Use of Pyrothorn SQL Testing sutie

S. Voutsinas, D. Morris

Institute for Astronomy, School of Physics and Astronomy, University of Edinburgh, Royal Observatory, Blackford Hill, EH9 3HJ, UK

Abstract

Documentation for the Pyrothorn Testing suite.

Keywords:

1. Introduction

As part of the GENIUS project we developed the Firethorn service, which enable users to run queries that combine local catalogues held at WFAU with remote catalogues served via VO-TAP services. We then built another layer ontop of the Firethorn services which was our own TAP implementation. Queries sent through either of these services were parsed using our own implementation of the CDS ADQL Parser (https://github.com/gmantele/taplib), which is also used by the GACS interface.

With our existing WFAU archives, our user interfaces directly query our SQL Server databases, with no ADQL - SQL Server translation. Additionally for all of our archives we collect user queries in databases.

In order to assure that our query engine, and mainly the ADQL Parser works as intended, we decided to build a testing suite that can run through these user queries and submit them through both our Firethorn service and directly through SQL Server and compare the results. The summary of these results allow us to measure how robust our parser and engine are, and analyse which queries fail and why. Some of the failing queries we found were caused due to the difference in ADQL and SQL Server syntax, which can be ignored, while others showed us bugs in the parser, some of which we fixed and pushed back as patches to the taplib ADQL Parser.

[1]

2. System Design

The main technologies used in this testing suite was Python, bash scripts and Docker. We run the tests on Virtual machines, where the only requirement is a Docker installation. The main test

[☆] `https://www.docker.com`

Email address: `dmr,stv@roe.ac.uk` (S. Voutsinas, D. Morris)

[1] `https://github.com/gmantele/taplib`

scripts consist of two folders, a setup and a test folder, along with the main run.sh bash script. The bash script takes a number of parameters described in the following section, and based on the parameters runs the according test, which runs docker instances of the services it requires. For example our basic Firethorn integration test will run docker containers for our Firethorn and OGSA-DAI services, a docker container for a MySQL database where the results of the test will be stored, and SQL Proxy containers that allow us to access the private network where our Databases are stored from a VM sitting on the outside. Dependencies for the pyrothorn library are declared and installed from the Dockerfiles. The tests also depend on a secret function, which uses a configuration file that the user running the test has access to where credentials and information needed by the testing suite are stored. Here's what this function looks like:

```
secrethost='user@host'
secretfile='${HOME:?}/secrets.file'

secret()
{
local key=${1:?}
ssh -o 'VisualHostKey=no' "${secrethost:?}" "sed -n 's/${key}=\\(.*\\)/\\1/p' \"${secretfil
}
```

A secrets file template is as follows:

```
firethorn.meta.data=
firethorn.meta.user=
firethorn.meta.pass=
firethorn.meta.host=


firethorn.user.data=
firethorn.user.user=
firethorn.user.pass=
firethorn.user.host=

firethorn.data.data=
firethorn.data.user=
firethorn.data.pass=
firethorn.data.host=

pyrothorn.storedqueries.data=
pyrothorn.storedqueries.host=
pyrothorn.storedqueries.user=
pyrothorn.storedqueries.pass=

ssh.tunnel.user=
ssh.tunnel.host=
ping=ping

absoluterows=100000
defaultrows=10000

clearwing_host=localhost
clearwing_port=80
clearwing_host_alias=
clearwing_tap_service=
clearwing_tap_service_title=""
default_community=
survey_database=
private_survey=
survey_database_user=
survey_database_password=
survey_database_server=

authentication_database=
authentication_table=
authentication_database_user=
authentication_database_password=
query_store_database_server=
query_store_database=
query_store_table=
firethorn_tap_base=
```

3. Pyrothorn tests

3.1. Full Catalogue test

The first type of test we built reads a list of queries from an SQL Server database, and runs them one by one through our Firethorn services and SQL Server. The results of the comparisons are stored in a MySQL database, and includes metadata on the query such as the duration for both, any syntax errors reported and number of rows returned.

Run as:

```
./run.sh  03 firethorn-branch firethorn-version
```

3.2. Integration tests

Because the first type of test loops through the full list of queries (¿3000 for a fairly new catalogue) which usually will take a few days to complete, we colllected a list of queries which we know pass the above test successfully into both a text file. We can then use this as an integration test, whenever we want to merge a branch and increase our Firethorn version. To make this faster, we exported these into a JSON file, which includes the expected number of rows when running the query through SQL Server. For this test we only send the query through Firethorn, and compare with the rows in the JSON file.

Run as:

```
./run.sh  01 firethorn-branch firethorn-version
./run.sh  05 firethorn-branch firethorn-version
```

3.3. Query Loop test

Another test (test 04) which we thought would be useful in order to test the robustness of our systems was a query loop test, which sends a query through every few seconds. This tests our queueing system and how the system and our Docker containers behave with a high load.

Run as:

```
./run.sh  04 firethorn-branch firethorn-version
```

3.4. Firethorn TAP deployment

In order to automate the deployment of our TAP services which are a layer on top of Firethorn, we added an option to run it through these scripts. This test takes a catalogue as a parameter and builds the environment required by the TAP service (Docker containers for Firethorn, OGSA-DAI, Apache, TAP_SCHEMA etc.)

Run as:

```
./run.sh 08  firethorn-branch firethorn-version ATLASDR1
```

3.5. TAP tests

There are two test option that will test a TAP service, one is option 06 which runs a list of queries through Firethorn and through the TAP service provided, comparing the rows returned for each query. Test 07 will run the above, but also run the taplint validation suite for the TAP services provided.

Run as:

```
./run.sh 06 firethorn-branch firethorn-version tap_service_url (TAP query test)
./run.sh 07 firethorn-branch firethorn-version tap_service_url (TAP queries test
and taplint)
```

3.6. Genius demonstrator deployment

Run option 09 is used to automate the deployment of our user interface (Clearwing), which is the interface used for the GENIUS demonstrator prototype. Run as:

```
./run.sh 09   ${newversion:?}  ${newversion:?}  1.2.3-genius
```

3.7. MySQL - SQL Server tests

The final test (test 11) will run a list of queries through MySQL and SQL server and compare the rows returned between the two.

```
./run.sh 09   ${newversion:?}  ${newversion:?}
```

3.8. Summary of available tests

```
 01 - Integration test, Query 1000 rows in Fir
 connection an compare results
 02 - Full ATLASDR1 query test, query 1000 row
 03 - Run same comparison test in historical q
 04 - Query loop test, Start a continuous loop
 05 - JSON Integration test, Query 1000 rows i
 06 - Tap test, Send 1000 rows through the giv
 07 - Perform 06 test, but also run a taplint
 08 - Build a TAP Service for a given catalogu
 09 - Build a Clearwing (webpy interface) cont
 10 - Create Firethorn chain
 11 - Run dual MySQL/SQLServer test. Compare q
```

[2] [3]

[2]http://www.star.bris.ac.uk/~mbt/stilts/sun256/taplint.html

[3]http://wfau.metagrid.co.uk/code/clearwing/