(Experimental) Spatial Indexing with Cassandra

Ken W. Smith

Lasair
THE UNIVERSITY of EDINBURGH
QUEEN'S UNIVERSITY BELFAST

Overview

Quick overview of Cassandra

Quick overview of Hierarchical Triangular Mesh (HTM)

Cassandra HTM API

Cassandra Ingest tool

Cassandra Cone searching tool

Some speed tests

Some caveats

Detection Storage

Diff detections likely to grow to AT LEAST ~ 30 billion over 10 years (assuming all 10 years stored, 10M per day for at least 300 full nights per year)

Until now using traditional relational databases (e.g. MySQL) but gets clunky with billion+ numbers and difficult to replicate live. (We have 34 billion detections in our ATLAS MySQL database.)

Decided to store detections OUTSIDE the relational database.

Experimented with storage outside the relational DB on the filesystem (CephFS - single file per detection). If done this way, obvious index of storage would be by object ID. Works, but difficult to scale.

Enter Cassandra (NoSQL)

Cassandra

Replication built in from the start (factor set to 3 in our tests)

Linearly Scaleable - more nodes = more data = more query/insert capacity

Fast inserts without affecting reads (apparently writes are 10x faster than reads)

No Control Nodes - all are equal - all can accept queries and inserts

Has an SQL-like interface ("CQL" - mixed blessing - implies more SQL-like capability than actually available)

Widely adopted in industry (e.g. Facebook, Twitter, Netflix)

Other brokers (ANTARES, ALeRCE) and core LSST exploring its use.

Need to build an API so Lasair (and end users) can insert data / query it

BUT can no longer do simple database joins of objects with detections

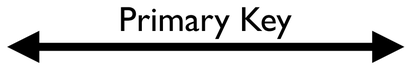
Cassandra vs Relational Database

Cassandra Data Model	Relational Data Model
Keyspace	Database
Column Family	Table
Partition Key	Primary Key
Column Name/Key	Column Name
Column Value	Column Value

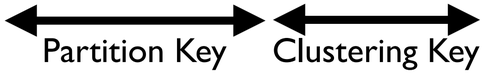
Cassandra is a Key - Value store

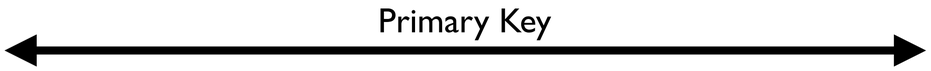
Data stored as SSTables (sorted string tables)
Flat tables - no relations - no table joins
“Tables” are groups of columns or “Column Families”

Cassandra Partition Keys (how the data is distributed)



objectId	candid	ra	dec	jd	magpsf	sigmapsf	fid
ZTF21aawxalc	1571160155615015000	121.1756238	-13.3685071	2459325.6601505	18.9818	0.16846	2



