LSST Data Access and VO
Pathfinding through TAP, ADQL and beyond

Brian Van Klaveren
SLAC / LSST
IVOA Interop Meeting
Stellenbosch, SA – 05/12/2016
LSST VO Efforts

Current Work with VO

• TAP Service
  – To support QServ and Level 1 Database

• ADQL parser

• All Python!

Future Work

• SIA

• SCS

• Ancillary VO stuff (registries, etc...)
Qserv overview

Qserv is LSST’s in-house distributed database

• Based on MySQL(MariaDB) + custom UDFs
• Geospatial partitioning of tables
  – Tables chunks are colocated on worker nodes
    • Including “Level 3” tables aka User Catalogs for join efficiency
  – Uses Shared Scan to perform analytical queries
• ~1-2PB at DR1
• Qserv directly supports Asynchronous (aka batch) queries
  – Likely through an extension of grammar or session variable
    • CREATE QUERY AS SELECT * FROM Objects WHERE ...;
    • SET QUERY_MODE = ‘ASYNC’;
    • This could be automatically handled via TAP /async
    • sync queries that trigger shared scan will likely return an error
      • Clearly we need to think about this carefully
• Users will be able to use TAP or directly use database connection
  – Authentication via TAP handled via OpenID.
  – Authentication to Qserv handled via Kerberos or OpenID
    • We are writing OpenID PAM Module, Kerberos PAM already exists
    • Authorization details being worked out
LSST’s intended TAP Use Cases

Three core use cases of TAP:

• Client -> Server
  – Python, Java, C++, etc...
  – Astronomy community/VO tools are main users

• Server -> Server
  – Large amounts transferred (~1-5GB+ results)
  – IPAC’S SUIT servers are target users

• Browser/User Agent -> Server
  – Enable highly interactive web applications
  – Enable Level 3 experiments and users to write better web applications and data portals
    • Easily tie in to Authentication system via OpenID and delegation
TAP Implementation: dbserv

• Initial non-TAP PoC written by Jacek Becla
  – JSON response format that is easy to work with

• Moving towards TAP interfaces
  – Led by me
  – Working on ADQL parser
  – We still like JSON though

• Qserv has native support for “async” batch-style queries
  – Slight impedance mismatch with UWS-based recommendations for TAP async
  – Level 1 Database will need more traditional TAP approach through UWS
  – Multiple TAP Services? Not sure.
ADQL Parser: lacquer

- **Prototype:** [https://github.com/brianv0/lacquer](https://github.com/brianv0/lacquer)
  - Don’t mind the mess right now 😊
- lacquer is based on Facebook’s Presto parser, ported to Python
  - ANTLR4+Java to PLY+Python (and LL to LALR)
  - Only added syntax rule is for TOP
  - Another port may be performed to C++ (Flex+Bison) for Qserv native ADQL support
    - i.e. `SET sql_mode = 'ADQL';`
  - Alternatively, clients may rewrite queries instead of relying on TAP or a SQL proxy
    - `curs.execute(sql("SELECT * FROM Objects WHERE 1=CONTAINS(...)"))`
- ADQL validation and query rewriting is decoupled from parsing, multiple backends can be supported
- Our approach is to enforce the majority of ADQL rules a validation stage rather than in the grammar
- Parser is 50% complete, need to finish rewriter framework, then implement ADQL validation and Qserv rewriter, and clean it all up!
Things we think we don’t like

- Regarding VOTable responses:
  - XML is a machine and human readable document format. BINARY2 isn’t human readable, so why use XML at all?
  - No uniform JSON
  - No non-CDATA Binary Format
    - BINARY2 is simple but still requires custom serializers
  - VOTable is not a response format
    - but it appears to be used as one ---->
  - We expect results to regularly exceed 1GB

- Personal Pet Peeves:
  - TOP instead of LIMIT syntax
    - Especially if ADQL 2.1 supports OFFSET.
    - REQUEST=doQuery -> unnecessary (and not RESTful)

