AQMS 2.0: Oracle to PostgreSQL Migration and Beyond

September 27, 2012 Version 0.3

Paul Friberg, ISTI

Version Changes:

Version	Comment	Authors/date
0.1	Document started	Paul Friberg 2012-09-21
0.2	Mods from Ellen Yu's comments	Paul Friberg 2012-09-26
0.3	AQMS 2.0 write up after Ellen's ideas	Paul Friberg 2012-09-27
0.4	In progress, Stephane issues synonyms, dynamic SQL, BLOB, Doug N mods and comments	Doug N., Stephane, Paul

Introduction

The current AQMS system as a whole is now 14+ years old and could use some re-examination from a design sense. That is, I try and look at as if we were starting today, how would we have implemented AQMS? This paper started out as a “Can we and more importantly how can we remove Oracle” thought experiment. In the end, I have come up with a radical proposal to modify AQMS and all of its components to use PostgreSQL and also to simplify configuration and management of the system. This new preliminary design, of which migration to PostgreSQL is a key part, should be considered AQMS 2.0 since the first version rolled out to the RSN’s and volcano observatories is tentatively being called AQMS 1.0. Lets think bigger than just a database migration. Despite this, the database migration can happen in a phased approach.

As a part of this migration plan, work already performed at UC Berkeley by Pete Lombard to move AQMS real-time programs to Linux is also important, since the new system should be able to run on Solaris (which seems to be waning) and Linux. Note that when Sun Microsystems was purchased by Oracle, they also purchased both Java and Solaris. The writing is on the wall that Solaris’s life-time may be shortened by Oracle since they seem to be focusing their energy behind the Linux operating system.

In this paper I begin by detailing the current use of Oracle by the Regional Seismic Networks (RSN) in the AQMS earthquake monitoring software and how a progression to PostgreSQL is possible. PostgreSQL is a cost free (LGPL), fully open source database with many enterprise level capabilities (unlike other open databases available). In this paper I detail specifically what features of Oracle are used in AQMS, why they are used, and what features are available in PostgreSQL 9.2 (current release as of this writing) and those features that are not. For those that are not, I offer a suggestion on how a change in AQMS design would obviate the need for these missing features.

Why was Oracle chosen for AQMS?

The reason Oracle was used by AQMS in the first place is because of its' enterprise replication capabilities; mainly its advanced replication feature. Replication, in a database sense, has many definitions, but in the context of AQMS it is asynchronous multi-master replication that is the critical feature. This means that any master databases are duplicates of one another but queries to each are not guaranteed to produce exactly the same results at exactly the same time; replication of one database to another happens “as it can” in time. The replication features Oracle provided were not available in any open source solutions at the time AQMS was written circa 1999-2000.

Why migrate to PostgreSQL?

In short, PostgreSQL is the most advanced database engine available that is completely open source and cost free. The move to PostgreSQL will eliminate the annual support licensing that Universities and the USGS have to pay to keep Oracle supported. In addition, by going with an open source database AQMS will allow itself to be used by the international community, where buying an Oracle license is a bigger ordeal than you can even imagine. AQMS can only benefit by having more users working out the kinks (as has proven true from the roll-out to the RSN's) as well as making new contributions to the system.

In addition to being free, PostgreSQL is widely used and actively being developed by over 200 programmers. A new release is provided every year with improvements/upgrade paths, and this shows that it is being moved forward. There is great documentation in PostgreSQL, commercial support is also available, and there is also a huge user community. If NASA and the FAA and the National Weather Service are putting all their mission critical applications in PostgreSQL, then I think it can handle Earthquakes.

Current database configurations at RSNs

RSN's use Oracle in 2 configurations, two real-time databases replicating to a single post-processing database or to two cross-replicating post-processing databases. In each configuration, snapshot replication is used to create the real-time database and replicate it to the post-processing master. In snapshot replication, the real-time database maintains a copy of the data generated on the real-time host (from Earthworm) and on a periodic basis replicates its contents in a one way direction to the post-processing host's database. The idea here is that each real-time system is self-contained and can run without being affected by users taxing the post-processing database (either analysts or external users).

Some RSNs have multiple post-processing databases, and use multi-master replication between these databases to ensure that data entered into one database is replicated to the other database(s). UCB uses 3-way multi-master replication to synchronize 3 master databases. One of the databases, which serves requests to the public, is replicated to, but has replication from it suspended, which makes it a “read-only” database for AQMS data.

In addition to replication, AQMS uses stored procedures for maintaining business logic in the database. The idea is to concentrate the decision making logic as functions in the database programming languages (PL/SQL or Java) rather than scattered in many different programs.

Oracle features used by AQMS

In this section we detail those features of Oracle used by AQMS in more detail.