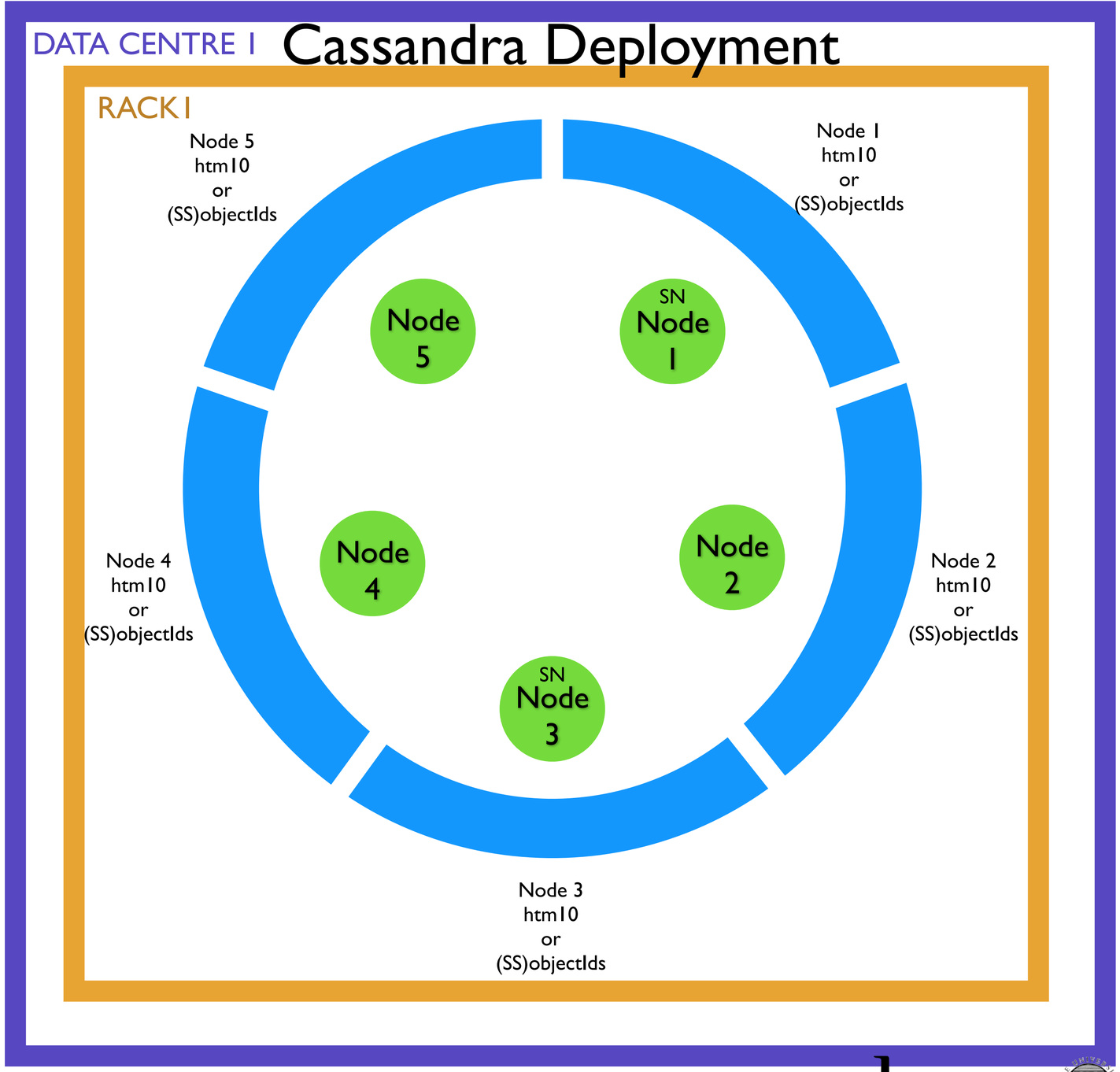


htm10	htm13	htm16	mjd	ra	dec	m	dm	filter
S30012213103	001	222	59258.66173805	273.57833	-11.52511	18.478	0.338	c

Primary Keys can be more than one column.
First column is a Partition Key (tells **which node to store the data**) -
e.g. ObjectId or SSObjectId or Spatial Index

Secondary columns can become Clustering Keys - e.g. (M)JD

Cassandra Deployment



Replication follows a ring architecture

Partition Key determines which Node to which the data is primarily copied (via Murmur3 hash*) - e.g. objectId or spatial index.

*Two alternative partitioners are available (random [MD5], and byte ordered), but their use is not recommended because of cryptographic CPU load and uneven partitioning.

Replication factor determines how many nodes to which the data is copied.

Replication goes clockwise in this architecture
Lasair and ATLAS architecture = 5 nodes.
ATLAS data also stored on SSD

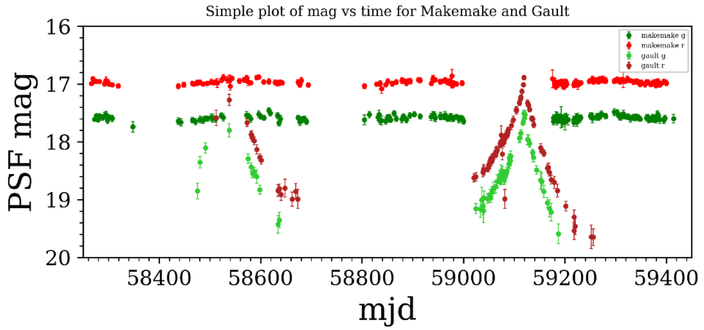
Lasair THE UNIVERSITY of EDINBURGH QUEEN'S UNIVERSITY BELFAST

What to store within Cassandra

Detections (diaSources)

objectId	candid	ra	dec	jd	magpsf	sigmapsf	fid
ZTF21aawxalc	1571160155615015000	121.1756238	-13.3685071	2459325.6601505	18.9818	0.16846	2

Solar System Objects (in ZTF this is a thin subset of Detections)



Thin HTM Spatially Indexed Detections (can use full table once object ID recovered)

