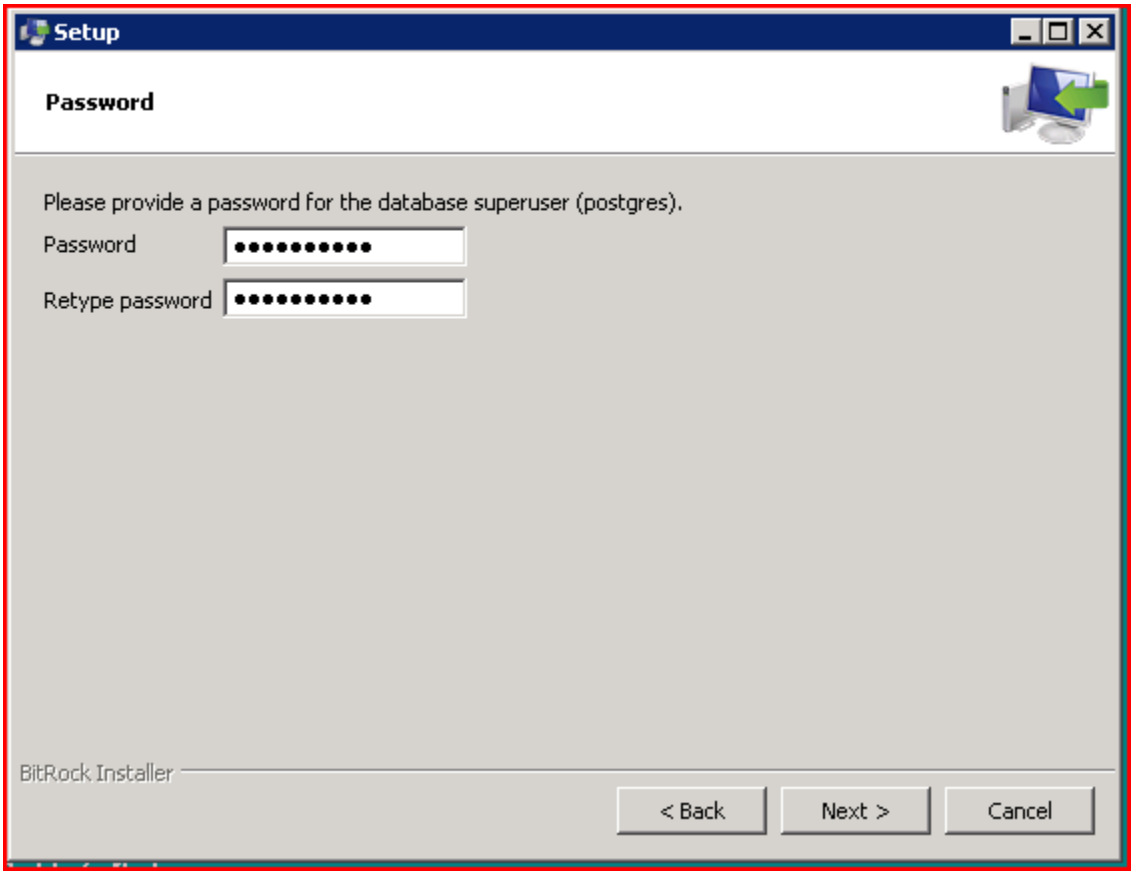
Install into → d:\postgresql



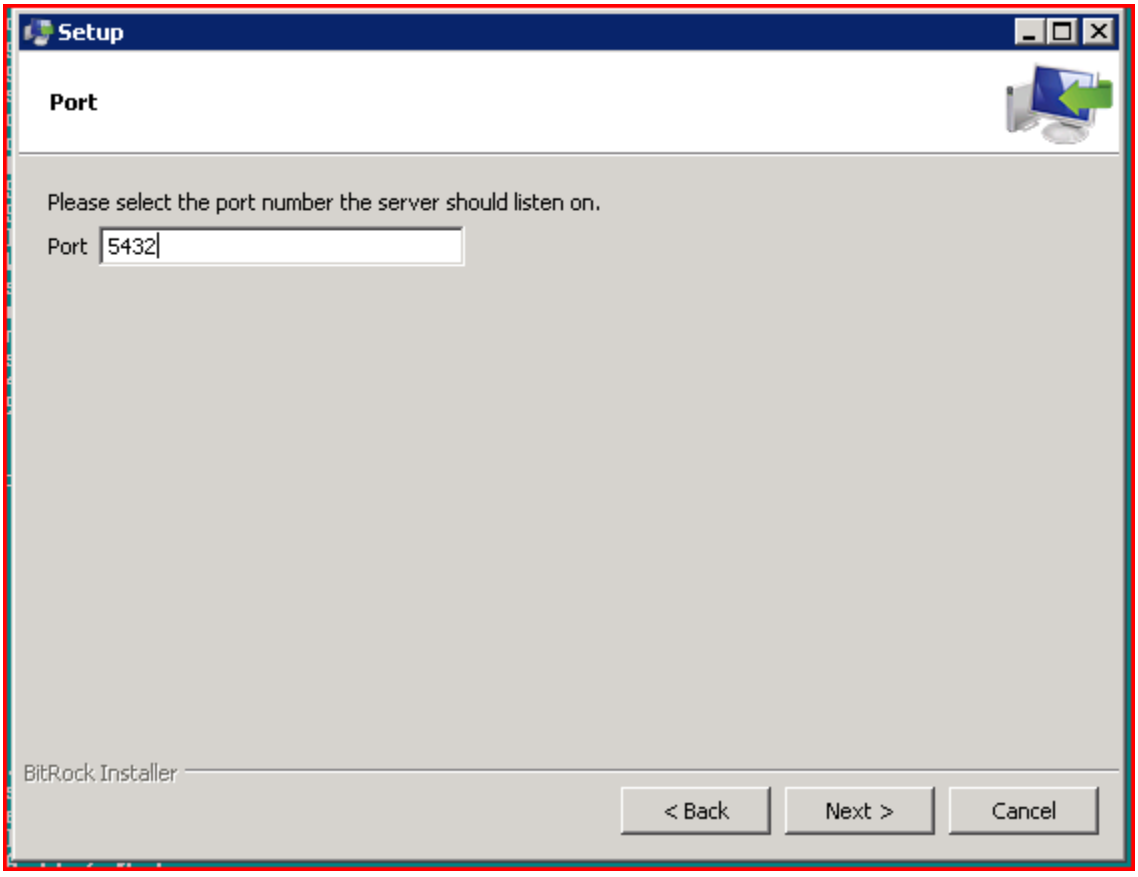
As the data directory , use → d:\postgresql\data