Multi-Master Asynchronous Replication

Multi-master replication refers to having multiple databases that are kept identically up to date. An insert or delete or update on one database eventually (hence asynchronous) propagates to all masters linked in the replication scheme. This means that a query to one database may not return the same result as on the other if sent simultaneously to each. AQMS operational procedures are to use only one of the databases at a time most post-processing functions so that a query to the database after an update has been applied will return the new information applied in the update.

Snapshot Database Creation

The real-time databases in AQMS are not completely built from scratch, but are snapshots of the master database. The idea is to allow select data in specific tables to flow from the master to the snapshot databases manually. This is only used for configuration push of station and channel information which is stored in the master database. In all other senses, the real-time database as it now exists is the same schema as the archival post-processing database. Note that the parametric tables in the real-time database are replicated one-way to the master archdb. Since the real-time databases are subset snapshots of the master database, they do not need to have the entire contents of any of the tables in the master database. In practice, this means that a real-time database may start with empty earthquake parameter tables, and the contents of the tables may be purged at any time when the system is not the authoritative real-time system.

Java Stored Procedures and Packages

While most database engines support Stored Procedures, none offer Java in the engine like Oracle has. Many (most?) of the AQMS stored procedures are written in Java. In addition to providing stored procedures, Oracle has a nice concept of packages, or groups of stored procedures that are identified by a given namespace. For example, the truetime package provides time conversion routines in SQL commands to simplify the confusing leap-second configuration and translation to normal epoch times.

Pro-C - pre-compiled SQL in C language

Pro-C is a way to pre-compile SQL into programs developed in the C programming language. Many would see this not as a feature, but as a lock-in to Oracle use; furthermore, few developers use pre-compiled SQL anymore. Most programmers now use JDBC for Java development with databases or ODBC for C development, and other language specific database agnostic libraries (e.g. OTL for C++, SQLAlchemy for python, DB/DBI for Perl etc). These example *DBC libraries provide an abstraction layer that insulates the programmer from Database brand specific issues. That said, two programs in heavy use in AQMS were coded using ProC: *dbselect* and *stp_server.* These two programs would require rewriting to use ODBC or the Postgres equivalent tool ECPG, which supposedly can compile ProC embedded SQL statements.

Additional Pro-C applications are also used heavily by UCB for continuous waveform archiving.

BLOB

BLOB & CLOB data types are not supported natively in PostgreSQL. However, it offers the type BYTEA, which provides similar API's. Large binary objects are currently used in AQMS for the moment tensor images as well as part of the gazetteer information.

Synonym

Synonyms are not currently supported in PostgreSQL. A synonym provides an alternative name for a table, view, sequence, procedure, stored function, package, materialized view, Java class schema object, user-defined object type, or another synonym.
Synonyms are used extensively at the NCEDC to allow transparent sharing of objects between different users. It also makes applications more portable by not having to specify the schema owner in the source codes.

Dynamic SQL

Dynamic SQL is not currently supported by PostgreSQL. It is used at the NCEDC in different applications (dbselect for example) and languages (Pro*C, SQLJ, …).

Partitioning

Oracle has the ability to partition tables and indices across multiple disks to provide enhanced query performance. This allows even parts of tables to be split across multiple disks.

Oracle Sequences

Sequences are named database entities that are used to keep ID's unique in table rows and are very critical to operating multiple real-time and post-processing databases asynchronously. In particular, each AQMS database starts with a different set of sequence numbers and increments by N (where N is > the number of databases involved in an AQMS installation) to avoid stepping on another system that may be creating its own sequences.

PostgreSQL capabilities for AQMS

PostgreSQL is a fully open source relational database that has many advanced capabilities like Oracle and has been in development since the early 90's. The current version available at www.postgresql.org is 9.2. The features in 9.2 that can be used in AQMS are:

- Multi-master replication - available asynchronously via Bucardo.
    - NOTE: Bucardo currently supports multi-master replication only between 2 databases. UCB uses multi-master replication among 3 databases.
- Multi-master replication in a cluster seems a better option. This is provided by PostgreSQL XC - http://postgresxc.wikia.com/wiki/Postgres-XC_Wiki
- Named sequences - comes built-in with PostgreSQL
- Built-in Procedural language capability (but not Java) - C/Tcl/Perl/Python or native PL/pgSQL as "Functions".
    - PL/psSQL is a different procedural languange than Oracle's PL/SQL.
    - PostgreSQL functions can return only a single value - they cannot return multiple objects that can be returned by Oracle stored procedures.
- Add-on Procedural Programming in Java - http://pgfoundry.org/projects/pljava/
- ora2pg utility - a Perl program that converts oracle databases and many elements and data to a Postgres database instance!
- ECPG - embedded SQL C compiler for Postgres which supposedly handles ProC syntax
- High Availability (HA) capabilities
- Table Partitioning - like Oracle, it too has partitioning capabilities for the CISN users who take advantage of this feature.
    - Table partitioning is handled differently in PostgreSQL than in Oracle.
- OTL used by AQMS RealTime C++ programs is compatible with PostgreSQL via ODBC otl.sourceforge.net

The logical question people will ask is why not use MySQL and our simple answer is that Oracle owns some part of MySQL and hence is subject to some control by this commercial entity (which may see it as a threat to its core database business). Furthermore, MySQL is not as fully developed as PostgreSQL since it started the game of chasing Oracle later in its development. In particular, the stored procedure/function capability is weaker in MySQL as is the multi-master replication.