(To Be Investigated) Image stamp “blobs” indexed by objectId

SQL-like queries

```
cqlsh:lasair> select objectId, candid, jd, magpsf, sigmapsf, fid from candidates where objectId IN
('ZTF19acziogq','ZTF20aaxtkhw','ZTF19aaiqpbl');

 objectid     | candid              | jd             | magpsf   | sigmapsf | fid
--------------+---------------------+----------------+----------+----------+-----
 ZTF19aaiqpbl | 1474364210615015021 |  2459228.864213 | 19.8234 |  0.15084 |   1
 ZTF19aaiqpbl | 1478330720615010062 | 2459232.8307292 |  19.032 |   0.1409 |   2
 ZTF19aaiqpbl | 1496329080615010063 | 2459250.8290856 | 19.3573 |  0.17656 |   2
 ZTF19aaiqpbl | 1498270180615010073 | 2459252.7701852 | 20.1956 |  0.21429 |   1
 ZTF19aaiqpbl | 1500274120615015006 | 2459254.7741204 |  19.976 |  0.20268 |   1
 ZTF19aaiqpbl | 1502232960615015007 |  2459256.732963 | 19.4225 |   0.1433 |   2
 ZTF19aaiqpbl | 1511231350615010028 | 2459265.7313542 | 19.3567 |  0.19184 |   2
 ZTF19aaiqpbl | 1524211820615010056 | 2459278.7118287 | 19.6038 |   0.2987 |   1
 ZTF19aaiqpbl | 1527228070615015012 | 2459281.7280787 | 19.2588 |  0.11524 |   2
 ZTF19acziogq | 1508189055215015006 | 2459262.6890509 | 19.3026 |  0.19416 |   2
 ZTF20aaxtkhw | 1517544004215010052 | 2459272.0440046 | 17.4548 |  0.15334 |   2

(11 rows)

cqlsh:lasair> select ssnamenr,jd,magpsf,fid from sscandidates where ssnamenr='136472' and jd > 2459380 and fid=1 allow filtering;

 ssnamenr | jd              | magpsf   | fid
----------+-----------------+----------+-----
   136472 | 2459381.7786343 | 17.5514 |   1
   136472 | 2459383.7370486 | 17.5187 |   1
   136472 | 2459385.7079282 | 17.5928 |   1
   136472 | 2459387.7777431 | 17.6131 |   1
   136472 | 2459395.7440278 | 17.6187 |   1
   136472 | 2459397.7681713 | 17.5907 |   1
   136472 |  2459400.765544 | 17.5517 |   1
   136472 | 2459414.7415972 | 17.5994 |   1

(8 rows)
```

Cassandra Limitations

Queries by ANY OTHER COLUMN other than primary key are difficult (but can use "ALLOW FILTERING" clause sparingly under certain circumstances).

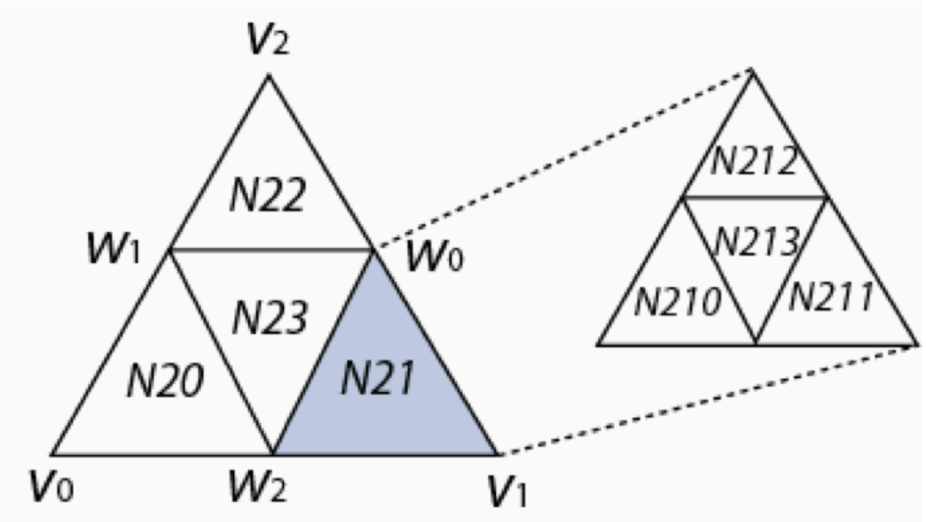
Question: How often do we search the detections table by anything other than objectId, position or date?

