Database considerations

v1.0 - 2017-11-24

Some investigation into loading siri-vm data into a Postgres database.

Bulk loading data

There are several strategies for bulk uploading data:

- Using the Postgres 'copy' protocol via `pgloader`. `pgloader` will where possible optimise uploads by doing them in parallel and will optionally drop indexes before starting and rebuild them in parallel once finished
- Writing a sequence of SQL INSERT commands from a script and executing them via `psql`
- Issuing SQL INSERT commands from a script via a Postgres interface such as Python's psycopg2

There are also two different strategies for populating columns with copies of data that also appears in the underlying JSON representation of the SIRI-VM data:

- Upload just the JSON data and subsequently populate the additional columns with `UPDATE <database> SET` commands
- Extract values for the additional columns in the loader script and upload these and the JSON record

Simple schema

This schema just contains the individual SIRI-VM records as JSON, but includes functional indexes built on potentially interesting fields within the JSON:

```
CREATE TABLE siri_vm_simple_test (
    id                      SERIAL PRIMARY KEY,
    info                    JSONB
    );

CREATE INDEX siri_vm_simple_test_info ON siri_vm_simple_test USING GIN (info);
CREATE INDEX siri_vm_simple_test_acp_id ON siri_vm_simple_test ((info->>'acp_id'))
;
CREATE INDEX siri_vm_simple_test_acp_lat ON siri_vm_simple_test ((info->>'acp_lng'
));
CREATE INDEX siri_vm_simple_test_acp_lng ON siri_vm_simple_test ((info->>'acp_lat'
));
CREATE INDEX siri_vm_simple_test_acp_ts ON siri_vm_simple_test (to_timestamp((info
->>'acp_ts')::double precision));
```

An obvious drawback to this is that we only end up with position expressed as separate latitude and longitude, and as text strings at that.

'Complex' schema

This schema still contains the individual SIRI-VM records as JSON, but promotes all the interesting information into separate columns and indexes them:

```
CREATE TABLE siri_vm_complex_test (
    id                      SERIAL PRIMARY KEY,
    acp_id                  TEXT,
    location4d              GEOGRAPHY(POINTZM,4326),
    acp_ts                  TIMESTAMP WITH TIME ZONE,
    info                    JSONB
    );

CREATE INDEX siri_vm_complex_test_acp_id ON siri_vm_complex_test (acp_id);
CREATE INDEX siri_vm_complex_test_location4d ON siri_vm_complex_test USING GIST (l
ocation4d);
CREATE INDEX siri_vm_complex_test_acp_ts ON siri_vm_complex_test (acp_ts);
CREATE INDEX siri_vm_complex_test_info ON siri_vm_complex_test USING GIN (info);
```

Loading results

All experiments were run on a MacBook Pro with a dual-core 2.9 GHz Intel Core i5 with 16GB of memory running Postgres 10.1/PostGIS2.4.0 locally. All scripting used Python 3.6.2, with psycopg2 2.7.3.1 where needed.

All tests started by TRUNCATING the relevant table. Indexes were dropped before loading started and

recreated once finished (by `pgloader` where possible). Tests were run 5 times.

Simple Schema

Extract just sirivm_json data to CSV, load this

```
psql -c "truncate siri_vm_simple_test" acp
./siri-vm-to-simple-csv.py ../data/sirivm_json/data_bin/2017/10/27/ | pgloader sir
i-vm-to-simple-database.load
```

Mean: 171 sec (2m51s); Max: 176 sec; Min: 169 sec

Extract and upload just sirivm_json files in a script

```
psql -c "truncate siri_vm_simple_test" acp
psql -f drop_simple_indexes.sql acp
./siri-vm-simple-insert.py ../data/sirivm_json/data_bin/2017/10/27/
psql -f add_simple_indexes.sql acp
```