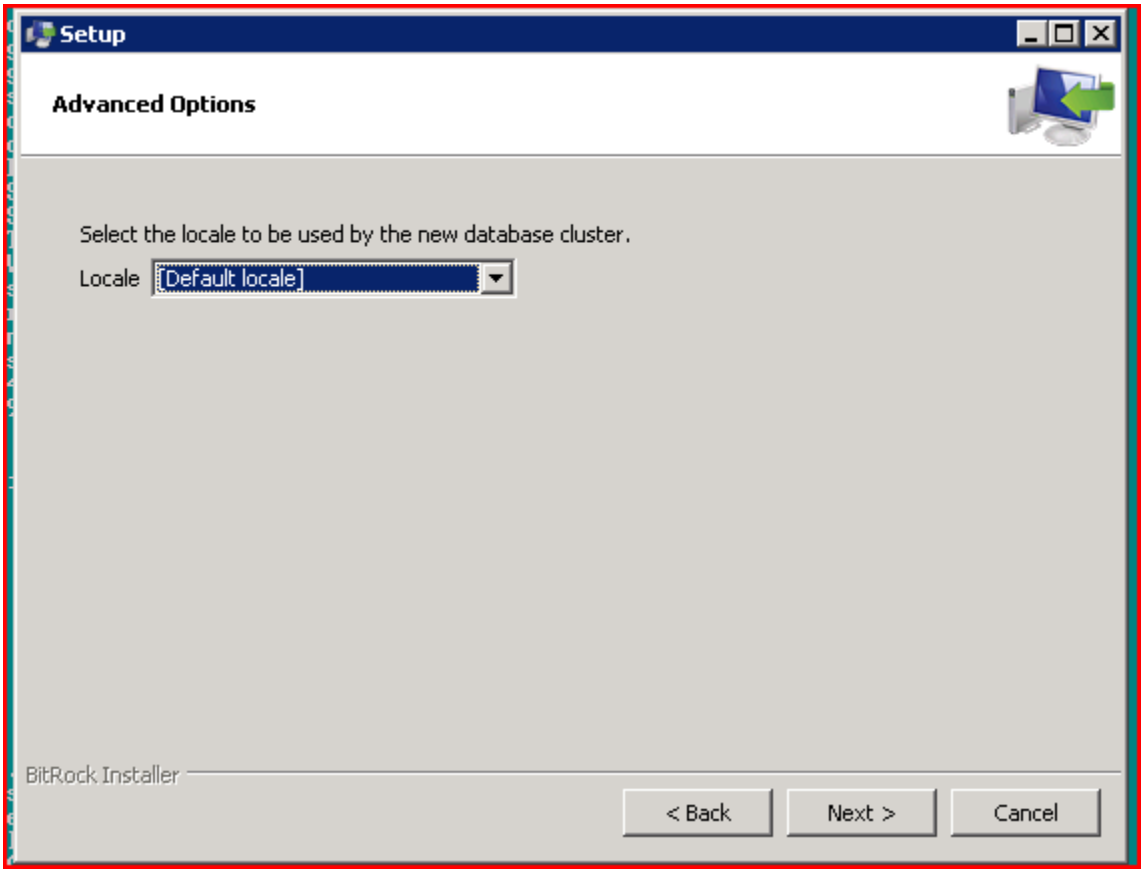
Choose and Set the password for the POSTGRES superuser and Windows service →eg XXXXXX



