Qserv: parallel, distributed SQL query service for the LSST sky catalog

Daniel L. Wang

SLAC National Accelerator Laboratory, Menlo Park, CA, USA

27 January 2015

Outline

What is LSST?

LSST = Large Synoptic Survey Telescope (under construction)

LSST DM Database team

- SLAC: Jacek Becla, John Gates, Andrew Hanushevsky, Kian-Tat Lim, Fritz Mueller, Andrei Salnikov, Daniel Wang
- IPAC/Caltech: Serge Monkewitz

Other contributors

- IN2P3: Fabrice Jammes
- Texas A&M: Vaikunth Thukral

More coming soon...

Astronomy objectives

- Nature of Dark Energy
- Solar System
- Optical Transients
- Galactic structure

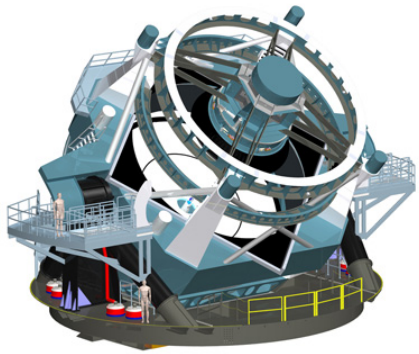


Image credit: LSST Corp / NOAO

Purpose: A baseline architecture

a catalog of celestial bodies for astronomers

- A workable solution
- Scale in CPU and data
- Affordable
- Reusable by science

No off-the-shelf solutions *priced for science* (on the horizon)

Problem: volume and complexity

Why is this hard?

LSST's final data release

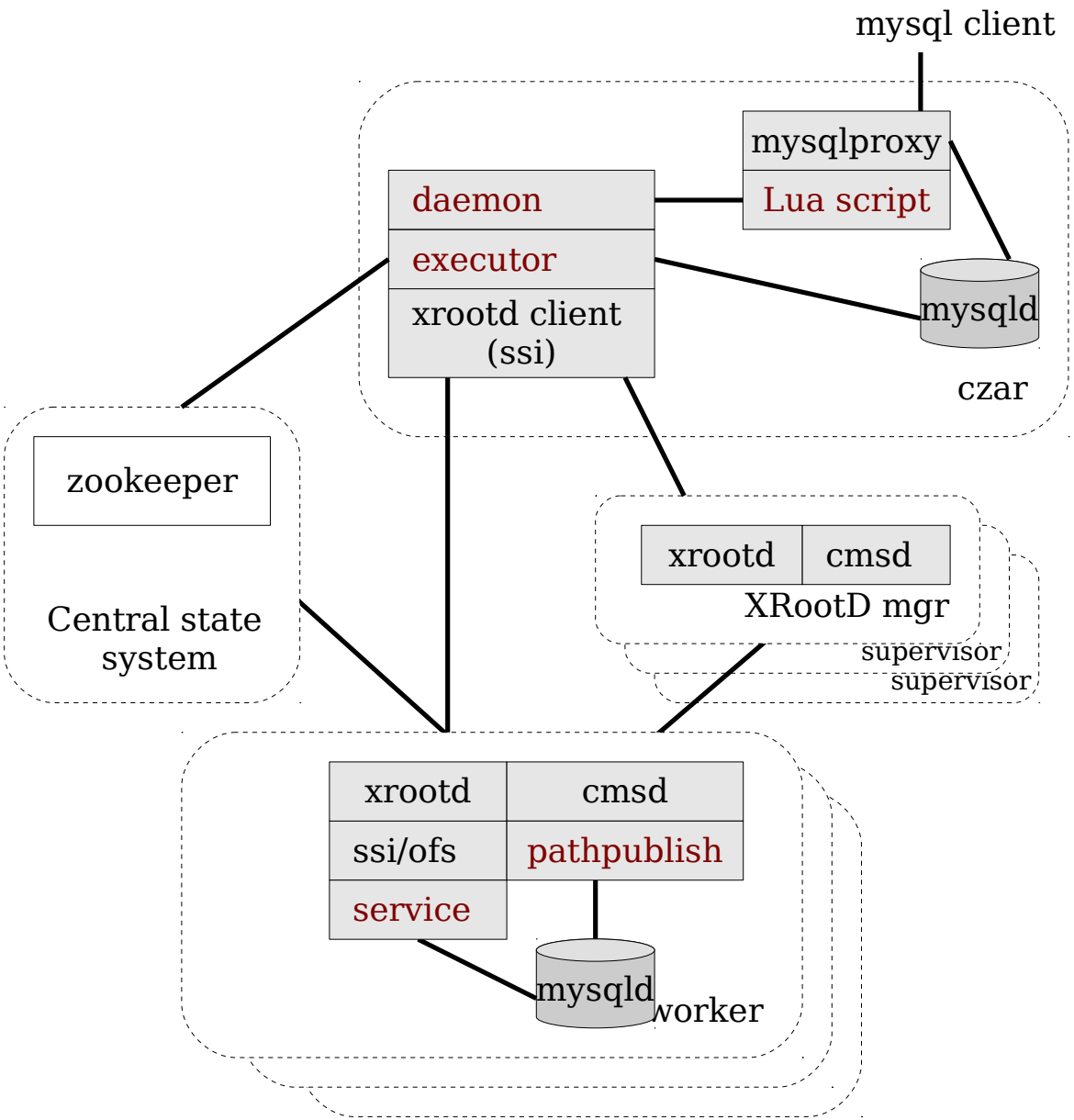
Table name	# rows	row size	footprint
Object	38×10^9	1.7kB	64TB
Object_Extra	38×10^9	26kB	1PB
Source (detections)	6.3×10^{12}	0.56kB	3.5PB
ForcedSource (expected det.)	38×10^{12}	40B	1.5PB