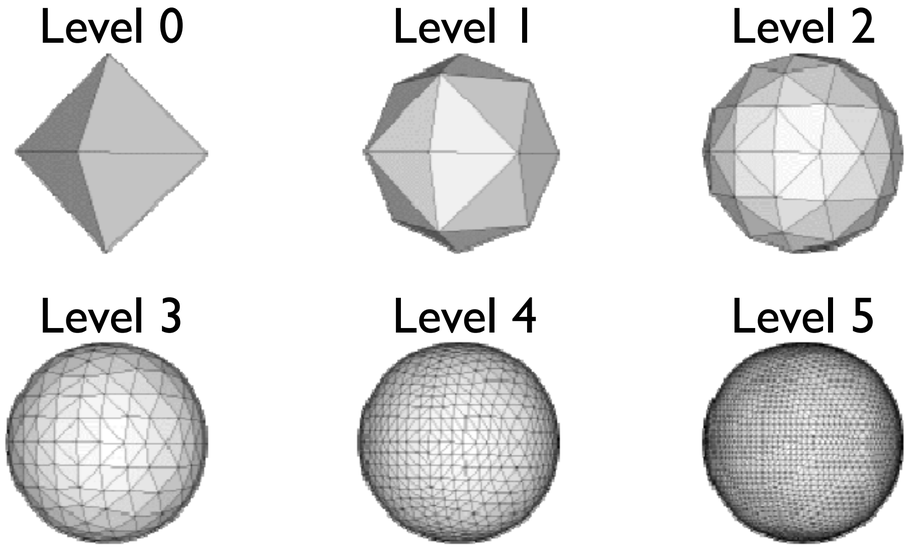
Mitigation: multiple "thin" tables (e.g. ssobjects, htmcandidates) + API

Workaround? For both Cassandra and CephFS (grab subset of objects by ID from the object summary table - then background filter the full lightcurves)

Can't do RANGE queries (BETWEEN). Mitigation: use IN (though queries can become large).

HTM quick overview





Quad tree index.

Numerically adjacent triangles are also spatially adjacent.
Hence standard HTM C++ API returns ranges

HTM level 10 area ~ 18 arcmin^2
HTM level 13 area ~ 0.3 arcmin^2
HTM level 16 area ~ 0.004 arcmin^2 (~ 15 arcsec^2)

Relational Databases & HTM

The HTM API returns RANGES of triangles within specified radius (e.g. 5 arcsec)
Here's a small C++ program to return sql and triangle ranges given an RA, Dec and a radius in arcsec. Results are returned in decimal ranges. This works well for relational databases (and speeds up disk access) but does NOT work well for Cassandra.

(In this case, the values are 85.13154994520691, 37.936764657367384 and 5 arcsec)

```
./HTMCircle 16 85.13154994520691 37.936764657367384 5 tcs_cat_gaia_dr2

select * from tcs_cat_gaia_dr2 where
   htm16ID between 64759070916 and 64759070917
or htm16ID between 64759070919 and 64759070919
or htm16ID between 64759070936 and 64759070936
or htm16ID between 64759070938 and 64759070939
or htm16ID between 64759070960 and 64759070975
;
```

Relational Databases & HTM

Increase the radius to 50 arcsec:

```
./HTMCircle 16 85.13154994520691 37.936764657367384 50 tcs_cat_gaia_dr2

select * from tcs_cat_gaia_dr2 where
   htm16ID between 64758611968 and 64758612223
or htm16ID between 64758612352 and 64758612352
or htm16ID between 64758612864 and 64758612879
or htm16ID between 64758612896 and 64758612911
or htm16ID between 64758612916 and 64758612919
or htm16ID between 64759070720 and 64759071743
or htm16ID between 64759922688 and 64759922943
or htm16ID between 64759923264 and 64759923264
or htm16ID between 64759923520 and 64759923523
or htm16ID between 64759923526 and 64759923526
or htm16ID between 64759923534 and 64759923534
;
```

Cassandra & HTM

Cassandra has a query language called CQL, which is SQL like, with the following exceptions

OR statements are not allowed - but IN statements ARE allowed. Ranges are NOT allowed (kind of).

Let's do the 5 arcsec query again, but this time we expand out all the triangles.

```
./HTMCircleAllIDs 16 85.13154994520691 37.936764657367384 5 tcs_cat_gaia_dr2

select * from tcs_cat_gaia_dr2 where htm16ID IN (
64759070916,64759070917,64759070919,64759070936,64759070938,
64759070939,64759070960,64759070961,64759070962,64759070963,
64759070964,64759070965,64759070966,64759070967,64759070968,
64759070969,64759070970,64759070971,64759070972,64759070973,
64759070974,64759070975);
```

Cassandra & HTM

Same query again, but increase the radius to 50 arcsec:

```
./HTMCircleAllIDs 16 85.13154994520691 37.936764657367384 50 tcs_cat_gaia_dr2

select * from tcs_cat_gaia_dr2 where htm16ID IN (
64758611968,64758611969,64758611970,64758611971,64758611972,
.
.
<1570 more triangles!!>
.
.
64759923521,64759923522,64759923523,64759923526,64759923534);
```

The query works, but it's getting very verbose, and doubling the radius again quadruples the triangle count.

