Towards an Ingres TPC-E Benchmark

Roy Hann

roy.hann@rationalcommerce.com

Actian Hybrid Data Conference
November 9, 2017

November 2016

- I was approached about developing an Ingres benchmark
- I had previously ported an open source clone of the TPC-C benchmark[1] for Ingres
  - Java/JDBC based
  - revealed the some of the pitfalls that await anyone porting a benchmark
- I have wanted a realistic OLTP benchmark for my own purposes for years
- ...so I opened my mouth and said "yes"

[1]BenchmarkSQL

Rational Commerce

November 2017

- A year later and I'm still not quite done
- I've learnt a lot
  - maybe not everything yet
- I can share some useful observations

Rational Commerce

Transaction Processing Performance Council

"The TPC is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry."

Rational Commerce

TPC-E

- TPC-E is an OLTP benchmark
- Supercedes the very naïve TPC-C benchmark
  - approved February 2007
- Portrays the activity of a stock brokerage
- Involves a mix of twelve concurrent transaction types
  - mix of complexity
  - executed on-line or triggered
- Database of 33 tables

Rational Commerce

Our Objectives

- Target: Ingres on Linux
- No (present) intention to publish
- Internal testing to improve Ingres
  - regression testing
  - performance improvement
  - estimate overhead of features
    - e.g. database constraints, SC930 tracing, configuration changes
- Long-term, who knows?
  - nobody is publishing new TPC-E benchmarks

