crunchy data

Statistical Analysis in PostgreSQL with PL/R

Joe Conway
joe.conway@crunchydata.com
mail@joeconway.com

Crunchy Data
March 24, 2018

Intro to PL/R

What is R?

- An open source language and environment for statistical computing and graphics. . .

What is PostgreSQL?

- PostgreSQL is a powerful, open source object-relational database system. It has more than 30 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

What is PL/R?

- R Procedural Language Handler for PostgreSQL. Enables user-defined SQL functions to be written in the R language. Actively developed since early 2003.

http://www.postgresql.org
http://www.joeconway.com/plr

crunchy data

Pros

- Leverage people's knowledge and skills
  - statistics/math, database, web are distinct specialties
- Leverage hardware
  - server better able to handle analysis of large datasets
- Processing/bandwidth efficiency
  - why send large datasets across the network?
- Consistency of analysis
  - ensure analysis done consistently once vetted
- Abstraction of complexity
  - keep system understandable and maintainable
- Leverage R
  - rich core functionality and huge ecosystem

crunchy data

Cons

- PostgreSQL user
  - Slower than standard SQL aggregates and PostgreSQL functions for simple cases
  - New language to learn
- R user
  - Debugging more challenging than working directly in R
  - Less flexible for ad hoc analysis
  - New language to learn

crunchy data

Creating PL/R Functions

- A little different from standard R functions

```
func_name <- function(myarg1 [,myarg2...]) {
  function body referencing myarg1 [, myarg2 ...]
}
```

- But similar to other PostgreSQL PLs

```
CREATE OR REPLACE FUNCTION func_name(arg-type1 [, arg-type2 ...])
RETURNS return-type AS $$
    function body referencing arg1 [, arg2 ...]
$$ LANGUAGE 'plr';

CREATE OR REPLACE FUNCTION func_name(myarg1 arg-type1
                                     [, myarg2 arg-type2 ...])
RETURNS return-type AS $$
    function body referencing myarg1 [, myarg2 ...]
$$ LANGUAGE 'plr';
```

crunchy data

Example of Use

```
CREATE EXTENSION plr;

CREATE OR REPLACE FUNCTION test_dtup(OUT f1 text, OUT f2 int)
RETURNS SETOF record AS $$
  data.frame(letters[1:3],1:3)
$$ LANGUAGE 'plr';

SELECT * FROM test_dtup();
 f1 | f2
----+----
 a  |  1
 b  |  2
 c  |  3
(3 rows)
```

crunchy data

Highlighted Features

- RPostgreSQL Compatibility
- Custom SQL aggregates
- R object $\Rightarrow$ bytea

crunchy data

RPostgreSQL Compatibility

- Allows prototyping using R, move to PL/R for production
- Queries performed in current database
- Driver/connection parameters ignored; dbDriver, dbConnect, dbDisconnect, and dbUnloadDriver are no-ops

```
dbDriver(character dvr_name)
dbConnect(DBIDriver drv, character user, character password,
          character host, character dbname, character port,
          character tty, character options)
dbSendQuery(DBIConnection conn, character sql)
fetch(DBIResult rs, integer num_rows)
dbClearResult (DBIResult rs)
dbGetQuery(DBIConnection conn, character sql)
dbReadTable(DBIConnection conn, character name)
dbDisconnect(DBIConnection conn)
dbUnloadDriver(DBIDriver drv)
```

crunchy data

RPostgreSQL Compatibility Example

- PostgreSQL access from R

```
tsp_tour_length<-function() {
  require(TSP); require(fields); require(RPostgreSQL)

  drv <- dbDriver("PostgreSQL")
  conn <- dbConnect(drv, user="postgres", dbname="plr", host="localhost")
  sql.str <- "select id, st_x(location) as x, st_y(location) as y,
              location from stands"
  waypts <- dbGetQuery(conn, sql.str)
  dist.matrix <-  rdist.earth(waypts[,2:3], R=3949.0)
  rtsp <- TSP(dist.matrix)
  soln <- solve_TSP(rtsp)
  dbDisconnect(conn)
  dbUnloadDriver(drv)

  return(attributes(soln)$tour_length)
}
```

crunchy data

RPostgreSQL Compatibility Example

- Same function from PL/R

```
CREATE OR REPLACE FUNCTION tsp_tour_length() RETURNS float8 AS $$
  require(TSP); require(fields); require(RPostgreSQL)

  drv <- dbDriver("PostgreSQL")
  conn <- dbConnect(drv, user="postgres", dbname="plr", host="localhost")
  sql.str <- "select id, st_x(location) as x, st_y(location) as y,
              location from stands"
  waypts <- dbGetQuery(conn, sql.str)
  dist.matrix <-  rdist.earth(waypts[,2:3], R=3949.0)
  rtsp <- TSP(dist.matrix)
  soln <- solve_TSP(rtsp)
  dbDisconnect(conn)
  dbUnloadDriver(drv)

  return(attributes(soln)$tour_length)
$$ LANGUAGE 'plr' STRICT;
```

crunchy data

RPostgreSQL Compatibility Example (cont.)

