PostgreSQL and PL/Python

Daniel Swann
Matt Small
Ethan Holly
Vaibhav Mohan

Our Instance

- Amazon AWS 64-bit CentOS large instance
  - 8GB RAM
  - 800GB storage volume

- Installed Postgres, PL/Python, and psycopg2 for Python <> Postgres communication

What is PostgreSQL?

- ORDBMS
- Transactional, ACID, SQL:2011
- Doesn’t employ Parallel Processing (like greenplum)
- Available for many platforms including Linux, FreeBSD, windows and Mac OS X

MadLib and GreenPlum

- Madlib is a library for scaled database analytics
  - However, it could not process 2D arrays for matrix multiplication
  - Decided to use PL/Python instead for in-database operations

- Madlib needed 4.1 or greater version of Greenplum to function
  - 2.5 was the only free version available
  - Therefore, we decided against creating a Greenplum cluster

Data Ingestion

- Discussed multiple schemas for the database
- Finally concluded that storing the matrix elements individually, with their indices as attributes, was the way to go
  - INSERT INTO large_dataset VALUES(row, col, element);
- Needed to modify postgres' memory usage limit in the kernel
  - 3GB of swap_buffer space
  - Ran into prob as Postgres service wasn't starting after this change
  - 5 GB allowed in sysctl.conf shared memory to overcome this
  - 100 GB, 200 GB, 800 GB DIsk

Script for ingesting data in PostgreSQL

First attempt:

```
drop table if exists data;
create table data(value float);
copy data from '/root/a.csv' DELIMITERS ' ' CSV;
```

Script for ingesting data in PostgreSQL

load.py
load_new.py

load.py

```
with open('/root/datasets/bigdataset.csv') as file:
        r=0
        for line in file:
                cur = conn.cursor()
                c=0
                for chunk in line.split():
                        st = "INSERT INTO Large_Dataset VALUES({0},{1},{2})".format(r,c,chunk)
                        if r%100==0:
                                print st
                        cur.execute(st)
                        c+=1
                r+=1
                cur.close()
                conn.commit()
    conn.commit()
```

load_new.py

```
with open('/dev/sdi/bigdataset.csv') as file:
        while c:
            try:
                c=file.read(1)
            except EOFError:
                print "Done."
                exit(1)
            if c==' ':
                if row>21840:
                    cur.execute(st.format(row,col,float(num)))
                    col+=1
                    num=''
            elif c=='\n':
                if row>21840:
                    print str(row) + " " + str(col) + " " + str(num)
                    cur.execute(st.format(row,col,float(num)))
                    conn.commit()
                    col=0
                print row
                row+=1
                num=''
            else:
                num+=c
    cur.close()
```

Code Interfaces

- c++ abstraction
  - not well documented
- R
  - setup was complicated
- PlPython (most promising)
  - write SQL functions using Python and Python libraries (like numpy)

What is PL/Python?

- The PL/Python procedural language allows PostgreSQL functions to be written in the Python language.
- <timesTwo.sql_in>
  CREATE FUNCTION timesTwo(x double precision) RETURNS double precision
  AS $$

  return x * 2;

  $$ LANGUAGE plpythonu STRICT VOLATILE;

- psql dbname <timesTwo.sql_in>

Haar Wavelet Transform

- As our data our are stored as <row,col,val> tuples, the sparse matrix multiplication in the Haar transform was efficient to perform

- For each element of the result matrix, we only select the two relevant tuples for that particular computation.
  - This may not be efficient enough. The other Postgres group created subtables as intermediate step in their algorithm so that each select would not have to scan the whole database.

Haar Wavelet Transform

```
FOR j in 0..(half_size-1) LOOP
        FOR i in 0..(size-1) LOOP
            EXECUTE format('SELECT value FROM %I WHERE row=2*%s AND col=%s', in_table, j, i) INTO a;
            EXECUTE format('SELECT value FROM %I WHERE row=2*%s+1 AND col=%s', in_table, j, i) INTO b;
            haar_val_1 := compute_transform(a, b, FALSE);
            haar_val_2 := compute_transform(a, b, TRUE);
            EXECUTE format('INSERT INTO %I VALUES (%s, %s, %s)', out_table, j, i, haar_val_1);
            EXECUTE format('INSERT INTO %I VALUES (%s+%s, %s, %s)', out_table, j, half_size, i, haar_val_2);
        END LOOP;
    END LOOP;
```

Thresholding

- INSERT INTO thresholded_data
  SELECT (row, col, threshold(value)) FROM transformed_data;

```
CREATE OR REPLACE FUNCTION threshold(in_table VARCHAR(30), out_table VARCHAR(30),
threshold float)
RETURNS VOID AS $$
BEGIN
	EXECUTE format('CREATE TABLE %I AS (SELECT * FROM %I WHERE value > %s)', out_table,
in_table, threshold);
END;
$$ LANGUAGE plpgsql;"""
```

Data Compression

- Compression was implicit to our data format.
  - When extracting our data from the database, we simply select where value != 0, and
-

Benchmarking Haar Transform

- Small Data: Around 2-5 hours
- Medium and Large Data: Ran for more than 5 days but couldn’t finish.

Benchmarking threshold

- Small data: threshold value = 0.376475
  - time : 25 sec
- Could not extract csv using pl/python

Issues and Problems

- Originally wanted to use MADlib for performing our matrix operations.
  - MADlib was expecting matrices in an odd data format… wait, that was a good format!
- Single rows/columns don’t fit in memory

Waxing Philosophical

- The greatest software engineering advances are those that replace specific cleverness with general cleverness.
  - The point of these DBMS systems is to provide such a solution.
- The only successful projects ended up scrapping old designs for clever ones.
  - These seem to be unsuccessful so far.