Cassandra and Spatial Indexing

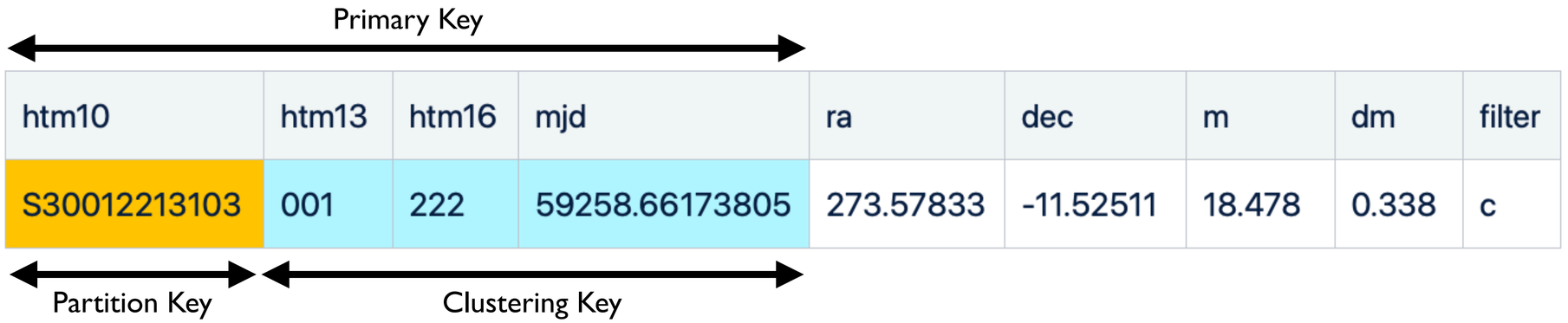
HTM is Hierarchical
Level 16 is a superset of Level 13
Level 13 is a superset of Level 10

	Decimal	Binary	Base4
HTM16	54680902005	**11001011101100111100010110010110101**	**N02323033011211311**
HTM13	854389093	**11001011101100111100010100101**	**N02323033011211**
HTM10	13349829	**1100101110110011111000101**	**N02323033011**

Use the string representation (base 4) representation (could just as easily use binary)

Split the deeper HTM levels into suffixes

Cassandra and Spatial Indexing



Query using the HTM10 field on its own (the partition key),
the HTM10 and HTM13 fields (= HTM level 13),
the HTM10 and HTM13 and HTM16 fields (= HTM level 16).
Can't (normally) query via a clustering key out of order*

* This behaviour can be overridden by the (not normally recommended): `ALLOW FILTERING` clause in the CQL query. (I think this CAN be safely used if the partition key is used in the query.)

Cassandra Spatial index python API (query by position & radius)

```
pip install gkhtm
```

```
from gkhtm._gkhtm import htmCircleRegionCassandra

whereClause = htmCircleRegionCassandra(272.40279,-9.97105,2.0)
for row in whereClause:
    print(row)

 where htm10 IN ('S30032030123') AND (htm13,htm16) IN (('120','201'),('120','202'),('120','203'),('120','231'),('120','232'),('120','233'))

whereClause = htmCircleRegionCassandra(272.40279,-9.97105,50.0)
for row in whereClause:
    print(row)

 where htm10 IN ('S30032030122') AND (htm13) IN (('110'),('112'),('113'),('220'),('300'))
 where htm10 IN ('S30032030123') AND (htm13) IN (('010'),('011'),('013'),('100'),('101'),('102'),('103'),('110'),('111'),('112'),('113'),
('120'),('121'),('122'),('123'),('130'),('131'),('132'),('133'),('320'),('321'),('322'),('323'))

whereClause = htmCircleRegionCassandra(272.40279,-9.97105,250.0)
for row in whereClause:
    print(row)

 where htm10 IN
('S30032030120','S30032030121','S30032030122','S30032030123','S30032030130','S30032030131','S30032030133','S30032030320','S30032030321','S3003
2030323')
```

Semi-arbitrary break points:

Radius <= 15 arcsec: Use HTM Level 16 (up to median 107 triangles based on 1000 random sky positions)
Radius between 15 arcsec and 200 arcsec: Use HTM Level 13 (up to median 280 triangles)
Radius between 200 arcsec and 1800 arcsec: Use HTM Level 10 (up to median 360 triangles)

