PostgreSQL 8.2 efektivně

Administrace

Pavel Stěhule

http://www.pgsql.cz

25. 2. 2007

Omezení přístupu k databázi

Přehled

Přístup k databázi lze omezit

- na úrovni protokolu

```
listen_addresses = 'localhost', '*'
port = 5432
```

- určením autentifikační metody uživatele pro každou databázi - soubor *pg_hba.conf*. Výchozí nastavení přeložené PostgreSQL je absolutně benevoletní. PostgreSQL v distribucích je bezpečná. Minimální doporučená úroveň je `md5`.
- na moderních o.s. typu unix lze pro lokální přístup použít metodu *ident*. Soubor *pg_ident.conf* slouží k mapování unix. jmen na uživatelské účty PostgreSQL.

Změna hesla

Heslo změníme příkazem:

```
ALTER USER name PASSWORD 'heslo';
```


Omezení přístupu k databázi

Poznámky I.

Zapomenuté heslo uživatele postgres

- V *pg_hba.conf* nastavíme uživateli postgres methodu trust. Po dobu, kdy je tato ověřovací metoda povolena je nutné zabránit přístupu dalším uživatelům k serveru.
- Spuštění databáze v single režimu Nemůžeme-li zajistit izolaci databáze, lze spustit PostgreSQL server v tzv. single režimu. Tento režim se používá jednak při inicializaci clusteru, jednak v případě nutných zásahů do systémového katalogu, které jsou v normálním režimu zakázány. **Neodborný zásah v tomto režimu může vést k poškození systémového katalogu s důsledkem nedeterministického chování databázového systému vedoucí ke ztrátě uložených dat.**

```
bash-3.1$ postmaster --single template1 -D /usr/local/pgsql/data/
PostgreSQL stand-alone backend 8.3devel

backend> alter user postgres password 'omega';
```

Omezení přístupu k databázi

Poznámky II.

Zamezení zadávání hesla

Standardně komunikaci mezi serverem a klientem zajišťuje knihovna *libpq*. Ta před explicitním vyžádáním hesla kontroluje, zda-li neexistuje soubor *~/.pgpass*. Pokud tento soubor není dostatečně zabezpečen, je ignorován. Tento textový soubor obsahuje řádky ve formátu `hostname:port:database:username:password`.

Údržba databáze

Správa datového adresáře

Inicializace

Slovo cluster má v PostgreSQL význam prostoru pro všechny databáze, ke kterým lze přistupovat určenou IP adresou a portem. Všechny databáze v clusteru sdílí konfiguaraci a uživatele. Na jednom počítači lze provozovat více clusterů stejných nebo různých verzí PostgreSQL.

```
initdb [OPTION]... [DATADIR]
    -E, --encoding=ENCODING  určí kódování
    --locale=LOCALE          určí locales
```

Poznámky

- v clusteru nic nemazat (`pg_resetxlog`)
- z clusteru nekopírovat datové soubory (nepřenosné)
- lze kopírovat celý adresář clusteru (zastavené PostgreSQL)
- lze kopírovat celý adresář clusteru (*aktivní export write ahead logu*)

Údržba databáze

Správa datového adresáře

Umístění datového adresáře

liší se dle zvyklostí konkrétních distribucí:

default `/usr/local/pgsql/data`, vytváří se ručně

Red Hat `/var/lib/pgsql/data`, vytváří se automaticky, při prvním startu

Kontrola korektního chování *LOCALE*

Okamžitě po instalaci ověřte, zda je korektně podporováno národní prostředí. Nejlépe `SELECT upper('příliš žluťoučký kůň')`. Vybrané *locale* clusteru se musí shodovat s kódováním databáze, např. pro UTF8 musíme používat *locale* `cs_CZ.UTF8`.

