Streaming replication between database engines via Tcl

PostgreSQL to SQLite in real time
Where we are now

- Caching PostgreSQL databases in Speed Tables
- Super fast shared-memory database
- Limited query ability
  - Only AND operations
  - Only query a single table.
- Kind of hammers the database fetching updates
  - Even with clever multi-level code and incremental fetches
Switching to SQLite

- Caching PostgreSQL databases in Speed Tables SQLite
  - Karl's Tcl code for the cache adapted effectively to SQLite

- Still pretty fast database
  - Slower than Speed Tables for raw searches
  - But much better indexing, and you can add indexes on the fly
  - Potential for being much faster

- Full SQL queries

- But still hammering the server with update requests
But PostgreSQL has this replication mechanism

- Replication slots watch the WAL (write-ahead log)
- Output plugin filters and reformats the output
- External application (pg_recvlogical) connects to DB and dumps the replication stream to stdout
Created a new output plugin based on the sample provided with PostgreSQL

- New output plugin - deltaflood - dumps all the change records as key-value pairs in tab separated value format

- This is very easy to feed into Tcl arrays or dicts
  - First [array set row [split $line "\t"]]
  - Then [subst -nocommands -novariables ...] as needed

- [http://github.com/flightaware/pg-deltaflood](http://github.com/flightaware/pg-deltaflood)
Deltaflood format

_table zzz _xid 88628916 _action delete a fox61
_table zzz _xid 88628916 _action replace a fox49
_table zzz _xid 88628916 _action update a fox61 b hen62
_table zzz _xid 88628917 _action update a fox17 b hen60
_table zzz _xid 88628918 _action insert a fox62 b hen17
_table zzz _xid 88628919 _action update a fox99 b hen38
_table zzz _xid 88628920 _action delete a fox54
_table zzz _xid 88628920 _action replace a fox11
_table zzz _xid 88628920 _action update a fox54 b hen93
_table zzz _xid 88628921 _action update a fox24 b hen78
_table zzz _xid 88628922 _action update a fox68 b hen76
_table zzz _xid 88628923 _action update a fox83 b hen51

_action:
insert  Insert a row in the database
delete  Delete a row from the database
replace  tag row for replacement
update  Update a row in the database
An extra level of staging

• We will have multiple hosts following the replication stream

• We want to avoid having multiple hosts running separate replication requests
  • Especially since each replication request requires a separate slot.
  • And having a host down would cause PostgreSQL to leak memory.

• We need to be able to restart at a given point in time when a host comes back up.
Daystream

• Flightaware uses an event stream format called "daystream" extensively.

• Stored in daystream files, read through the universal daystream client library

• Files may be local or streamed from another host

• Each line is tagged with a timestamp and sequence number
  • Client library supports starting at any given timestamp and sequence

• Each line is tab-separated key-value pairs - convenient
Daystream

1507507200  _s  0  _table  zzz  _xid  88628908  _action update  a  fox47  b  hen30
1507507200  _s  1  _table  zzz  _xid  88628909  _action update  a  fox97  b  hen11
1507507200  _s  2  _table  zzz  _xid  88628910  _action update  a  fox47  b  hen95
1507507200  _s  3  _table  zzz  _xid  88628911  _action update  a  fox97  b  hen38
1507507200  _s  4  _table  zzz  _xid  88628912  _action update  a  fox15  b  hen51
1507507200  _s  5  _table  zzz  _xid  88628913  _action update  a  fox7  b  hen94
1507507200  _s  6  _table  zzz  _xid  88628914  _action delete  a  fox70
1507507200  _s  7  _table  zzz  _xid  88628915  _action update  a  fox53  b  hen83
1507507200  _s  8  _table  zzz  _xid  88628916  _action delete  a  fox61
1507507200  _s  9  _table  zzz  _xid  88628916  _action replace  a  fox49
1507507200  _s 10  _table  zzz  _xid  88628916  _action update  a  fox61  b  hen62
1507507200  _s 11  _table  zzz  _xid  88628917  _action update  a  fox17  b  hen60
1507507200  _s 12  _table  zzz  _xid  88628918  _action insert  a  fox62  b  hen17
1507507200  _s 13  _table  zzz  _xid  88628919  _action update  a  fox99  b  hen38
1507507200  _s 14  _table  zzz  _xid  88628920  _action delete  a  fox54
1507507200  _s 15  _table  zzz  _xid  88628920  _action replace  a  fox11
1507507200  _s 16  _table  zzz  _xid  88628920  _action update  a  fox54  b  hen93
1507507200  _s 17  _table  zzz  _xid  88628921  _action update  a  fox24  b  hen78
1507507201  _s  0  _table  zzz  _xid  88628922  _action update  a  fox68  b  hen76
1507507201  _s  1  _table  zzz  _xid  88628923  _action update  a  fox83  b  hen51
An extra level of staging

- This is basically deltastream output, plus the timestamp
- So now we have our extra level of staging
- Each host can restart reading where they left off
- Only need to have one replication slot in the database
- Missing hosts don't cause the database to grow
pg_sqlite

- A new command in Pgtcl, pg_sqlite, that can be used to rapidly copy data from PostgreSQL to Sqlite3.

```tcl
set res [$pgdb exec "SELECT * FROM TABLENAME;"]
pg_sqlite $sqlitedb import_postgres_result $res \
    -into tablename \
    -as {col type col type ...} \
    -pkey {col col col}
pg_result $res clear
```

- Optional, only included if Tcl is built with sqlite3 support

- We can rapidly initialize the database using pg_sqlite

- [http://github.com/flightaware/Pgtcl](http://github.com/flightaware/Pgtcl)
  - generic/pgtclSqlite.c
Deltastream and deltamirror

• Straight Tcl applications

• Deltastream reads pg_recvlogical output and feeds it directly into daystream
  • Literally just concatenates time, sequence, and the line read from pg_recvlogical

• Deltamirror reads from daystream and writes the output into sqlite3
  • Maintains a timestamp updated at the end of each transaction, so the replication can be cleanly continued from daystream after a restart.
Bringing it all together

• Read the PostgreSQL schema and save it in PostgreSQL tables for future reference.

• Set up the replication slot to replicate the tables we're interested in.

• Start up deltastream to create the daystream files.

• Then for each new host:

  • Populate the sqlite3 tables using pg_sqlite ...
    import_postgres_result

  • Start replication from daystream files using deltamirror.
One more thing

- Getting the sqlite3 database handle from the Tcl sqlite3 database command requires a bit of parkour
  - As far as I could determine there's no formal API for this.

- Luckily the clientData field for the command has the database object as the first element.

  ```
  struct SqliteDb {
      sqlite3 *db; /* The "real" database structure. MUST BE FIRST */
      // other stuff we don't look at...
  };
  ```

- For safety's sake we need to make sure this is a valid pointer
  - First create a known valid sqlite3 command and save off its objProc
  - Only proceed if the command we're passed uses the same objProc