A PostgreSQL Based Billing System for a Telco

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Platform

- 4 Xeon x7650 (8 core) processors
  - with hyperthreading OS see 64 processors
- 128Gb RAM
- PostgreSQL 9.0
  - recently upgaded from 8.3
- Hot standby handles stats and long running reports
- Main Enlighten database is 88Gb
- Long Distance call database is 47Gb
- load average usually around 7

2 Application

- OpenACS / AOLServer
- 14 Web servers load balanced
- no state or session affinity
- connection pools

  - max lifetime is 2 hours

- also has scheduling server

3 Billing System Concepts

- Development
  - physical location, a collection of properties
    - subdivision
    - apartment complex
    - condo building
    - trailer park
- Billing Cycle
  - when a development's billing period starts
- Bill Run
  - instance of a bill cycle for a development for a specific month
- Invoice
  - bill for a single subscriber / provider combination in a bill run

4 Bill Run Life Cycle

- Data Quality
  - must pass data quality checks before can enter draft stage
- Draft state
  - where rating is performed
- Pro Forma state
  - rating is frozen and reviewed
  - if re-rating is required,return to Draft
- Final State
  - point of no return
  - items are marked as invoiced
  - accounting system is updated
  - printable invoice generation is triggered
  - data is archived

5 Rating Queue

- items placed on queue by users
- items placed on queue by scheduler
  - nightly, all bill runs in Draft state
- queue processor run by scheduler

6 Rating

- done by calling a database function
- wipe out previous invoice data for bill run
- fetch subscribed services and prices from catalog
- create line items for subscribed services
- fetch one-off charges (e.g. PPV)
- create line items for one-off charges
- rate long distance calls
- rate third party items
- rate taxes and surcharges
- fetch balances, payments, adjustments
- calculate invoice totals

7 Long Distance Calls

- LD call data is large and lives in another database
- so rating engine for LD lives there
- requires a small amount of data from services database
- 17 tables are replicated to LD database
  - about 1.2 Gb
  - uses londiste
- rating done via dblink() call

8 Taxes

- tax data obtained from commercial vendor
- fixed length fields and highly denormalized
- preprocessed into CSVs and loaded into db tables, usually monthly
- catalog items are marked with tax categories
- stored procedure rates line items according to algorithms specified by vendor
- processing is quite complex
  - some items need to be aggregated, others not
  - different tiers of taxes
  - taxes on taxes
- third party vendor's tax tables are not complete
- Surcharges
- tax rating has been a major performance bottleneck
  - now create a cache of tax rates per bill run

9 Archiving

- Final action taken on any bill run
- data spooled as CSV files
- collected and loaded into secure database

10 Printing Preparation

- Actual printing is done by third party print processor
- Some invoices are not printed
- Data is spooled as XML, one file per invoice
  - xml constructed using Postgres XML primitives
  - no hand crafted XML tags

11 Invoice XML Generation example

```
create or replace function cb_ob_invoice_xml_vod_details
       (invoice_number int)
       returns xml
       language sql as
$$
    select xmlagg (
            xmlelement(name "DETAIL", NULL,
               xmlconcat(
                  xmlelement(name "Date_Time", NULL, date_time),
                  xmlelement(name "Charge_Type", NULL, charge_type),
                  xmlelement(name "Title", NULL, title),
                  xmlelement(name "Amount", NULL, amount),
                  xmlelement(name "Tax", NULL, other_tax),
                  xmlelement(name "Total", NULL, total)
               )
            )
        )
    from cb_ob_bill_extract_vod_usage($1)
$$;
```

12 Print Processing

- spool processed nightly
- generation in parallel on separate 8 processor (virtual) server
- Apache fop
- hand crafted stylesheet
- currently adding major appearance enhancements, and different styles per provider
- elapsed time for generation is slightly over 0.5s per invoice
- when generated, zipped and shipped to print processor
- also loaded in special purpose database

13 Relation to Accounting System

- Enlighten does not keep track of payments, balances, etc
- These live in a SQLServer database of great obscurity
  - table names like "rm00103"
  - communicate using PL/PerlU + DBD::Sybase/FreeTDS
- Enlighten fetches this data from SQLServer daily
  - required for rating
- daily push of newly final bill runs to SQLServer:
  - push new customers
  - push new line items
  - create mirror document for invoice in SQLServer
  - push each line item in each invoice

14 Performance

- currently generate one invoice at a time
- can only process one bill run at a time
- steps are timed to identify bottlenecks
- lots of room for performance gains
- first goal: run rating in parallel

15 Rating Performance Statistics

Data from 2011-09-12

```
cap=$ select count(*) as invoices,
             avg(rate_time)
      from (select invoice_pk,
                   max(end_time) - min(start_time) as rate_time
            from public.cb_ob_rating_timings
            group by invoice_pk) q;
 invoices |       avg
----------+-----------------
   189033 | 00:00:00.350561
```

16 Scalability

- parallel queue should process all we need any time soon
- longer term:
  - processing items in bulk within a bill run
  - shard database and rate across multiple machines

17 Development

- originally developed mainly by 4 people
- 4 months from initial design to first invoice
- has been relatively bug free
- being on 9.0 makes thing easier than 8.3
  - rewriting queries using Common Table Expressions
  - auto-explain with query text

18 Conclusion

- project has been an unqualified success
- PostgreSQL handles the application very well
- major factor in success: process data in the database
- If we did it again I'd probably do most of it the same way

19 The End