Proposed AQMS 2.0

Overview

Since we are considering database migration, we can also re-work some of the complexity that is AQMS 1.0 and make things simpler and more cohesive. We should look at this as an opportunity to improve what we have since we have a 10+ year old design that has been in operation by many groups now. One design that could significantly simplify AQMS configuration and management is by eliminating the real-time databases as they currently exist entirely and building many post-processing cross-replicating databases (archdb) in a High Availability PostgreSQL cluster using Bucardo or PgSQL-XC (see figure below). This design eliminates the need for configuration pushes to real-time systems, maintaining real-time system databases, and confusing alarming configurations (one alarming configuration and set of tables used by multiple programs differently on real-time and post-processing). It simplifies the management of the database greatly and load is spread across multiple copies of the master. Real-time hosts would then connect with a dedicated post-processing host which would then cross replicate to all other masters.

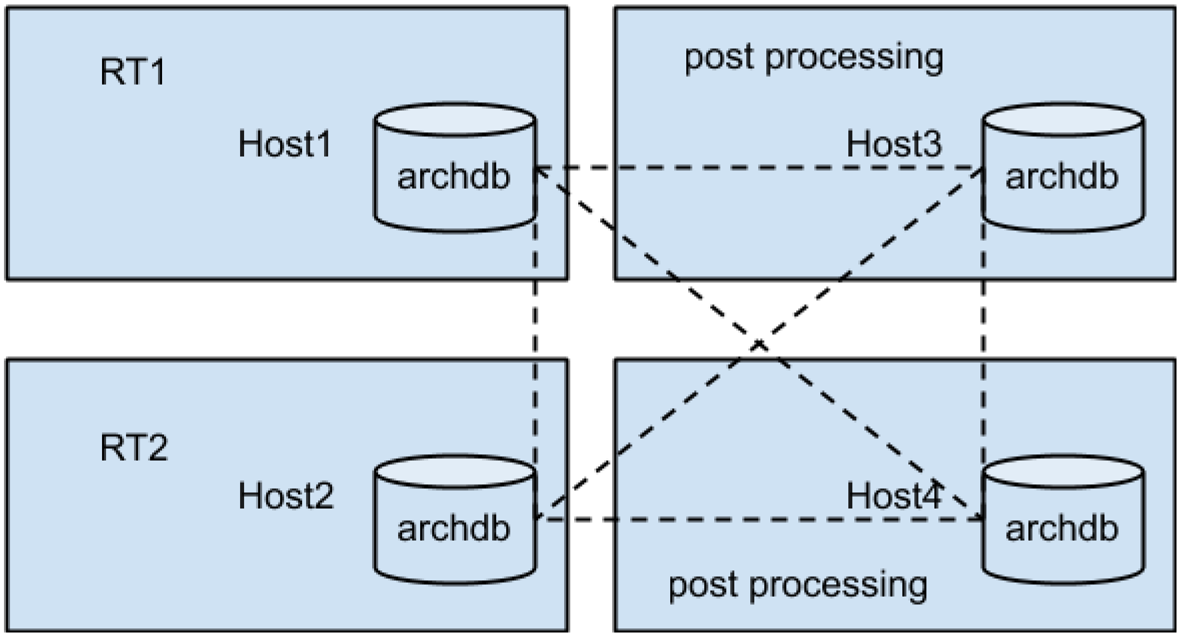

Figure 1 showing the same cross replicated archdb database across all hosts.

Handling configuration push

To get over the issue of loss of “configuration push”, which isolates incorrect station/channel changes from happening in a production system, we would still need a way of having specific network configurations (sets of stations and channels to process). We need network configurations that are live or “in production” and new ones for test only and definitive historical ones. The way to do this is via the views used to build application_channels (in the PP sense) and config_channels (in the RT sense). These views could be easily modified to have dates of validity (app channels tables already have this). This would also be a good time to consolidate the use of differently named channel configuration tables on both the RT and PP hosts...which is confusing and redundant.