- Output from R

```
tsp_tour_length()
[1] 2804.581
```

- Same function from PL/R

```
SELECT tsp_tour_length();
 tsp_tour_length
------------------
 2804.58129355858
(1 row)
```

crunchy data

Aggregates

- Aggregates in PostgreSQL are extensible via SQL commands
- State transition function and possibly a final function are specified
- Initial condition for state function may also be specified

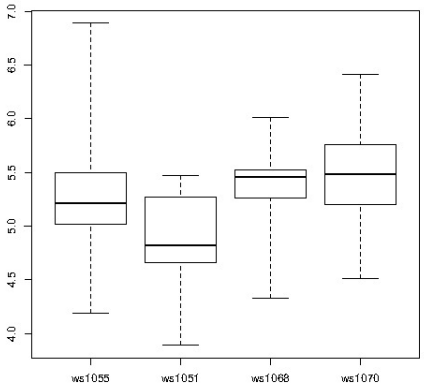
crunchy data

Aggregates Example

```
CREATE OR REPLACE FUNCTION r_quartile(ANYARRAY) RETURNS ANYARRAY AS $$
  quantile(arg1, probs = seq(0, 1, 0.25), names = FALSE)
$$ LANGUAGE 'plr';

CREATE AGGREGATE quartile (ANYELEMENT) (
  sfunc = array_append, stype = ANYARRAY,
  finalfunc = r_quantile, initcond = '{}');

SELECT workstation, quartile(id_val) FROM sample_numeric_data
WHERE ia_id = 'G121XB8A' GROUP BY workstation;
 workstation |            quantile
-------------+--------------------------------
 1055        | {4.19,5.02,5.21,5.5,6.89}
 1051        | {3.89,4.66,4.825,5.2675,5.47}
 1068        | {4.33,5.2625,5.455,5.5275,6.01}
 1070        | {4.51,5.1975,5.485,5.7575,6.41}
(4 rows)
```

crunchy data

Aggregates Example - Quartile Boxplot Output



crunchy data

Stock Data Example

- get Hi-Low-Close data from Yahoo for any stock symbol
- plot with Bollinger Bands and volume
- requires extra R packages - from R:

```
install.packages(c('xts','Defaults','quantmod','cairoDevice','RGtk2'))
```

crunchy data

Stock Data Example

```
CREATE OR REPLACE FUNCTION plot_stock_data(sym text) RETURNS bytea AS $$
  library(quantmod); library(cairoDevice); library(RGtk2)

  pixmap <- gdkPixmapNew(w=500, h=500, depth=24)
  asCairoDevice(pixmap)

  getSymbols(c(sym))
  chartSeries(get(sym), name=sym, theme="white",
                TA="addVo();addBBands();addCCI()")

  plot_pixbuf <- gdkPixbufGetFromDrawable(NULL, pixmap,
                pixmap$getColormap(),0, 0, 0, 0, 500, 500)
  buffer <- gdkPixbufSaveToBufferv(plot_pixbuf, "jpeg",
                character(0),character(0))$buffer

  return(buffer)
$$ LANGUAGE plr;
```

crunchy data

Stock Data Example

- Need screen buffer on typical server:

```
Xvfb :1 -screen 0 1024x768x24
export DISPLAY=:1.0
```

- Calling it from PHP for CYMI

```
<?php
$dbconn = pg_connect("...");
$rs = pg_query( $dbconn, "select plr_get_raw(plot_stock_data('CYMI'))");
$hexpic = pg_fetch_array($rs);
$cleandata = pg_unescape_bytea($hexpic[0]);

header("Content-Type: image/png");
header("Last-Modified: " .
       date("r", filectime($_SERVER['SCRIPT_FILENAME'])));
header("Content-Length: " . strlen($cleandata));
echo $cleandata;
?>
```

crunchy data

Stock Data Example - Output

crunchy data

Seismic Data Example

- Timeseries, waveform data
- Stored as array of floats recorded during seismic event at a constant sampling rate
- Available from online sources in individual file for each event
- Each file has about 16000 elements