Cassandra Ingest tool

`pip install gkhtm; pip install gkutils; pip install gkdbutils; pip install cassandra-driver`

```
cassandraIngest -h
Ingest Generic Database tables using multi-value insert statements and multiprocessing.

Usage:
  cassandraIngest <configFile> <inputFile>... [--fileoffiles] [--table=<table>] [--tableDelimiter=<tableDelimiter>] [--bundlesize=<bundlesize>] [--nprocesses=<nprocesses>] [--
nfileprocesses=<nfileprocesses>] [--loglocationInsert=<loglocationInsert>] [--logprefixInsert=<logprefixInsert>] [--loglocationIngest=<loglocationIngest>] [--
logprefixIngest=<logprefixIngest>] [--columns=<columns>] [--types=<types>] [--skiphtm] [--nullValue=<nullValue>] [--fktable=<fktable>] [--fktablecols=<fktablecols>] [--
fktablecoltypes=<fktablecoltypes>] [--fkfield=<fkfield>] [--fkfrominputdata=<fkfrominputdata>] [--racol=<racol>] [--deccol=<deccol>]
  cassandraIngest (-h | --help)
  cassandraIngest --version

Options:
  -h --help                               Show this screen.
  --version                               Show version.
  --fileoffiles                           Read the CONTENTS of the inputFiles to get the filenames. Allows many thousands of files to be read, avoiding command line constraints.
  --table=<table>                         Target table name.
  --tableDelimiter=<tableDelimiter>       Table delimiter (e.g. \t \s ,) where \t = tab and \s = space. Space delimited assumes one or more spaces between fields [default: \s]
  --bundlesize=<bundlesize>               Group inserts into bundles of specified size [default: 1]
  --nprocesses=<nprocesses>               Number of processes to use per ingest file. Warning: nprocesses x nfileprocesses should not exceed nCPU. [default: 1]
  --nfileprocesses=<nfileprocesses>       Number of processes over which to split the files. Warning: nprocesses x nfileprocesses should not exceed nCPU. [default: 1]
  --loglocationInsert=<loglocationInsert> Log file location [default: /tmp/]
  --logprefixInsert=<logprefixInsert>     Log prefix [default: inserter]
  --loglocationIngest=<loglocationIngest> Log file location [default: /tmp/]
  --logprefixIngest=<logprefixIngest>     Log prefix [default: ingester]
  --columns=<columns>                     List of columns, comma separated, no spaces. If blank, assumes all columns of the input data.
  --types=<types>                         PYTHON column types in the same order as the column headers.
  --skiphtm                               Don't bother calculating HTMs. They're either already done or we don't need them. (I.e. not spatially indexed data.)
  --nullValue=<nullValue>                 Value of NULL definition (e.g. NaN, NULL, \N, None) [default: \N]
  --fktable=<fktable>                     Cassandra has a flat schema, so join to another table file via a foreign key (e.g. exposures).
  --fktablecols=<fktablecols>             The valid columns in the foreign key table we want to use - comma separated, no spaces (e.g.
expname,object,mjd,filter,mag5sig,zp_mag,fwhm_px,exptime,detem).
  --fktablecoltypes=<fktablecoltypes>     The valid (python) column types in the foreign key table we want to use - comma separated, no spaces (e.g.
str,str,float,str,float,float,float,float,float).
  --fkfield=<fkfield>                     Foreign key field [default: expname]
  --fkfrominputdata=<fkfrominputdata>     Foreign key from input data. If set to filename it will use the datafile filename as the key [default: filename]
  --racol=<racol>                         Column that represents the RA [default: ra]
  --deccol=<deccol>                       Column that represents the Declination [default: dec]
```