Údržba databáze

Opakující se činnosti

Pravidelné

- **1x denně** `VACUUM ANALYZE` (`cron`, `autovacuum`),
- 1x měsíčně `REINDEX` databáze nebo alespoň nejčastěji modifikovaných tabulek,

Nepravidelné

- pokud dochází vyhrazený prostor na disku pro data `VACUUM FULL`,
- pokud hrozí přetečení čítače transakcí (varování v logu) `FACUUM FREEZE` (1x za několik let),
- analýza pomalých dotazů (limit 200ms) a jejich eliminace přidáním nových indexů.
- čištění datového schéma (nutno zálohovat a dokumentovat)
  - odstranění nepoužívaných tabulek (`pg_stat_all_tables`),
  - odstranění nepoužívaných indexů (`pg_stat_all_indexes`).

Každý index zabírá prostor na disku a zpomaluje operace `INSERT`, `UPDATE`, `DELETE`. Proto indexy vytváříme pouze tehdy, když mají nějaký efekt.

Údržba databáze

Zastavení, spuštění PostgreSQL

Start, Reload, Restart serveru

- `/etc/init.d/postgres (start|reload|restart|stop|status)`
- `pg_ctl` lze specifikovat režimy ukončení
  - `smart` čeká, až se odhlásí všichni klienti,
  - `fast` nečeká na odhlášení klientu, provede úplný proces ukončení,
  - `immediate` skončí okamžitě s důsledkem obnovy po nekorektním ukončení při příštím startu.

Preferujeme co nejšetrnější možnou metodu. V případě, podivného chování jednoho klientského procesu, lze zaslat signál `sigint` obslužnému procesu klienta.

Ukončení klienta z prostředí SQL

1. získání *pid* problémového klienta

```
select procpid, usename, current_query from pg_stat_activity;
procpid | usename |  current_query
  10144 | root    |  select fce();
```

2. odstranění procesu `select pg_cancel_backend(10144);`

Údržba databáze

Zastavení, spuštění PostgreSQL

Obnova po havárii

Pokud server nebyl ukončen korektně, je možné, že v paměti zůstanou servisní procesy PostgreSQL. Ty je nutné před opětovným spuštěním serveru ukončit. Výjmečně je nutné vyčistit sdílenou paměť příkazem `ipcclean`. Také je nutné explicitně odstranit soubor `dbcluster/postmaster.pid`. Za normálních okolností se spustí automaticky proces obnovy databáze na základě údajů uložených ve *write ahead logu*.

Doporučení

Je celkem na místě ověřit integritu databáze dumpem databáze. V případě problémů

- reindexace databáze včetně systémových tabulek
- obnova ze zálohy
- identifikace poškozených řádků a jejich odstranění. Příznakem poškozené databáze je pád serveru při sekvenčním čtení.

Správa uživatelů

createuser

```
Usage:
  createuser [OPTION]... [ROLENAME]

Options:
  -s, --superuser           role will be superuser
  -S, --no-superuser        role will not be superuser
  -d, --createdb            role can create new databases
  -D, --no-createdb         role cannot create databases
  -r, --createrole          role can create new roles
  -R, --no-createrole       role cannot create roles
  -l, --login               role can login (default)
  -L, --no-login            role cannot login
  -i, --inherit             role inherits privileges of roles it is a
                            member of (default)
  -I, --no-inherit          role does not inherit privileges
  -c, --connection-limit=N  connection limit for role (default: no limit)
  -P, --pwprompt            assign a password to new role
  -E, --encrypted           encrypt stored password
  -N, --unencrypted         do not encrypt stored password
  -e, --echo                show the commands being sent to the server
  -q, --quiet               don't write any messages
```


Správa uživatelů

CREATE ROLE

```
Command:     CREATE ROLE
Description: define a new database role
Syntax:
CREATE ROLE name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | CREATEUSER | NOCREATEUSER
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | IN ROLE rolename [, ...]
    | ROLE rolename [, ...]
    | ADMIN rolename [, ...]
    | USER rolename [, ...]
    | SYSID uid
```

Správa uživatelů

Výklad

Pravidla chování rolí I.