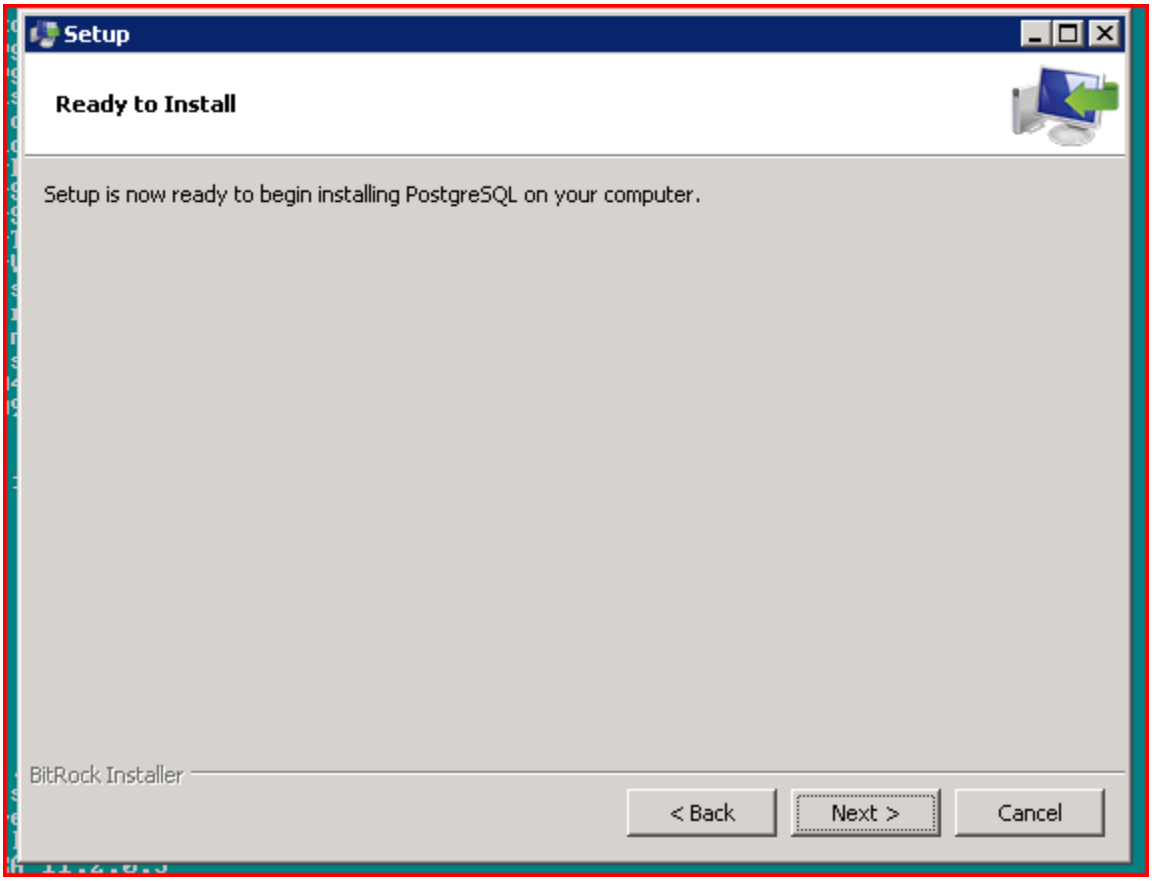
Set the listener port → 5432



Set the locale → default (or English,UK)



The installer is now ready to install on your Windows box!