Requirements

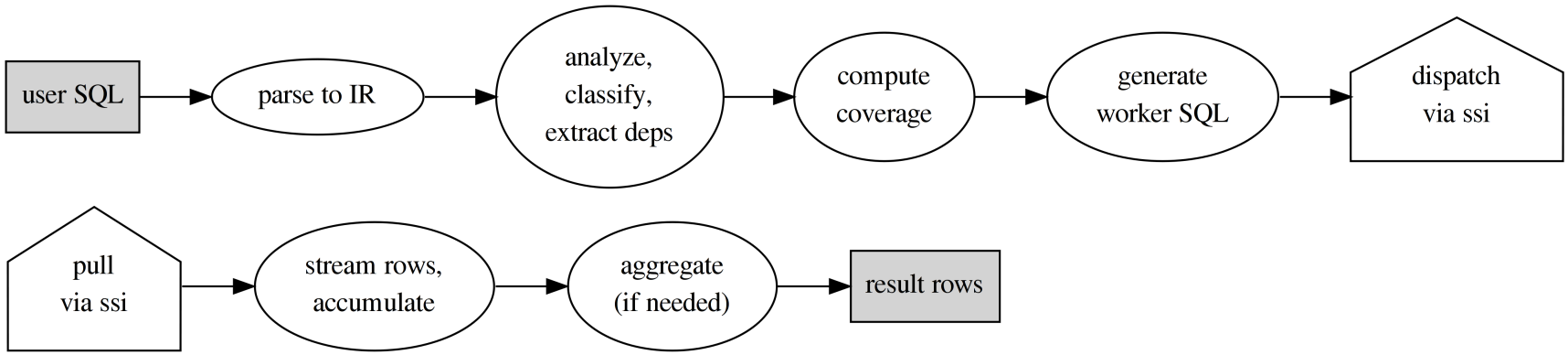
- Lowest latency ($<$ 10 seconds) for object (or small area) retrieval
- "Fast" **efficient** table scans ($<$ 1 hour)
- Efficient spatial joins* ($<$24 hours)
- High concurrency

* Brute force: Object$\bowtie$Object $\rightarrow$ O(1.4×10^{21}) [1.4 zetta!]
Object$\bowtie$ForcedSource$\rightarrow$O(1.4×10^{24}) [1.4 yotta!]

