PIC
port d'informació
científica

Vacuum

- Records deleted or obsoloted by an update are not reclaimed as free space and cannot be reused
- Vacuum claims that space for the system to reuse, but does not reclaim the space
    - No lock, can be done in production
- Vacuum full actually reclaims the space
    - Gets an exclusive lock, not production
    - But backup / restore seems a more efficient procedure than vacuum full

PIC
port d'informació científica

Partitioning

- What is it?
    - Splitting one large logical table into various small physical tables
- What for?
    - Certain queries can be optimized
        - various smaller indexes
        - full scans only on subtables
    - Operations on a whole partition are efficient and easy
        - droping a whole partition
        - copying it to another media
    - Seldom used data can be put into cheaper or slower tablespaces

PIC
port d'informació
científica

Partitioning

- Needed?
  - Our dCache instance has an ever-growing database that is interesting to keep: BillingDB
  - At PIC, it is around 14 Gb, and doorinfo by itself is more than 8 Gb. That's only for 2008.
  - We use Brian Bockleman's GraphTool package to do some nice plots, and would like to keep that information available.

PIC port d'informació científica

Partitioning: Implementation

- Create a *master* table
  - This table will contain no data, have no checks, and no indexes

```
CREATE TABLE doorinfo (
    datestamp timestamp without time zone,  (discrimination field)
    cellname character varying,
    "action" character varying,
    "owner" character varying,  (index)
    mappeduid numeric,
    mappedgid numeric,
    client character varying,
    "transaction" character varying,  (index)
    pnfsid character varying,
    connectiontime numeric,
    queuedtime numeric,
    errorcode numeric,
    errormessage character varying,
    path character varying
);
```

PIC
port d'informació
científica

Partitioning: Implementation

- Create several *child* tables, that each inherit from the *master* table. Each of these tables is refered as *partition*

```
create table doorinfo_y2007 ( ) inherits (doorinfo);
create table doorinfo_y2008 ( ) inherits (doorinfo);
create table doorinfo_y2009 ( ) inherits (doorinfo);
create table doorinfo_y2010 ( ) inherits (doorinfo);
```

Partitioning: Implementation

- Add table constraints to define the allowed keys in each partition

```
create table doorinfo_y2008 ( check (timestamp >= DATA '2008-01-01' AND
   timestamp < DATE '2008-12-31') ) inherits (doorinfo);
create table doorinfo_y2009 ( check (timestamp >= DATA '2009-01-01' AND
   timestamp < DATE '2009-12-31') ) inherits (doorinfo);
create table doorinfo_y2010 ( check (timestamp >= DATA '2010-01-01' AND
   timestamp < DATE '2010-12-31') ) inherits (doorinfo);
create table doorinfo_y2011 ( check (timestamp >= DATA '2011-01-01' AND
   timestamp < DATE '2011-12-31') ) inherits (doorinfo);
```

Partitioning: Implementation

- Create indexes on the key columns if needed
  - We have two indexes on doorinfo, *owner* and *transaction*

```
create index doorinfo_y2007_owner on doorinfo_y2007 (owner);
create index doorinfo_y2007_transaction on doorinfo_y2007 (transaction);
create index doorinfo_y2008_owner on doorinfo_y2008 (owner);
create index doorinfo_y2008_transaction on doorinfo_y2008 (transaction);
```

PIC
port d'informació
científica

Partitioning: Implementation

- Define a trigger to redirect data inserted into the master table

```
create or replace function doorinfo_insert_trigger()
returns trigger as $$
begin
  insert into doorinfo_y2009 values (new.*);
  return null;
end;
$$
language plpgsql;

create trigger insert_doorinfo_trigger
  before insert on doorinfo
  for each row execute procedure doorinfo_insert_trigger();
```

- With this approach, we will need to update the function each year

Partitioning: Implementation

- You can define a more complicated trigger that automatically selects the correct partition

```
create or replace function doorinfo_insert_trigger()
return trigger as $$
begin
  if (new.datestamp >= DATE '2007-01-01' and new.logdate < DATE '2007-12-31' ) then
    insert into doorinfo_y2007 values (new.*);
  elseif (new.logdate >= DATE '2008-01-01' and 'new.logdate < DATE '2008-12-31') then
    insert into doorinfo_y2008 values (new.*);
  else
    raise exception 'date out of range';
  end if;
  return null;
end;
$$
language plpgsql;
```