Click on the next button and wait…

Unclick stackbuilder option and finish.

So far, fairly easy!

TableSpace Creation

Before creating the application database, tablespaces need to be created into which various database disc storage components (for example, temp, application data, system catalogs) can be installed in a meaningful (and if not SSD based, resilient and IO performant) way.

Open a DOS window

cd d:\postgresql

edit

pg_env.bat

to remove the quote marks in the PATH entry, then run.

The PostgreSQL environmental variables are now set up.

Fire up *psql* →

set PGPASSWORD=XXXXXX (ie the PostgreSQLsuperuser password entered in the GUI install)
psql postgres

Then run the SQL →

CREATE TABLESPACE appdata LOCATION 'd:\pgData01';
CREATE TABLESPACE sysdata LOCATION 'd:\pgSys01';
CREATE TABLESPACE temp LOCATION 'd:\pgTemp01';

and check all is OK using →

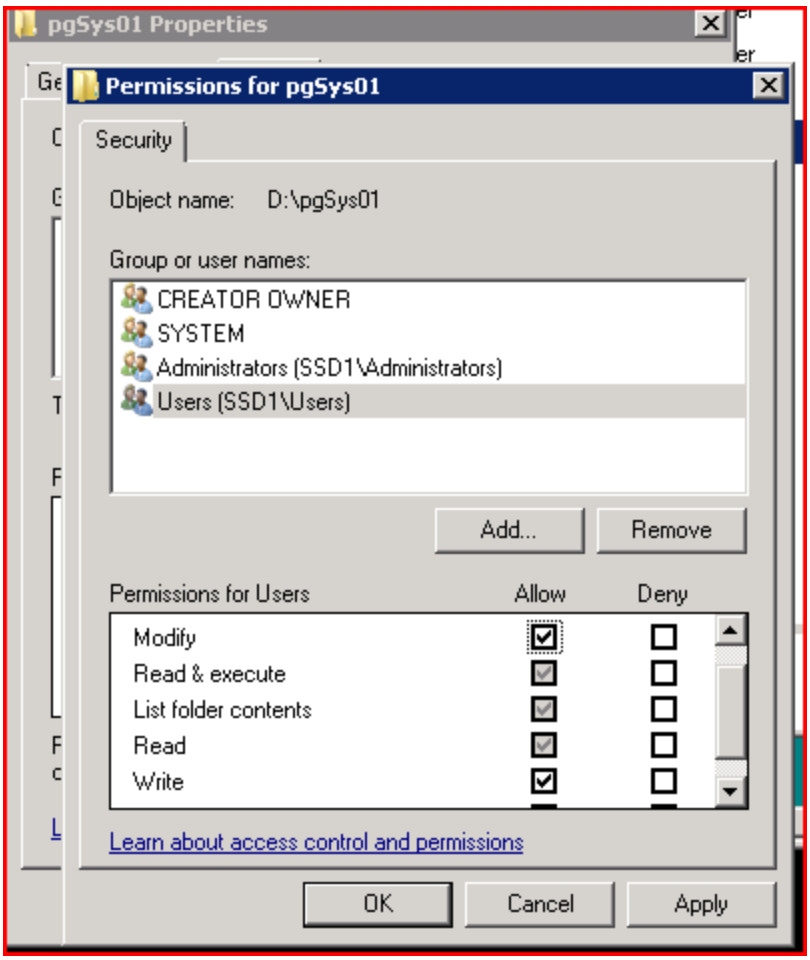
*SELECT * FROM pg_tablespace;*
\q

If you encounter the following error →

```
postgres=# CREATE TABLESPACE appdata LOCATION  'd:\pgData01';
ERROR:  could not set permissions on directory "d:/pgData01": Permission denied
```

Then you will need to give write/modify priv on the directory to the 'Users' group.