crunchy data

Seismic Data Example

- Load 1000 seismic events

```
DROP TABLE IF EXISTS test_ts;
CREATE TABLE test_ts (dataid bigint NOT NULL PRIMARY KEY,
                      data double precision[]);
CREATE OR REPLACE FUNCTION load_test(int) RETURNS text AS $$
  DECLARE
    i    int; arr  text; sql  text;
  BEGIN
    arr := pg_read_file('array-data.csv', 0, 500000);
    FOR i IN 1..$1 LOOP
      sql := $i$INSERT INTO test_ts(dataid,data) VALUES ($i$ || i || $i$,'{$i$ || arr || $i$}')$i$;
      EXECUTE sql;
    END LOOP;
    RETURN 'OK';
  END;
$$ LANGUAGE plpgsql;

SELECT load_test(1000);
 load_test
-----------
 OK
(1 row)
Time: 37336.539 ms
```

crunchy data

Seismic Data Example

- Load 1000 seismic events (alternate method)

```
DROP TABLE IF EXISTS test_ts_obj;
CREATE TABLE test_ts_obj (
  dataid serial PRIMARY KEY,
  data bytea
);

CREATE OR REPLACE FUNCTION make_r_object(fname text) RETURNS bytea AS $$
  myvar<-scan(fname,sep=",")
  return(myvar);
$$ LANGUAGE 'plr' IMMUTABLE;

INSERT INTO test_ts_obj (data)
SELECT make_r_object('array-data.csv')
FROM generate_series(1,1000);

INSERT 0 1000
Time: 12166.137 ms
```

crunchy data

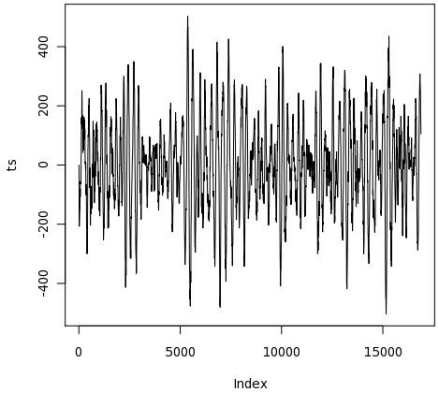
Seismic Data Example

- Plot the waveform

```
CREATE OR REPLACE FUNCTION plot_ts(ts double precision[]) RETURNS bytea AS $$
  library(quantmod)
  library(cairoDevice)
  library(RGtk2)

  pixmap <- gdkPixmapNew(w=500, h=500, depth=24)
  asCairoDevice(pixmap)

  plot(ts,type="l")
  plot_pixbuf <- gdkPixbufGetFromDrawable(NULL, pixmap,
                                          pixmap$getColormap(),
                                          0, 0, 0, 0, 500, 500)
  buffer <- gdkPixbufSaveToBufferv(plot_pixbuf,
                                   "jpeg",
                                   character(0),
                                   character(0))$buffer
  return(buffer)
$$ LANGUAGE 'plr' IMMUTABLE;

SELECT plr_get_raw(plot_ts(data)) FROM test_ts WHERE dataid = 42;
```

crunchy data

Seismic Data Example - Waveform Output



crunchy data

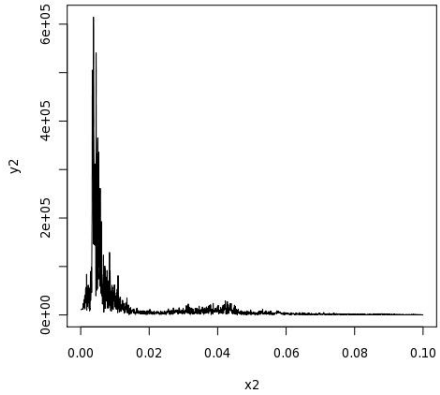
Seismic Data Example

- Analyze the waveform

```
CREATE OR REPLACE FUNCTION plot_fftps(ts bytea) RETURNS bytea AS $$
  library(quantmod); library(cairoDevice); library(RGtk2)

  fourier<-fft(ts)
  magnitude<-Mod(fourier)
  y2 <- magnitude[1:(length(magnitude)/10)]
  x2 <- 1:length(y2)/length(magnitude)
  mydf <- data.frame(x2,y2)

  pixmap <- gdkPixmapNew(w=500, h=500, depth=24)
  asCairoDevice(pixmap)

  plot(mydf,type="l")
  plot_pixbuf <- gdkPixbufGetFromDrawable(NULL, pixmap, pixmap$getColormap(),
                                                        0, 0, 0, 0, 500, 500)
  buffer <- gdkPixbufSaveToBufferv(plot_pixbuf, "jpeg", character(0),
                                                        character(0))$buffer
  return(buffer)
$$ LANGUAGE 'plr' IMMUTABLE;

SELECT plr_get_raw(plot_fftps(data)) FROM test_ts_obj WHERE dataid = 42;
```

crunchy data

Seismic Data Example - Waveform Analysis Output



crunchy data

Questions?

Thank You!
joe.conway@credativ.com
mail@joeconway.com

crunchy data