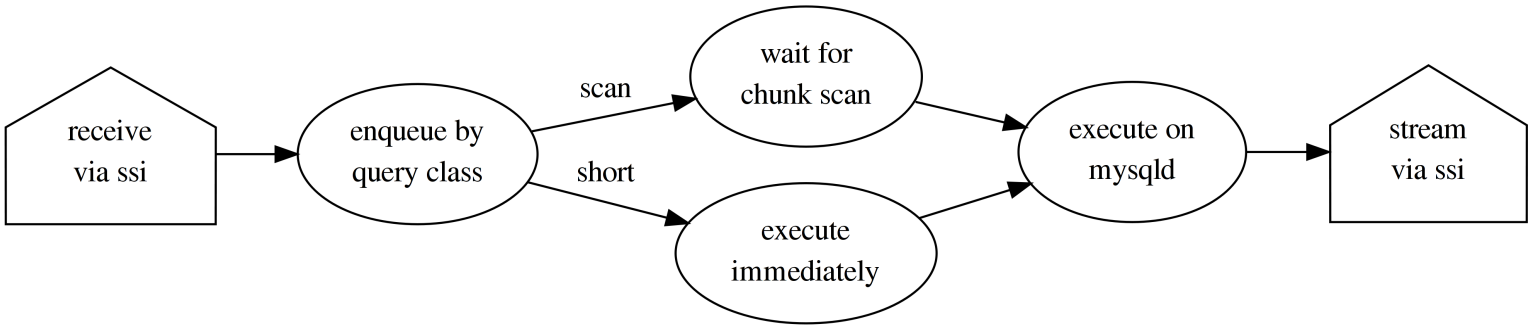
Block diagram



Czar processing flow



Worker processing flow



Partitioning

The secret sauce

- Idea: break up queries into independently-processed pieces
- But, keep small queries small: optimize small area-selections

→ spatially-contiguous tiles, "chunks" of the sky

Drivers

- Region queries
  - Small area → interactive, avoid involving many workers
  - Large area → spread load over many workers
- Near neighbor queries
  - `SELECT * FROM Object o1, Object o2 WHERE scisql_angSep(o1.ra, o1.decl, o2.ra, o2.decl) < R;`
  - Avoid all-to-all comms, quadratic scaling.

XrootD as RPC

Idea

XrootD already handles node join/leave, auto-failover among replicas.
Workers publish owned spatial regions as XrootD "files"

Implementation

- Encode data dependency as path: $/ < group > / < chunkId >$
- Workers scan their storage upon startup, publish using cmsd plugin
- Czar dispatches using experimental XrdSsi API.

What is XrdSsi?

A new, fully-asynchronous RPC-ish API to XrootD (instead of file access).

- open
- write
- read
- close
- new resource
- send request
- receive result
- release resource

Assumptions that would break a filesystem

- Open a non-existent file for writing should always fail. (invalid resource)
- Writes to the same file can happen among multiple replicas without any consistency checking. (different queries on one resource)
- It's okay if a file takes a day or week to read after opening. (slow queries)
- If I write to a file on a server, it is not okay to read the same path from another server (holding a replica). (results only on "write" server)
- Client might have 1M files open. This is totally fine and within the expected load.

XrdSsi API abstractions

- `XrdSsiService` : service instance (RPC server), provides resources
- `XrdSsiService::Resource` : path in a service (function name), receives requests
- `XrdSsiRequest` : request buffer and response stream (arguments, return value), mediate input/output transfer

Similar abstractions on both client and server:
→ Considering short-circuit XrdSsi for single-process app