- Závislosti mezi rolemi tvoří orientovaný graf, který nesmí obsahovat cyklus (Pavel může získat práva Tomáše, zároveň ale Tomáš nemůže získat práva Pavla).
- Uživatel získá práva rolí, jejichž je členem (ke kterým má přístup, které může převzít).

  ```
  CREATE ROLE tom_a_pavel IN ROLE tom, pavel;
  ```

  Role *tom_a_pavel* může převzít roli Tomáše nebo Pavla (je to nadřazená role těmto rolím), a má práva jako Tomáš a Pavel dohromady.
- Roli můžeme definovat také tak, že určíme kdo tuto roli může používat.

  ```
  CREATE ROLE developer ROLE tom, pavel;
  ```

  Roli *developer* může použít jako Tomáš tak Pavel. Pokud role nemá atribut `INHERIT`, pak se uživatel do této role musí explicitně přenout příkazem `SET ROLE developer`. Cokoliv je vytvořeno (vlatsněno) touto rolí může používat (přístup, modifikace struktury) jak Tomáš, tak Pavel.

Správa uživatelů

Výklad

Pravidla chování rolí II.

- Uživatel může změnit vlastnictví objektů ke kterým má práva příkazem:

  ```
  ALTER TABLE objekt OWNER TO developer;
  ```

  ale nemůže se vzdát svých práv, tj. nemůže změnit vlastnictví tak, aby přišel o objekt (nelze databázový objekt darovat někomu neznámému s nímž nemám nic společného.

Rekapitulace

`CREATE ROLE` vytvoří novou roli.
`GRANT co TO komu` delegování určitého práva roli.
`GRANT r1 TO r2` role r2 získává stejná práva jako má role r1.
`REVOKE` odejmutí práv.
`ALTER TABLE .. OWNER TO ..` změna vlastnictví objektu.
\dg zobrazení rolí a členství v psql.

Export, import dat

Možnosti

SQL dump tabulek a struktury

Systémovým příkazem pg_dump dokážeme exportovat tabulku (nebo více tabulek) a to jak data, tak strukturu. Výsledný soubor obsahuje SQL příkazy. Data lze uložit jako sadu příkazů INSERT nebo jako jeden příkaz COPY.

COPY na serveru

Tento příkaz vytvoří (přečte) textový soubor (hodnoty oddělené čárkou nebo tabelátorem) uložený na serveru. Při zpracování nedochází k přenosu dat po síti. Musíme mít k dispozici prostor na serveru přístupný pro uživatele postgres.

COPY z klienta

Obdoba příkazu COPY implementovaná jako příkaz konzole psql. Čte (ukládá) soubory na stranně klienta, zajišťuje přenos dat po síti, a zpracování na straně serveru. Formát souboru je stejný jako na u příkazu COPY na serveru.

Export, import dat

pg_dump

```
pg_dump [OPTION]... [DBNAME]
      -a, --data-only           pouze data
      -c, --clean               předřadí příkazy pro zrušení
                                objektů
      -C, --create              vloží příkazy k vytvoření
                                objektů
      -d, --inserts             použije INSERT místo COPY
      -E, --encoding=ENCODING   použije určené kódování
      -s, --schema-only         pouze schéma
      --disable-triggers        po dobu načítání dat blokuje
                                triggery
      -t, --table=TABLE         pouze určitou tabulku
```

Export, import dat

COPY

```
COPY tablename [ ( column [, ...] ) ]
    (FROM|TO) { 'filename' | (STDIN|STDOUT) }
    [ [ WITH ]
          [ BINARY ]
          [ HEADER ]
          [ OIDS ]
          [ DELIMITER [ AS ] 'delimiter' ]
          [ NULL [ AS ] 'null string' ]
          [ CSV [ HEADER ]
                [ QUOTE [ AS ] 'quote' ]
                [ ESCAPE [ AS ] 'escape' ]
                [ (FORCE NOT NULL column [, ...]|
                   FORCE QUOTE column [, ...]) ]
```


Export, import dat

Efektivita

Přehled jednotlivých metod

Uvedená tabulka obsahuje údaje o importu jednoho miliónu řádek jednosloupcové celočíselné tabulky (testováno na notebooku Prestigio Nobile 156, P1.6, 500M).