- Pagination/Framing of /results/result (esp. regarding async)
  - Does TAP return data or files? Is TAP a VOTable interface or a database interface?
  - Somewhat related to VOTable issues
    - And coupled to Response Format
    - VOTable isn’t ideal for streaming
  - Again, MAXREC instead of LIMIT and OFFSET
Our solutions to some issues

• For VOTable issues, use “Accept” header liberally and support our three main use cases
  – Maximum Compatibility – VOTable/XML and FITS
    • Consider HDF5 or SQLite?
  – Ease of Use – VOTable-inspired JSON (inside Response container)
    • Incremental/Stream processing in Java, C++, Python via Jackson, yajl, ijson
  – Binary conciseness and cross-language compatibility
    • Protobufs, CapnProto, Thrift, Avro, CBOR, MessagePack, etc...
    • No clear winner, depends on use case

• For ADQL issues, we will comply with the spec as much as possible, but also be more liberal in accepting widely adopted SQL-isms (like LIMIT)
  – TapRegExt for this?

• Similar approach to result pagination
  – We get a single result, but it’s very large!
  – Would like to stream it, but XML isn’t awesome for streaming
  – And we’d like to implement a DBAPI/JDBC-like client interface for TAP, which might also enable good dataframe (pandas) integration
{
    "result": {
        "table": {
            "metadata": {
                "elements": [{
                    "$type": "field",
                    "name": "word",
                    "datatype": "text"
                }, {
                    "$type": "field",
                    "name": "num",
                    "datatype": "integer"
                }, {
                    "$type": "field",
                    "name": "largenum",
                    "datatype": "long"
                }, {
                    "$type": "field",
                    "name": "time",
                    "datatype": "text",
                    "xtype": "timestamp"
                }, {
                    "$type": "field",
                    "name": "raw",
                    "datatype": "binary"
                }]
            }
        },
        "data": [
            ["hello", 1, "12345678901234", "2015-01-01T12:00Z", "cmF3IGRhdGE="],
            ["world", 1, "98765432101234", "2016-01-01T12:00Z", "bW9yZSByYXcgZGF0YQ=="
        ]
    }
}
More generally:

Client -> Server is the currently the intended use case of the DAL standards.

Should the following use cases be considered when designing DAL standards?

• Server -> Server
• Browser -> Server

Or are these use cases non-goals of the VO DAL?
Moving Forward

- We will continue to implement TAP, ADQL in the near future
  - And critically examine SIA, SCS
- Not sure if we’ve correctly identified issues or non-issues yet with TAP.
  - TAP/dbserv is very much still a prototype/pathfinder project for us
- We haven’t quite scratched the surface of SIA yet.
  - We do have an “imgserv”, which is similar in function to SIAP V1
- Hard to find all the current implementations
  - We need Python implementations right now

And two last questions:
- **What does it mean to be a VO site?**
- **What is the estimated manpower (in FTEs) to implement, from scratch, a VO site?**
Prototypes and pathfinding:

- **JSON output**
  - [https://gist.github.com/brianv0/07cf0acd83bde6f450a9](https://gist.github.com/brianv0/07cf0acd83bde6f450a9)

- **Pathfinding Binary formats: Protobuf/CapnProto/Thrift versions of VOTable**
  - [http://gist.github.com/brianv0/be283e3674755d76396e](http://gist.github.com/brianv0/be283e3674755d76396e)

- **Prototype Parser:**
  - [http://github.com/brianv0/lacquer](http://github.com/brianv0/lacquer)
  - Development likely moving to /lsst/lacquer soon

- **Lacquer output (Derived from a VizieR query)**
  - [http://gist.github.com/brianv0/c00e7e6a9ec89b28ea6aa7432a8201a7](http://gist.github.com/brianv0/c00e7e6a9ec89b28ea6aa7432a8201a7)

- **Prototype TAP: dbserv**
  - [http://github.com/lsst/dax_dbserv](http://github.com/lsst/dax_dbserv)