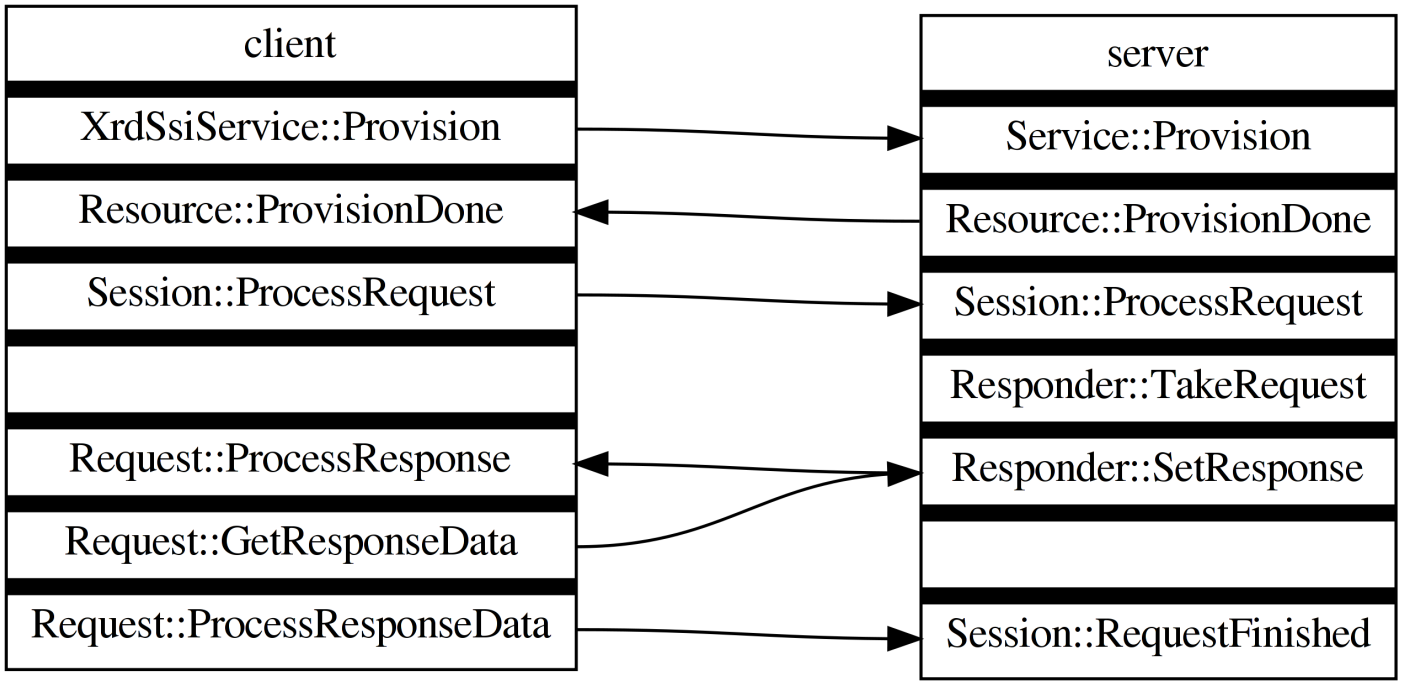
Client-side

- `XrdSsiService::Resource` : request to a service
- `XrdSsiRequest` : request payload and result callback

Server-side

- `XrdSsiService` : answer resource requests over the wire
- `XrdSsiSession` : request processor for requests, callbacks
- `XrdSsiResponder` : transport abstraction

XrdSsi Client-Server flow



Why use XrdSsi?

Compared to XrootD files

- Working around file-based assumptions was error-prone and abstractions were often leaky.
- Difficult to predict when file-based assumption would cause problems, usually trial-error with help of XrootD dev.

Compared to other RPC

- Most RPC mechanisms don't expect up to 1M function names.
- RPC interfaces buffer parameters and results.
- RPC interfaces provide type marshalling/encoding.

Status

- Overall: Working prototype ($\rightarrow$ 300 nodes)

- LSST: under construction. First light 2019
- Query execution: ok (stability, scalability, throughput, latency)
- SQL support: limited (no subqueries, no BLOB/BIT)
- Testability: Single-node installation, data loading

Lessons

- Need to parse? Use a parser
- Need to manipulate? Use an IR
- Need scalability? You need:
  - reliability/fault-recovery
  - parallelism, distribution
  - data: sequential access
  - asynchronous operations
- Working with astronomers, *understand spherical geometry*

Thank you!