Metoda	**Velikost**	**Čas**
INSERTS (autocommit on)	37,8 M	10 min
INSERTS (autocommit off)	37,8 M	2.2 min
COPY	6,8 M	10 sec
\COPY (UDP)	6.8 M	10 sec
COPY BINARY	10 M	7 sec
COPY (+ 1 index)	6.8 M	17 sec

Závěr

- Preferovat COPY,
- Zrušit všechny indexy (nejsnáze pomocí SQL procedury),
- zablokovat triggery (`ALTER TABLE name DISABLE TRIGGER ALL`).

Export, import dat

Import dat z dostupné živé databáze

Import pomocí univerzálního datového rozhraní

Pro opakované importy se vyplatí použít jednoduchou funkci v untrusted PLperlu a v této funkci se připojit ke zdrojové databázi, získat data a ta vrátit jako výsledek funkce. Odpadají tím někdy komplikované transformace dat (např. mezi PostgreSQL a MySQL) a také se snáze tato metoda automatizuje.

```
CREATE PROCEDURE exec_ora(connection_string text, sqlquery text)
RETURNS SETOF RECORD AS $$ use DBI;
  $dbh = DBI->connect('dbi:Oracle:', $_[0], '', {});
  $sth = $dbh->prepare($_[1]); $sth->execute();
  while ($dt = $sth->fetchrow_hashref) { return_next $dt; }
  $sth->finish(); $dbh->disconnect(); return undef;
$$ LANGUAGE plperlu;

SELECT * FROM exec_mysql('anonymous/tiger@heslo',
  'SELECT a,b FROM data WHERE ins = '10.2.2007') AS (a int, b int);
```


Zálohování, obnova databáze

Přehled

Přehled technik zálohování

K dispozici jsou tři způsoby zálohování:

- *SQL dump* příkazy pg_dump, pg_restore. Data lze uložit jako SQL skript nebo v speciálním komprimovaném formátu.
- *záloha na úrovni souborového systému* **Server musí být zastaven!** Lze zálohovat a obnovovat pouze kompletní db. cluster.

  `tar -cf backup.tar /usr/local/pgsql/data`
- *online zálohování* je založeno na tzv. *write ahead logu* (WAL), což je soubor do kterého se zapisují všechny změny v datech ještě předtím než se zapíší do datových souborů. Primárně tento log slouží k obnově databáze po havárii, můžeme jej však exportovat, uložit a použít pro vytvoření zálohy.
  - jediné možné řešení průběžného zálohování,
  - náročné na diskový prostor,
  - náročné na administraci,
  - zálohování i obnova je velice rychlé,
  - záloha není kompatibilní mezi 32 a 64 bit platformami.

Zálohování, obnova databáze

Vytvoření zálohy exportováním WAL

Postup

1. V *postgresql.conf* nastavíme `archive_command`. Tento příkaz zajistí přenesení segment logu na bezpečné médium. PostgreSQL neodstraní segment, dokud příkaz neproběhne bezchybně.

   ```
   archive_command = 'cp -i "%p" /mnt/server/archivedir/"%f"'
   ```

2. Export logu aktivujeme voláním `select pg_start_backup('navesti')`. Jako návěstí můžeme použít libovolný řetězec, např. cesta k záloze.
3. V tomto případě můžeme bezpečně za chodu zkopírovat obsah dovohého adresáře PostgreSQL. Není třeba zálohovat adresář *pg_xlog*.
4. po dokončení kopírování deaktivujeme export logu voláním `SELECT pg_stop_backup()`. Export logu může běžet libovolnou dobu, což je základ průběžného zálohování.

Průběžné zálohování

Segment se exportuje po naplnění (16MB) nebo po přednastaveném časovém intervalu (8.2). Efektivně jej lze komprimovat.

Zálohování, obnova databáze

Obnova ze zálohy exportovaného WAL

Postup

1. Zazálohujte si aktuální cluster (včetně *pg_xlog*).
2. Překopírujte data ze zálohy (zazálohovaný datový adresář).
3. Upravte soubor *recovery.conf* a uložte jej v adresáři clusteru. Vzor naleznete v podadresáři *shared*. Minimální změnou je nastavení položky `archive_command`.
4. do nadresáře *pg_xlog* zkopírujte všechny nezazálohované soubory z adresáře *pg_xlog* (to jsou WAL segmenty, které vznikly po deaktivaci exportu).
5. Nastartujte server. Při startu se automaticky spustí proces obnovy na základě WAL.