The simplest (but least security conscious) method is to modify the properties then security settings .→



Alternatively, and slightly better in terms of system security, run (with Windows Administrator privilage)→

icacls<dir> /grant networkservice:M

For example,

icacls d:\pgData01 /grant networkservice:M
icacls d:\pgSys01 /grant networkservice:M
icacls d:\pgTemp01 /grant networkservice:M

and then via *psql* , run→

CREATE TABLESPACE appdata LOCATION 'd:\pgData01';
CREATE TABLESPACE sysdata LOCATION 'd:\pgSys01';
CREATE TABLESPACE temp LOCATION 'd:\pgTemp01';

Returning to the tablespace creation, the server configuration needs to be updated to reflect the default and temp tablespaces.

Take a safety copy of the server configfile **postgresql.conf** (found in %PGDATA%)→

D:
cd %PGDATA%
copy postgresql.conf postgresql.conf.orig

Modify the server config→

default_tablespace = 'appdata'

temp_tablespaces = 'temp'

under the Client Connection Defaults section.

Restart the PostgreSQLWindows Service.

Create the Database

Next, from the DOS prompt, create the database itself (eg name = db92, unicode char set, sysdata = default TABLESPACE that will hold our system catalogs) →

createdb -D sysdata -E UNICODE -e -U postgres db92

```
D:\PostgreSQL\data>createdb  -D sysdata -E UNICODE  -e  -U postgres db92
CREATE DATABASE db92 TABLESPACE sysdata ENCODING 'UNICODE';
```

Move the WAL logs

The WAL logs are stored in the directory *pg_xlog* under the data directory.

Shut the server down, move the directory *pg_xlog* to *d:\pgWal01* then create a symbolic link from the original location in the main data directory to the new path.

Stop the Windows service.

Open a command prompt Window as Administrator and execute the following →

d:

cd %PGDATA%

icacls D:\pgWal01 /grant networkservice:M

copy pg_xlog D:\pgWal01

rename pg_xlog old_pg_xlog

mklink /J pg_xlog D:\pgWal01

Restart thePostgreSQLWindows Service.

Server Configuration

Change
log_directory = 'D:/pgServerLogs'
to reflect the drive used (for example D:).

Tune as needed.

For example, the following is the tuning for the SSD based DW.

```
shared_buffers = 8192MB
temp_buffers = 512MB
work_mem = 512MB
bgwriter_delay = 200ms
bgwriter_lru_maxpages = 1000
bgwriter_lru_multiplier = 5.0

wal_level = minimal
fsync = on
synchronous_commit = on
wal_sync_method = fsync_writethrough
commit_delay = 10000
commit_siblings = 3

checkpoint_segments = 20
checkpoint_timeout = 300s
checkpoint_warning = 30s

log_directory = 'D:/pgServerLogs'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
log_rotation_size = 100MB
log_min_duration_statement = 30000

autovacuum = on
log_autovacuum_min_duration = 60000
autovacuum_max_workers = 6
autovacuum_vacuum_scale_factor = 0.01
autovacuum_analyze_scale_factor = 0.05

default_transaction_isolation = 'read committed'
max_locks_per_transaction = 25600
```

In very simple terms, the tuning was
- to use 25% physical memory
- no archiving of WAL logs

Please note that this tuning may not necessarily be applicable to your PosgreSQL installation.

Restart thePostgreSQL Windows Service.

User Accounts

Please see Reference 1 for information on how to set up PostgreSQL end user accts.

Configure PostgreSQL to accept JDBC Connections

To allow the postmaster listener to accept TCP/IP connections from client nodes running 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 db92 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 db92 cxd 0.0.0.0/0 password

PostgreSQL on SSDs :Performance

So what of the performance of a PostgreSQL based DW running on the aforementioned configuration ?

PG does not parallelise table scans so the single BE server (per client) performs all IO (although some IO is effectively parallel using async IO on some platforms)

Single BE server call table scan speed was observed at →

- Up to 440Mb/sec physical IO (with circa 1GB/sec logical IO), 10- 15% cpu
- 250m mobile phone calls scanned in 114 seconds

Interestingly, this non-parallelised IO throws up a number of benefits→

- The non-parallel nature of DB server IO effectively regulates the IO resource used per end user – no single user can use all the physical IO!
- The hardware configuration could handle 8 concurrent users scanning data (as opposed to simply active users), based upon cpu/physical IO rate

The use of SSD based hardware takes PostgreSQL into the small Data Warehouse or DataMart space.

Concluding Remarks

This brief paper demonstrates, for R&D/information purposes, the deployment of PostgreSQL 9.2.3 onto Windows 2008 R2 on SSD based hardware..

Chris Drawater has been working with RDBMSs since 1987 and can be contacted at drawater@btinternet.com .

References

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