Questions?

- Code: https://github.com/LSST/qserv/
- Mailing list, wiki online
- Ask me: danielw@slac.stanford.edu

Looking for new users, devs (we are hiring: `http://ls.st/dr3`).

Partitioning: Sizing considerations

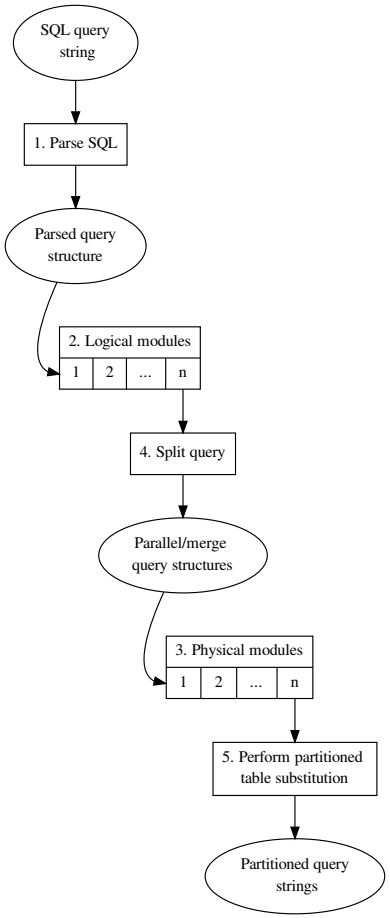
- Full-scan dispatch overhead: 10k is alright (less chunks = faster)
- Self-join: subchunk size $<$ 5k rows (more subchunks = faster)
- load balancing: "many" chunks per node (more chunks = better balance)
- subchunk size: edge length $\gg$ overlap distance (larger = bigger relative to overlap $\rightarrow$ less wasted space
- Memory footprint:
  2$\times$ [sizeof(Object_X)+sizeof(Source_X)] $<$ sizeof(DRAM per spindle)
  ..so chunk_y can be read in while chunk_x is being operated on.
  See Shared Scans

HOWTO: Execute a query

- `echo "SELECT * FROM Object WHERE areaspec_circle(3,4,0.02) AND rFlux < 0.03 LIMIT 2;" | mysql --port=4040 --protocol=TCP LSST`
- Proxy: `conn.submitQuery(queryStr)`
- Daemon: Admit query for processing, return result handle
- Executor:
  - Parse/analyze/generate chunk queries,
  - dispatch queries
  - Accumulate results in mysqld as they are ready
  - Perform aggregation (as appropriate)
  - Signal result ready
- Proxy: Fetch results from mysqld and return to user

Details: Analyze and prep

```
SELECT * FROM Object WHERE areaspec_circle(3,4,0.02) AND rFlux < 0.03 LIMIT 2;
```



- Parse and build IR.
- Infer db from context(LSST); Lookup LSST.Object
- Compute chunk coverage from circle(3,4,0.02)
- Replace restrictor call IR with IR for `scisql_ptInSphCircle(ra_PS,decl_PS,3,4,0.02)=1`
- Replace Object with LSST.Object_CC in IR
- Generate aggregation: `SELECT * FROM result_14121 LIMIT 2`
- Generate query template
- For each chunk# in coverage, substitute and dispatch

Details: Dispatch

For each chunk-query to be dispatched:

- Generate dispatch msg
- Open XrdSsi request for /q/<db>/<chunk#> (CB)
- Write dispatch msg (CB)
- Read response (CB)
- Accumulate result rows

XrdSsi request/response

- Request path from XRootD mgr
- mgr redirect to server with /q/<db>/<chunk#>
- Request path from server (redirect?)
- Channel open w/ server

Trivia: XRootD client multiplexes logical connections on single TCP connection

Details: XRootD name abstraction

- Stateless, cached $\rightarrow$ fast restarts
- Automagically form connection topo for lookups
- Enables data-addressed dispatch
- Chunk query connection monitoring