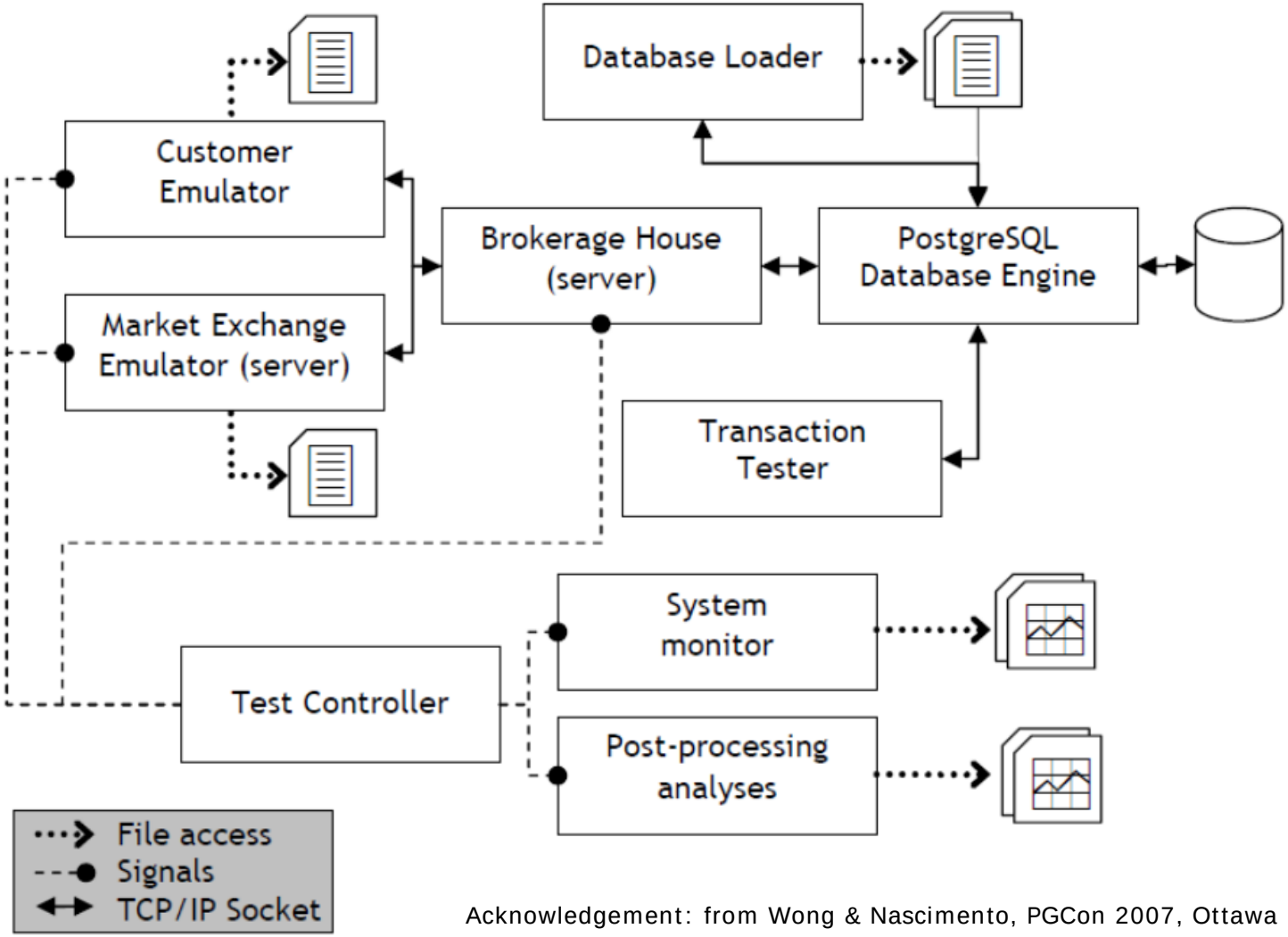
Rational Commerce

DBT-5

- We quickly discovered the open source PostgreSQL DBT-5 benchmark
  - non-compliant implementation of TPC-E
  - developed by Rilson Nascimento and Mark Wong
  - available from sourceforge.net
- Seemed a good way to get a head-start
- Mostly C++
  - practically dictated by the use of C++ for the TPC-supplied framework
  - uses **libpqxx**, the C++ API for PostgreSQL

Rational Commerce

DBT-5 Architecture



Acknowledgement: from Wong & Nascimento, PGCon 2007, Ottawa

DBT-5

- Intended to be platform-independent
  - self-configures to the local environment at build-time
    - turns out to require a massive amount of learning
    - not useful to us; we're interested only in Linux
- Intended to be product-agnostic
  - architecture tries to allow DBMS-specific classes to be overridden
  - not entirely successful
    - probably never attempted till I came along
- But still, an appealing starting point

Rational Commerce

PostgreSQL API

- libpqxx is a low-level API
- roughly similar to using a cross between Ingres OpenAPI and embedded dynamic SQL
  - i.e. laborious, verbose, tedious, hard to read
- PostgreSQL *does* have embedded SQL
  - ECPG
  - pretty nearly identical to Ingres ESQL/C
  - but ECPG doesn't properly support C++
    - unlike Ingres ESQL/C

Rational Commerce

Data Generation

- The data generator is fully working
  - loads data directly to Ingres
- Highlighted an important question
  - of the many possible compliant database designs, which do you choose?
  - how can you know the performance effects?
    - e.g. hardware types versus software types (BIGINT versus DECIMAL(18,0))
  - I chose to use TPC pseudo-type names in my DDL script, and m4 macros to convert to Ingres types
  - **dclgen** creates my C structures

Rational Commerce

TPC Pseudo Types

```
-- Clause 2.2.4.3
CREATE TABLE customer_account
(
   ca_id IDENT_T NOT NULL,
   ca_b_id IDENT_T NOT NULL REFERENCES broker (b_id),
   ca_c_id IDENT_T NOT NULL REFERENCES customer (c_id),
   ca_name CHAR(50),
   ca_tax_st NUM(1) NOT NULL,
   ca_bal BALANCE_T NOT NULL,
   PRIMARY KEY (ca_id)
)
```