Mean: 211 sec (3m31s); Max: 230 sec; Min: 196 sec

'Complex' schema

Extract just sirivm_json data to CSV, load this and run an UPDATE

This UPDATE statement:

```
UPDATE siri_vm_complex_test set
    acp_id = info->>'acp_id',
    location4d = ST_GeogFromText('SRID=4326;POINT(' || (info->>'acp_lng') || ' ' |
| (info->>'acp_lat') || ' 0 ' || (info->>'acp_ts') || ')'),
    acp_ts = to_timestamp((info->>'acp_ts')::double precision);
```

```
psql -c "truncate siri_vm_complex_test" acp
psql -f drop_complex_indexes.sql acp
./siri-vm-to-simple-csv.py ../data/sirivm_json/data_bin/2017/10/27/ | pgloader sir
i-vm-to-complex-database-with-update.load
psql -f add_complex_indexes.sql acp
```

Mean: 199 sec (3m19s); Max: 237 sec; Min: 189 sec;

Extract sirivm_json data in a script and derive additional values, emit as CSV and load this

```
psql -c "truncate siri_vm_complex_test" acp
./siri-vm-to-complex-csv.py ../data/sirivm_json/data_bin/2017/10/27/ | pgloader si
ri-vm-to-complex-database.load
```

Mean: 171 sec (2m51s); Max: 173 sec; Min: 170 sec

Extract sirivm_json data in a script and derive additional values, emit as SQL INSERTS and pipe to `psql`:

```
psql -c "truncate siri_vm_complex_test" acp
psql -f drop_complex_indexes.sql acp
./siri-vm-to-complex-sql.py ../data/sirivm_json/data_bin/2017/10/27/ | psql -q acp
psql -f add_complex_indexes.sql acp
```

Mean: 219 sec (3m39s); Max: 221 sec; Min: 215 sec

Extract and upload sirivm_json and additional values in a script:

```
psql -c "truncate siri_vm_complex_test" acp
psql -f drop_complex_indexes.sql acp
./siri-vm-complex-insert.py siri_vm3 ../data/sirivm_json/data_bin/2017/10/27/
psql -f add_complex_indexes.sql acp
```

Mean: 205 sec (3m25s); Max: 208 sec; Min: 204

Conclusions

Loading CSV data with `pgloader` was the fastest strategy, followed by loading simple CSV and subsequently running UPDATE TABLE to populate additional columns. These were followed by doing the upload from a script, with creating and executing SLQ INSERT statements last. The slowest method took about 30% longer than the fastest.

The schema chosen makes very little difference to the load time.

Appending results

Appending results, particularly in real time, is a different situation from bulk loading them, not least because the option of droping indexes and re-adding them isn't available.

Here are a couple of tests, one for each of the simple and complex schema, that involve loading one day's worth of data by the fastest method above and then loading a further day's data. In each case the subsequent load was done with indexes defined and with a commit after loading the records from each SIRI+VM JSON file.

Simple schema:

```
psql -c "truncate siri_vm_simple_test" acp
./siri-vm-to-simple-csv.py ../data/sirivm_json/data_bin/2017/10/27/ | pgloader sir
i-vm-to-simple-database.load
./siri-vm-simple-insert-commit.py ../data/sirivm_json/data_bin/2017/10/26/
```

Mean: 683 sec (11m23s); Max: 693 sec; Min: 680 sec

Uploading the first day will have taken about 171 sec (see above), so the additional day took about 512 sec (8m32s).

Complex schema

```
psql -c "truncate siri_vm_complex_test" acp
./siri-vm-to-complex-csv.py ../data/sirivm_json/data_bin/2017/10/27/ | pgloader si
ri-vm-to-complex-database.load
./siri-vm-complex-insert-commit.py ../data/sirivm_json/data_bin/2017/10/26/
```

Mean: 668 sec (11m8s); Max: 692 sec; Min: 650 sec

Uploading the first day will have taken about 171 sec (see above), so the additional day took about 497 sec (8m17s).

Conclusions

Loading into the complex schema seems to be slightly faster than the simple one. This could be becasue maintaining the functional indexes based on data in the JSON is slightly more complex than diirectly indexing columns.