HOWTO: Table scan

```
SELECT * FROM Object WHERE rFlux < 0.03 LIMIT 2;
```

- No area restriction, Object=partitioned, mark as scan LSST.Object
- Set chunk coverage=everything
- Repeat dispatch for all chunks in db:
- chunk query: `SELECT * FROM LSST.Object_<chunk#> WHERE rFlux < 0.03 LIMIT 2;`
- Accumulate rows
- Merge query: `SELECT * FROM merge_34235 LIMIT 2`

HOWTO: Area query

```
SELECT * FROM Object WHERE
areaspec_poly(2,2,3,5,4,10,4,0,3,1);
```

- Detect polygon with vertices (2,2),(3,5),(4,10),(4,0),(3,1), Object=partitioned, if area beyond threshold, mark as scan
- Infer db=LSST, lookup partitioning for LSST.Object, lookup partitioning columns
- Compute chunk coverage using geom lib
- Repeat dispatch for covered chunks:
- chunk query: `SELECT * FROM LSST.Object_<chunk#> WHERE scisql_ptInConvexPoly(ra_PS,decl_PS,2,2,3,5,4,10,4,0,3,1)=1;`
- Accumulate rows
- Return table as-is, no merging necessary

HOWTO: point query

```
SELECT * FROM Object WHERE objectId IN (2112525,123125);
```

- Infer db=LSST, lookup partitioning for LSST.Object, lookup partitioning columns
- Check IN predicate; objectId=keyColumn for LSST.Object
- Lookup chunk# for objectId IN (2112525) (123125)
- Dispatch for two chunks of interest.
- chunk query: `SELECT * FROM LSST.Object_<chunk#> WHERE objectId IN (2112525,123125)`
- Accumulate rows
- Return table as-is, no merging necessary

HOWTO: near-neighbor

```
SELECT * FROM Object o1, Object o2 WHERE o1.objectId <>
o2.objectId AND scisql_angSep(o1.ra_PS,o1.decl_PS,
o2.ra_PS,o2.decl_PS) < 0.000001;
```

- Infer db=LSST, lookup partitioning for LSST.Object
- No spatial restriction: full-sky, mark as scan
- Dispatch for full-sky, generate subchunk lists for all chunks
- Subchunk creation(per subchunk): `CREATE TABLE Object_<chunk#>_<subchunk#> SELECT * FROM LSST.Object_<chunk#> WHERE subChunkId=<subchunk#>;`
- chunk query sequence:
  - `SELECT * FROM Object_<chunk#>_<subchunk#> o1, Object_<chunk#>_<subchunk#> o2 WHERE ...`
  - `SELECT * FROM Object_<chunk#>_<subchunk#> o1, ObjectFullOverlap_<chunk#>_<subchunk#> o2 WHERE ...`
  - ... (each subchunk)
- Accumulate rows (wait a long time for N*k join)
- Return table as-is, no merging necessary

HOWTO: reference match

```
SELECT * FROM Object o, RefMatch m, RefObject r WHERE
o.objectId = m.objectId AND m.refObjectId = r.refObjectId;
```

- Infer db=LSST, lookup partitioning for LSST.Object
- No spatial restriction: full-sky, mark as scan
- Dispatch for full-sky, generate subchunk lists for all chunks
- Create subchunk tables as in near-neighbor case.
- per-subchunk query sequence:
  - `SELECT * FROM Object_<chunk#>_<subchunk#> o, RefMatch_<chunk#>_<subchunk#> m, RefObject_<chunk#>_<subchunk#> r WHERE o.objectId = m.objectId AND m.refObjectId = r.refObjectId;`
  - `SELECT * FROM Object_<chunk#>_<subchunk#> o, RefMatch_<chunk#>_<subchunk#> m, RefObjectFullOverlap_<chunk#>_<subchunk#> r WHERE o.objectId = m.objectId AND m.refObjectId = r.refObjectId;`
- Accumulate rows
- Return table as-is, no merging necessary