Rational Commerce

m4 to Software Types

```
define(`CHAR',`varchar($1)')dnl
define(`NUM',`decimal($*)')dnl
define(`SNUM',`decimal($*)')dnl
define(`ENUM',`decimal($*)')dnl
define(`SENUM',`decimal($*)')dnl
define(`BOOLEAN',`boolean')dnl
define(`DATE',`ansidate')dnl
define(`DATETIME',`timestamp(0)')dnl
define(`BLOB',`byte($1)')dnl
define(`BLOB_REF',`long byte')dnl
define(`IDENT_T',`NUM(11)')dnl
define(`TRADE_T',`NUM(15)')dnl
define(`FIN_AGG_T',`SENUM(15,2)')dnl
define(`S_PRICE_T',`ENUM(8,2)')dnl
define(`S_COUNT_T',`NUM(12)')dnl
define(`S_QTY_T',`SNUM(6)')dnl
define(`BALANCE_T',`SENUM(12,2)')dnl
define(`VALUE_T',`SENUM(10,2)')dnl
```

Rational Commerce

m4 to Hardware Types

```
define(`BESTINT',
    `ifelse(eval(`$1<3'),1,`integer1',
            eval(`$1<5'),1,`integer2',
            eval(`$1<10'),1,`integer4',
            eval(`$1<19'),1,`integer8',
            `decimal($1,0)')')dnl
define(`BESTFLT',
    `ifelse(eval(`$1<8'),1,`float4',
            eval(`$1<17'),1,`float8',
            `decimal($*)')')dnl
define(`CHAR',`ifelse(`$1',`1',`char(1)',`varchar($1)')')dnl
define(`NUM',`ifelse(`$#',`2',`BESTFLT($*)',`BESTINT($1)')')dnl
define(`SNUM',`NUM($*)')dnl
define(`ENUM',`ifelse(`$#',`2',`decimal($*)',`BESTINT($1)')')dnl
define(`SENUM',`ENUM($*)')dnl
...
define(`IDENT_T',`NUM(11)')dnl
...
define(`S_QTY_T',`SNUM(6)')dnl
define(`BALANCE_T',`SENUM(12,2)')dnl
```

Rational Commerce

Coding Issues Encountered

- A couple of quirks of Ingres embedded SQL have emerged
- e.g. the benchmark has queries with a variable number of arguments
  - example: Broker-Volume-Frame-1

```
select broker_name, sum(TR_QTY * TR_BID_PRICE) as volume
from TRADE_REQUEST, SECTOR, INDUSTRY COMPANY, BROKER,
SECURITY
where TR_B_ID = B_ID
  and TR_S_SYMB = S_SYMB
  and S_CO_ID = CO_ID
  and CO_IN_ID = IN_ID
  and SC_ID = IN_SC_ID
  and B_NAME in (:broker_list) and SC_NAME = :sector_name
group by B_NAME
```

Rational Commerce

Coding Issues Encountered

- We would like all query plans to be cached
  - EXEC SQL REPEATED ...
- Can't supply a variable-length list of values at run-time for an IN predicate
- Can't supply both a temporary table and scalar arguments to a DBP
- No way round it
  - have to reparse and reoptimize every query instance
  - no big deal in real life but exaggerated in TPC-E

Rational Commerce

Bugs

- The published DBT-5 code has errors
  - syntax that won't compile
    - at least not with the compiler I'm using
  - many headers not explicitly included
  - bugs
    - presumably fixed, but fixes not published
    - e.g. this, which took three days to track down:

```
CDateTime& CDateTime::operator = (const CDateTime& dt)
{
    m_dayno = dt.m_dayno;
    m_msec = dt.m_msec;
    m_szText = NULL;
    return *this;
}
```

Rational Commerce

In Hindsight

- Developing a TPC-E "flavoured" benchmark from scratch would have been quicker for *me*
  - underestimated the portability-related skills required
  - misjudged the extent to which the PostgreSQL API shaped the coding style
    - e.g. error handling
  - should have spent more time learning C++ before diving in

Rational Commerce

So, When're You Gonna be Done?

- The Ingres bits are relatively easy
  - a lot of the work is in trying to discern the logic concealed in the libpqxx calls
  - converting to embedded SQL is a breeze
    - mostly
- Getting a build process has been a bear
  - basically has to be reverse-engineered
    - by-guess-and-by-golly
- I've been 90% done for 90% of the time I've been working on it
- "Soon"

Rational Commerce

Questions?

Rational Commerce