Cassandra cone search tool

`pip install gkhtm; pip install gkutils`

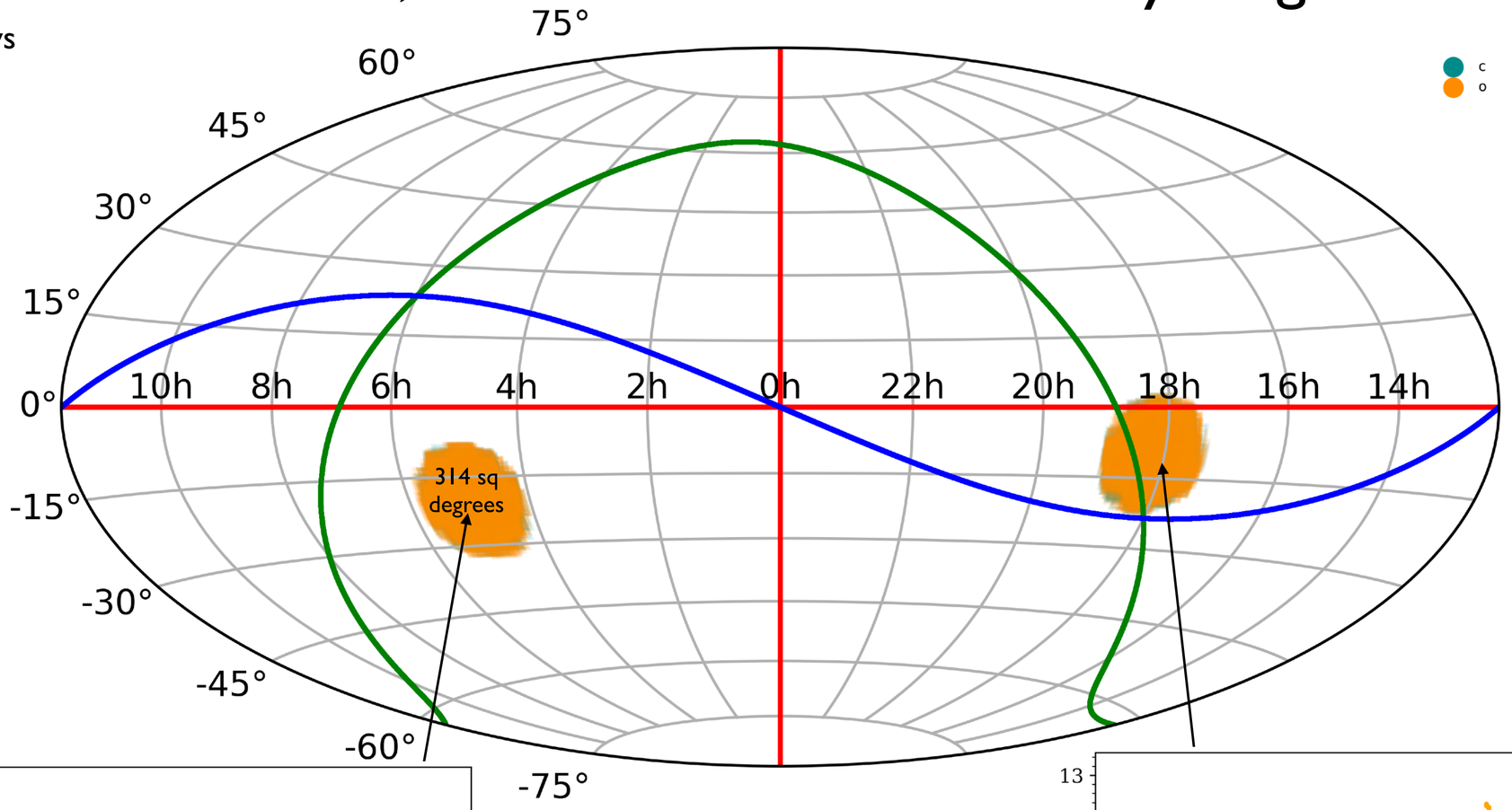
```
coneSearchCassandra -h

Query cassandra by RA and dec. The coords variable should be RA and dec, comma separated with NO SPACE. (To facilitate negative declinations.)

Usage:
  coneSearchCassandra <configFile> <coords> [--radius=<radius>] [--coordsfromfile] [--saveresults] [--resultslocation=<resultslocation>] [--number=<number>]
[--table=<table>] [--namecolumn=<namecolumn>] [--querycolumns=<querycolumns>] [--nprocesses=<nprocesses>] [--loglocation=<loglocation>] [--
logprefix=<logprefix>] [--coldate=<coldate>] [--colmag=<colmag>] [--colmagerr=<colmagerr>] [--colfilter=<colfilter>] [--colra=<colra>] [--coldec=<coldec>] [--
colexpname=<colexpname>]

  coneSearchCassandra (-h | --help)
  coneSearchCassandra --version

Options:
  -h --help                             Show this screen.
  --version                             Show version.
  --coordsfromfile                      Treat the coordinates parameter as a file of coordinates.
  --radius=<radius>                     Cone search radius in arcsec [default: 2].
  --saveresults                         If set, store the results in a file whose prefix starts with this.
  --resultslocation=<resultslocation>   If saveresults is set, store the results in this directory [default: /tmp].
  --number=<number>                     If set and is smaller than the total list, choose a random subset.
  --table=<table>                       Table to search [default: atlas_detections].
  --namecolumn=<namecolumn>             If set, choose this as the name of the result file. Otherwise lc_pid_0000001.csv, etc [default: source_id].
  --querycolumns=<querycolumns>         Grab the selected colums, comma separated, no spaces - instead of everything. Could speed up queries.
  --nprocesses=<nprocesses>             Number of processes to use by default to get/write the results [default: 1]
  --loglocation=<loglocation>           Log file location [default: /tmp/].
  --logprefix=<logprefix>               Log prefix [default: coneSearch].
  --coldate=<coldate>                   The column that represents date [default: mjd].
  --colmag=<colmag>                     The column that represents mag [default: m].
  --colmagerr=<colmagerr>               The column that represents mag error [default: dminst].
  --colfilter=<colfilter>               The column that represents the filter [default: filter].
  --colra=<colra>                       The column that represents the RA [default: ra].
  --coldec=<coldec>                     The column that represents the declination [default: dec].
  --colexpname=<colexpname>             The column that represents the exposure name OR (e.g.) objectId [default: expname].
```