Alarming issues

The current AQMS 1.0 alarming is very flexible, but also very complicated since there are configurations on each host in the system and cross-linking them makes things even more so. To simplify the confusing alarming chains for post-processing and real-time, we would have one chain that replicates across multiple hosts of which one would be designated primary and any other could be turned on as needed but would read from the same schema as is done now. A way to provide simple yet effective failover for alarming would be to have one alarming program set be designated primary (for handling alarm decisions, actions, and distribution) and a second set be designated backup. If the backup notices new activity (new events or other changes) that have not been acted on by the primary within some pre-defined time-delay, it picks up and tries to fulfill the role.

Another confusing part of alarming that exists currently in AQMS 1.0 is that you require a CMS signal to force a check on the database. The real-time programs write a new event or magnitude to the database and then send a CMS signal to notify alarming, but ideally the database should invoke this from the PCS system directly as is done for post-processing. The database should be the notifier of needs for alarming, not the program that wrote to the database. This simple change to the alarming processes would greatly improve the design since we wouldn’t need a new decision process for every new type of change to the database; instead we just include a new PCS trigger to the database.

Fallout from the design

There are consequences of the proposed design, and the biggest of these is that the real-time systems would no longer be stand-alone entities that could function happily if cut off from the post-processing hosts. The hard aspect to get over in this design is that real-time systems would no longer have “their own” database on-board. This caveat is trivial to surmount however. One way to overcome this lack of stand-alone-ness is to actually move the real-time processing engines onto the database host hosting one of the master databases. This has already been done with ancient Sun SPARC hardware in AQMS with Oracle by ISTI and is called “AQMS in a single host” and is in fact how Coso runs their network. Note there is a 6-second (or more in some networks) replication delay between the rtdb and the archdb there were often conflicts in data being worked on by programs and by analysts in the archdb. While the replication delay may still exist, if all is one master database, real-time transactions into the database can be made more cohesive at this stage to eliminate multiple bundles of information being updated separately across many masters.

Dealing with sequences

The question of sequences and shadow and primary hosts then arises if we have one single master database cross-replicated over many hosts. In this instance we need to assure that one real-time host doesn't step on another when inserting similar parametric information for an event. Overcoming that is easy since sequences are named and names can be any length we could come up with a naming scheme, but an easier namespace designation for SQL databases are schemas and we simply need to designate different schema's to hold sequences for different insert contexts (real-time versus post-processing for instance). To deal with primary and shadow real-time systems (which one produces authoritative solutions?), we also use the schema namespace concept to isolate the simple role table.

While this is a first pass design proposal, I am sure others can come up with even more simplified schemes to improve and make the AQMS system more reliable and simpler to operate and maintain and this migration is an opportunity to do just that.

Another AQMS 2.0 simplification, move closer to Earthworm

At the time the PostgreSQL development is occurring, we should also consider simplifying other aspects of the AQMS system that are not related to the database. In particular, the real-time AQMS seems overly complicated with the current configuration of two separate systems, AQMS programs and Earthworm (EW) programs. AQMS programs have their own logging, messaging, persistence, and configuration libraries which are all similar and close to Earthworm, but slightly different. Earthworm provides exactly the same functionality as all of the AQMS real-time libraries. If every AQMS RT program were remade into an Earthworm module, they could all be monitored and controlled under that umbrella. The idea would be to remove CMS entirely and use EW messages and file persistence instead. Programs like hyps2ps and trig2ps which are EW to AQMS connectors would be unnecessary. Programs that handle hypocenters (ec) and subnet triggers (trig2db and tc) would be made to listen to messages from EW rings and files could be persisted for critical messages (as are done now anyways for things like hypoinverse hypocenter archive files). The AQMS magnitude and shakemap modules (trimag and ampgen), could easily be made to listen for an EW message indicating that the event was written to the database instead of waiting for a CMS signal as they do now. This additional design change would significantly ease configuration and management of processes in addition to making less points of failure.

While moving all to Earthworm would be non-trivial, there are a number of significant gains we could achieve in this transformation, including but not limited to:

1. simpler configuration and management
2. more robust solution (less points of failure)
3. cross platform solution (more users and hence more feedback)
4. a single development environment

Phased development and testing

The progression to PostgreSQL should be done slowly to test the capabilities and usability of this system for AQMS. For that reason, I propose that the testing and configuration be done at one of the larger networks (e.g., Caltech). Below is a list of steps in order that should allow a phased development and testing of AQMS on PostgreSQL:

1. Modify schema and starting data set to PostgreSQL using ora2pg.
2. Test multi-master replication of new database using Bucardo with 2 masters.
3. Compile and run real-time programs to test OTL compatibility with this database.
4. Work on stored procedures written in PL/SQL starting with those used by real-time programs to convert them to PL/PGSQL functions.
5. Full real-time test of AQMS PostgreSQL system.
6. Begin working on post-processing configurations and packages.
7. Convert java post processing stored procs to java plugin pljava.
8. Test various post-processing programs (Jiggle/rcg/pws/wa/PCS).