Konfigurace databáze

Volba souborového systému

Vliv souborového systému na výkon databáze

Nelze obecně říci, který souborový systém je optimální. Při **umělých** testech bylo pořadí souborových systémů následující: *JFS*, *ext2*, *Reiser3*, *ext3*, *XFS*. Rozdíl mezi nejpomalejší a nejrychlejší testovací konfigurací byl 30% (což se nebere jako natolik významná hodnota, aby se o tom příliš diskutovalo). U "high" řadiču je nutné explicitně povolit write cache (baterií zálohované řadiče).

Závěr

- používejte takový souborový systém, který je pro vás důvěryhodný (RAID 10),
- na UNIXech používejte mount parametr `noatime`,
- pokud jste jištěni UPS, lze pro *ext3* použít mount parametr `data=writeback`. výkonově se dostanete na úroveň *ext2*, v žádném případě se nedoporučuje změnit konfigurační parametr *fsync* na `off`,
- pokuste se umístit WAL na jiný nejlépe RAID 1 disk než data (symlink) .
- verifikujte konfiguraci testem *pgbench*, který by měl zobrazovat rámcově srovnatelné hodnoty s jinou instalací PostgreSQL na podobném hw.

Konfigurace databáze

Přidělení paměti

Strategie

PostgreSQL má mít přiděleno maximum paměti, aniž by zpomalila o.s. Přidělení paměti je určeno hodnotami určitých parametrů v *postgresql.conf*. Pouze parametr `work_mem` lze nastavovat dynamicky. Neexistuje úplná shoda ohledně doporučených hodnot. Jako rozumný minimální kompromis lze brát hodnoty z konfigurace verze 8.2.

postgresql.conf I.

`shared_buffers` velikost cache pro systémové objekty [32..256] MB (6..10%).

`work_mem` velikost alokované pracovní paměti **pro každé spojení**, omezuje použití diskové mezipaměti při operaci sort, určuje maximální velikost hash tabulek při operaci hash join, lze ji nastavit dynamicky před náročným dotazem [1..10] MB.

`max_fsm_pages`, `max_fsm_relation` určuje maximální počet diskových stránek, pro které se udržuje mapa volných stránek (příkaz DELETE). Příliš malá hodnota způsobí nevyužívání uvolněných bloků a alok. nových.

Konfigurace databáze

Přidělení paměti

postgresql.conf I.

`maintenance_work_mem` velikost paměti používané pro příkazy jako jsou `VACUUM` nebo `CREATE INDEX`. Jelikož není pravděpodobné, že by došlo k souběhu provádění těchto příkazů lze bezpečně tuto hodnotu nastavit výrazně vyšší než je `work_mem`. Doporučuje se 32 ... 256MB nebo 50..75% velikosti největší tabulky.

Konfigurace databáze

Parametrizace plánovače dotazů

Poznámka

Až na výjimky, parametry týkající se výběru nejvhodnějšího způsobu provádění dotazu jsou určené pouze pro vývojáře PostgreSQL, a mají umožnit vzdálenou diagnostiku nahlášených problémů.

postgresql.conf II.

`efective_cache_size` předpokládaná velikost celkové diskové vyrovnávací paměti použité pro jeden dotaz (včetně shared_buffers PostgreSQL a části sys. cache použité pro soubory PostgreSQL). Pozor na souběžné dotazy z různých tabulek, které se musí vejít do dostupného prostoru. Tato hodnota nesouvisí se skutečnou potřebou paměti. Vyšší hodnota znamená preferenci indexu (vyšší pravděpodobnost, že cennově náročnější index nalezneme v cache). Nižší preferenci sekvenčního čtení tabulky. Výchozí nastavení je 128 M. V Linuxu RAM - o.s. - ostatní aplikace (Linux agresivně používá cache).