PIC
port d'informació
científica

Partitioning: Notes

- Ensure that the constraint_exclusion parameter is enabled in postgresql.conf
- Constraint exclusion
    - Query optimization technique
    - Allows planner to scan only one child table instead of all the tables if a constant is used in the query

```
select * from doorinfo where datestamp >= DATE '2008-05-21';
```

PIC
port d'informació
científica

Partitioning: Notes

- No automatic way to verify the CHECK constraints
- VACUUM & ANALYZE commands have to be executed on the subtables as well
- Constraint exclusion
    - Query optimization technique
    - Allows planner to scan only one child table instead of all the tables

PIC
port d'informació
científica

Partitioning: Notes

- To migrate a table to other media, you can use ALTER TABLE name SET TABLESPACE new_tablespace;
  - That will not migrate indexes; move them by hand

PIC
port d'informació
científica

Warm Standby

- What is it?
  - It is a technique for being able to have an up-to-date database that mirrors de production database
- What it is for
  - Up-to-the-minute (constant) backups of the database
  - Fast recovery
- Involves playing with the WAL files

Warm Standby: Write Ahead Log

- Keeping logs of every change on the data pages
- Changes to data have to be written after they have been logged and the log is safe on disk
- This allows roll-forward / redo
  - We can replay what happened before a crash
- This allows committing transactions without needing to flush data to disk
  - Less I/O

PIC
port d'informació
científica

Warm Standby: Write Ahead Log

- WAL files live in the pg_xlog directory
- Postgres uses WAL files if there is a crash in the postgres server process
- But those same WAL files can also be used for
  - Continous Archiving (backups up to the minute)
  - Point-in-time Recovery (time machine)

Warm Standby: Continous Archiving

- Setup a script that copies WAL files modifying some parameters in postgresql.conf
  - archive_mode
  - archive_command
  - archive_timeout
- We can keep a backup of the file-system level database and the WAL files, and replay them as backup recovery procedure
  - Simulating a crash – postgresql won't be able to tell the diference

Warm Standby: Continous Archiving

- We can also have a secondary server continously in crash-recovery mode recovering from WALs sent from the primary server
  - also called file-based log shipping
- That means we will have a live backup of the data, with synchronization up to the minute
- If you also want non-live backups, you can backup from the secondary
  - reduced load on the primary server

PIC
port d'informació
científica

Warm Standby: Procedure

- Set up primary and standby systems as near identically as possible, including two identical copies of PostgreSQL at the same release level.
- Set up continuous archiving from the primary to a WAL archive located in a directory on the standby server.

PIC
port d'informació científica

Warm standby: Procedure

- Make a base backup of the primary server, and load this data onto the standby.
- Begin recovery on the standby server from the local WAL archive, using a recovery.conf that specifies a restore_command that is looping waiting for new WAL files to be shipped.

PIC port d'informació científica

Warm standby

- This smells like High Availability, but it is not – it is Online Backup / Fast Recovery
- For high availability, we need the capability of do Failover
- This can be done
  - Via an external tool, i.e., Heartbeat
  - Through other aproaches, like Slony

PIC
port d'informació
científica

Slony

- What is it?
  - It is a master-slave replication system to replicate large databases to a reasonably limited number of slave systems
  - Providing cascading and automatic failover
  - External package not provided by postgres developers
  - Can be installed by compiling from source code or via provided rpms

Slony: Concepts

- Master
  - Node that can read and write the database
- Slave
  - Node that can only read the database
- Subscription
  - Nodes subscribe to a set of tables; master is the *provider* while slave are the *subscriptors*
- Is it based on record-based log shipping
  - slaves have each modified record shipped to them so they have an almost identical copy of the master

PIC
port d'informació
científica

Slony: Concepts

- New slony (2.0) only runs with Postgres 8.3
  - Uses new features from 8.3

PIC
port d'informació
científica

Backups vs Warm Standby vs Slony

- Backups: Disaster Recovery
  - Downtime: reaction time + some hours
  - Data loss: some hours
    - can be lowered highly if you also archive WAL files
- Warm standby: Fast Recovery
  - Downtime: reaction time + some minutes
    - can be lowered with an automatic failover tool like heartbeat
  - Data loss: up to 1 minute
- Slony: High Availability
  - Downtime: reaction + some minutes
    - Can be automated with failover – order of seconds (lag)
  - Data loss: failover – some transactions