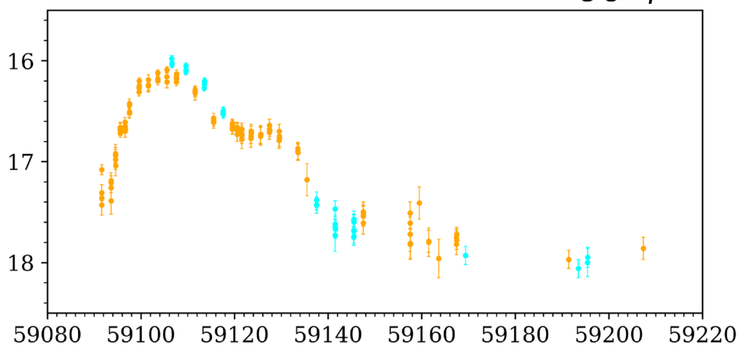
ATLAS, Cassandra and DB of Everything

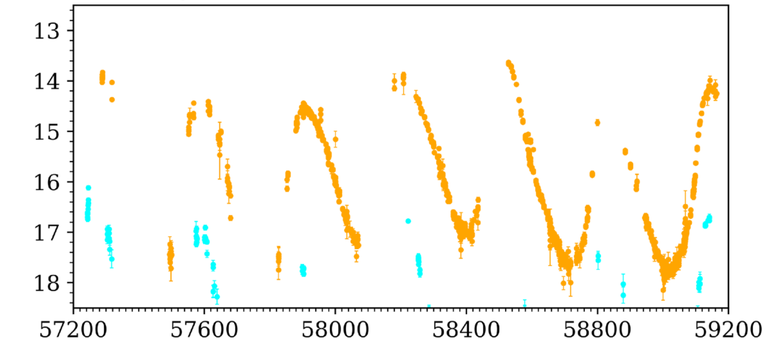
2 x 10 degree radius areas

Ingested about 28B rows
5 x 1.2 TB
Entire 4 years of data
(Test areas only so far)
Spatially indexed

Whole sky = 1.2T rows
Total size 350TB all nodes
70TB each for 5 nodes
7TB each for 50 nodes
(4yrs, 2 telescopes)







Lasair
THE UNIVERSITY of EDINBURGH
QUEEN'S UNIVERSITY BELFAST

Data Loading into Cassandra & 0.5M random cone searches (via new API)

28 billion ATLAS rows loaded at a rate of 5 to 6 billion per day by flooding all 5 nodes with data (loading capacity might be improved further by using "COPY")

Tested linear scaling - more nodes directly scales to insert capacity
(scaling may flatten off as inter-node chatter gets larger with larger clusters)

Many random cone searches

Positions selected from (e.g.) Gaia DR2 (12 < G < 19).

```
coneSearchCassandra ~/config_cassandra_atlas.yaml \
~atls/galactic_centre_all_gaia_objects_2degrees_ra_dec_mag_12_19.txt \
--coordsfromfile --table=atlas_detections --nprocesses=32 --number=100000 \
--saveresults --resultslocation=/tmp/atlas_lightcurves \
--querycolumns=mjd,m,dminst,filter,ra,dec
```

Scales linearly with number of cores (see above chatter caveat for large number of nodes)

Multiprocessed on each of 5 nodes - 32 cores & 100,000 cone searches = 8 minutes
(total of 0.5 million LC, producing 1 lightcurve/ms)

Solar System objects table generated. HTM Indexed candidates (detections) generated

Caveats & Outlook

The code is experimental!

Cassandra 3.x only. (Cassandra 4 supported in future - hopefully.)

It uses the ascii string representation of base 4. (Binary will be better for table compactness.)

HTM level choices are arbitrary (though the backend API can be made to select any group of levels)

Arbitrary max radius of 1800 arcsec (0.5 degrees). Could be safely extended to 1 or 2 degrees.

Some cone searches require multiple database hits (to get round the OR limitation).

Yet to compare SSD vs HDD performance. (All load tests here on SSD)

Code only tested with CentOS7, Ubuntu, MacOS so far. (Related to compilation of HTM C++ library)

JD/MJD additional filtering to be (load) tested - especially at lower levels where index usage is tricky

Current cone search tool is cone only, but future versions will be able to do box searches, etc.