Konfigurace databáze

Parametrizace procesu *autovacuum*

Strategie

Častější provedení příkazu `VACUUM` než je nutné, je menším zlem než nedostatečně časté provádění tohoto příkazu. Spouští se periodicky (`cron`) nebo při překročení dynamické prahové hodnoty (*autovacuum*). Tato hodnota roste s velikostí tabulky.

postgresql.conf III.

`stats_row_level` aktualizace provozních statistik (režie 20%).
`autovacuum` povolení samotného procesu (vyžaduje provozní statistiky)

Příkaz `VACUUM` se spustí, pokud počet modifikovaných řádků je větší než *autovacuum_vacuum_threshold* + (*autovacuum_vacuum_scale_factor* ∗ *velikost_tabulky*). Přibližně 20% počtu řádek v tabulce.
Příkaz `ANALYZE` se spustí, pokud počet modifikovaných řádků je větší než *autovacuum_analyze_threshold* + (*autovacuum_analyze_scale_factor* ∗ *velikost_tabulky*). Přibližně 10% počtu řádek v tabulce.

Monitorování databáze

Sledování aktivity

Pohled do systémových tabulek

- pohled *pg_stat_activity* obsahuje přehled činnosti přihlášených klientů.
- pohled *pg_stat_database* obsahuje počet přihlášených klientů k jednotlivým databázím.
- pohled *pg_stat_all_tables* obsahuje provozní statistiky tabulek.
- pohled *pg_stat_all_indexes* obsahuje provozní statistiky indexů.
- pohled *pg_locks* obsahuje seznam aktivních zámků.

postgresql.conf IV.

Sledování déletrvajících a chybných dotazů:

`log_min_error_statement = error` loguje všechny chybné SQL příkazy.
`log_min_duration_statement = 200` loguje všechny SQL příkazy prováděné déle než 200ms.

Na vytěžení údajů z logu lze použít tzv. PostgreSQL log analyzer *pgFouine*.

Monitorování databáze

Zjištění obsazeného prostoru databázovými objekty

```
-- setříděný výpis seznamu databází podle velikosti
root=# select datname, pg_size_pretty(pg_database_size(datname))
          from pg_database
         order by pg_database_size(datname) desc;
  datname   | pg_size_pretty
------------+----------------
 root       | 168 MB
 postgres   | 4088 kB
 regression | 3864 kB
 ...

 -- setříděný výpis tabulek v databázi podle velikosti
 root=# select n.nspname, c.relname, pg_size_pretty(pg_total_relation_size(c.oid))
           from pg_class c left join  pg_catalog.pg_namespace n on n.oid = c.relnamespace
          where c.relkind = 'r'
          order by pg_total_relation_size(c.oid) desc
          limit 10;
   Schema   |    relname     | pg_size_pretty
------------+----------------+----------------
 root       | test_data      | 138 MB
 public     | accounts       | 25 MB
 pg_catalog | pg_proc        | 864 kB
 pg_catalog | pg_depend      | 744 kB
 pg_catalog | pg_attribute   | 536 kB
 public     | history        | 344 kB
```


Monitorování databáze

Monitorování podílu nedostupných záznamů

```
  -- funkce pgstattuple a pgstatindex z balícku pgstattuple
postgres=# \x
Expanded display is on.
postgres=# select * from pgstattuple('foo');
-[ RECORD 1 ]------+------
table_len          | 8192
tuple_count        | 0
tuple_len          | 0
tuple_percent      | 0
dead_tuple_count   | 2
dead_tuple_len     | 93
dead_tuple_percent | 1.14
free_space         | 8064
free_percent       | 98.44

postgres=# select * from pgstatindex('foox');
-[ RECORD 1 ]------+-----
version            | 2
tree_level         | 0
index_size         | 8192
root_block_no      | 1
internal_pages     | 0
leaf_pages         | 1
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 0.93
leaf_fragmentation | 0
```

Instalace doplňků

Postup

Poznámka

PostgreSQL je navrhován minimalisticky, tj. co nemusí být částí jádra, přesouvá se do rozšiřujících (contrib) modulů. Příkladem může být *tsearch2*, *fuzzystrmatch*, *pgcrypto* nebo *pgbench*. Ukázka obsahuje instalaci doplňku *orafce*, což je sada funkcí inspirovaná knihovnou RDBMS Oracle.

1. Pokud jste v adresáři contrib `make; make install`,
2. V privátním adresáři `make USE_PGXS=1; make USE_PGXS=1 install`
3. Z podadresáře postgresql share/contrib načíst soubor orafce.sql jako superuser
   `psql db < /usr/local/pgsql/share/contrib/orafce.sql`
4. U některých doplňků je nutné explicitně zpřístupnit nové funkce příkazem `GRANT`. Tímto způsobem určujeme, kdo smí nové funkce používat.

pgbench

`pgbench` je jednoduchá klientská aplikace, která generuje unifikovanou zátež serveru PostgreSQL. V podstatě nelze jednoznačně interpretovat výslednou hodnotu udávající počet realizovaných zákaznických transakcí za vteřinu.

Instalace doplňků

Testování

Regresní test

Každý doplněk obsahuje sadu testu umožňujících alespoň rámcově ověřit funkčnost na dané platformě. Pokud selže regresní test, reportujte popis chyby a platformy správci testovaného doplňku.

```
make USE_PGXS installcheck
```

Postup při přechodu na novou verzi

Plán

Pozn.

Při minoritní změně (změna za destinou tečkou) jsou verze datově kompatibilní (tudíž stačí pouze změnit binární soubory). Při aktualizaci mezi nekompatibilními verzemi je třeba provést dump databáze. V případě, kdy budeme migrovat na novější verzi je doporučováno, aby dump byl provedem příkazem `pg_dump` z novější verze (tj. nejdříve aktualizujeme klientské aplikace, poté server ... všechny PostgreSQL aplikace se dokážou připojit k serveru jiné verze (pouze dostanete varování)). Aktuální verze dokáží načíst dump z `pg_dump` verze 7.0.

Tip

Migraci můžeme zkrátit paralelním provozem nového serveru na jiném portu a migrací přes rouru:

```
pg_dumpall -p 5432 | psql -d postgres -p 6543
```

Ověřte si, že Vám nikdo v té době nepřistupuje k SQL serverům.

Postup při přechodu na novou verzi

Možné problémy

Migraci vždy testujte

- Ještě před dumpem v novější verzi vytvořete zálohu v aktuální verzi. Není zaručeno, že se starší klient dokáže připojit k novějšímu serveru, tj. bez této zálohy by cesta zpět mohla být obtížná.
- Prostudujte si odpovídající RELEASE NOTES.
- V ideálním případě máte k dispozici UNIT testy.
- Upgrade příliš neodkládejte, je jistější udělat několik menších kroků, než jeden skok. Nové verze PostgreSQL vycházejí jednou ročně. Zaručeně podporované jsou dvě verze zpátky (oprava chyb, bezpečnostní záplaty, atd). Jako optimální je upgrade každe dva roky.

Kde mohou nastat problémy?

- přísnější kontrola UTF8 ve verzi 8.2 (tj. předchozí dump může být považován za nekorektní). Oprava - použití filtru `iconv`.
- příliš stará verze PostgreSQL (7.3 a starší) - provádějte upgrade inkrementálně.

Tip

Načítání dump souboru lze urychlit dočasným přenastavením níže určených konfiguračních parametrů. Tyto hodnoty jsou určené pouze pro načítání dat (předpokládají jednouživatelský režim), které lze v případě problémů opakovat, a **nejsou** určené pro produkční režim.

```
fsync = off
shared_buffers = [1/3 of memory]
wal_buffers = 1MB
checkout_timeout = 1h
checkpoint_segments = 300
maintenance_work_mem = [1/3 of memory]
```

Nezapomeňte, že pro upgrade potřebujete minimálně 2x tolik volného prostoru, jako je aktuální velikost datového adresáře. **Po dokončení importu nezapomeňte konfiguraci nastavit na provozní hodnoty. fsync = off může vést ke ztrátě dat.**

PostgreSQL 8.2 efektivně

Administrace

Pavel Stěhule

http://www.pgsql.cz

25. 2. 2007