MOPS Partitioning

LSST Database

Data Challenges and Production

Typical mops activities:

- DayMOPS queries DiaSources from a given night, generates tracklets (which are stored in *mops_Tracklet* table) and links them into tracklets (links are stored in *mops_TrackletsToDIASource* table)
- DayMOPS is taking tracklets from >3 nights and generates moving objects (stored in *MovingObject* table. Each moving object contain information about an orbit). (During this process tracks are created, and they are run through orbital determination program. Tracks are saved (*in *mops_TrackToTracklet*). [**Schema for that table is not in EA, it is only in daymops/trunk/tests/, need to fix!**]). Each moving object is also linked to the DiaSources which match its orbit (via a reference movingObjectId in DiaSource table), and to the corresponding tracklets (via *mops_MovingObjectToTracklet* table)

Separating mops tables to simplify partitioning

To provision for the fact that some DiaSources/Sources/ForcedSources/ForcedDiaSource will be linked to objects, and some to moving objects, current scheme is:

- Source, DiaSource tables: there are 2 columns in each of these tables: objectId and movingObjectId. Typically, only one of these ids would be set.
- Forced* tables are going to be handled through 4 separate tables: ForcedSourceForObject, ForcedDiaSourceForObject, ForcedSourceForMovingObject, ForcedDiaSourceForMovingObject

Disadvantage of keeping together Sources and DiaSources that belong both to objects and moving objects is that whatever partitioning scheme we chose for Sources/Objects?, the MovingObject table will be forced to use that scheme too (this is required in order to avoid cross-node communication during joins).

Proposal:

- **split Source into SourceForObject and SourceForMovingObject**
- **if uncertain where a given Source belongs, create an entry in both tables, and set a special flag indicating given source has a "sibling"**
- **same scheme for DiaSource**

This would allow us to keep the partitioning scheme for moving object related tables completely separate from partitioning we chose for Object/Source?* tables.

Partitioning mops tables

Partitioning should be driven by the analyses that are expected to be run most frequently (common queries). Common queries will definitely require correlating moving objects (selected by querying a range of orbital parameters stored in MovingObject table) with their DiaSources and/or tracklets. Many queries will involve (ra,dec,time) coordinates. In addition, QA analysts will be querying MovingObjects to see if they are similar to mops_SSM (mops_SSM contains synthetic solar system model orbits).

Derived requirements: DiaSources, mops_Tracklet, mops_SSM, and related tables (mops_TrackletsToDIASource, mops_MovingObjectToTracklet, ...) should all be partitioned the same way as MovingObject table. In practice, we expect to cluster them based on movingObjectId.

> KTL: most of the mops_* tables will be created *before* MovingObject, so it will be difficult to partition them by movingObjectId...

Main question: how to partition MovingObject table. The goal is to find partitioning scheme such that most queries that use moderate numbers of moving objects (but not all of them) can be satisfied with a single partition instead of requiring a full table scan. Moving objects don't have a fixed position (rules out typical spatial partioning).

[Need to brain-storm what the best partitioning scheme is...]

> KTL: I think the only thing that may be useful is Lynne's suggestion of semi-major axis. This will at least break down the MovingObjects into groups that astronomers tend to work with: NEOs, main belt asteroids, Trojans, KBOs, Oort TNOs, etc. (jmyers agrees).
>
> LJ : One problem with using *only* the semi-major axis ('a') is that the distribution of objects is extremely uneven with 'a'. In particular, the main belt asteroids make up almost 9 million of those MovingObjects (out of the 11M MovingObjects we have in the MOPS solar system model today). If this is ok (the other groups will likely be within one partition, but main belt asteroids will be spread over more than one partition), then is good enough for now.

Indexing

Today, MOPS takes care of most indexing by building in-memory kd-trees. In pretty much every possible place, a kd-tree is built on most frequenty queried data from a given night, and queries are done on that tree. Examples:

- there is a kd-tree used in precovery attribution - it's a tree on tracklet locations, so we can compute ephemerides for the moving objects and then do range searches on "orphan" tracklets.
- For building per-image ephemeris, we dump all objects, perform coarse ephemeris for the night, build a kd-tree on image locations, then perform per-object linear interpolation to each image time and then range search the image tree to find images which may be intersected.
- 6-D kd-trees are used for finding similar orbits and merging them.

The only place where mops is using typical database indexes are indices on MJD (and sometimes on "field")

Questions: how big should these fields/sky tiles be? Does it depend on the moving object density?

Building in-memory kd-trees is probably not scalable as the number of moving objects increases. We expect to have ~one billion moving objects in the last data release (DR11).**[need to do spreadsheet analysis!]**

jmyers: I think that this will continue to be tenable. 1 billion is not too painful a number if we have a few very high-memory machines; failing that, there are known methods for building distributed KD-Trees which support range search (?http://www.pmw.org/~gardnerj/ntropy/).

LJ : most places we will likely want to continue using these in-memory kd-trees. if there are places it would be better to swap to calling things from the database, we need to look at this in the MOPS design review. what information and numbers are necessary to determine whether it's more efficient to call from indexed db, versus build and use in-memory tree? Is this a DC3b or eventual LSST question?

Simulation

The simulation is taking known ephemerides and generates DiaSources for them. This involves matching RA/Dec/time of rough nightly ephemerides against the RA/Dec/time of the OpSim, and then generate precise ephemerides for each Opsim pointing.

In DC3b, this simulation will cover 1 year worth of data (production will cover 10 years). For *each night* the numbers are:

- ephemerides: 11 million (these are "nightly ephemerides aka at midnight")
- observation positions in OpSim: (in production) : 800-1000 (in DC3b) : because of the limited number of pointings .. maybe 100-200
- DiaSources: (in production): ~2M on OpSim 3.61 night = 100 (test case). This can vary significantly. (in DC3b) : <100K / night
- tracklets: ~10-20 times more than the number of DiaSources

Question: are we creating a new set of "nightly ephemerides" each night, or are we updating entries generated during processing previous nights? (e.g., will we end up with 4 billion or 11 million of these "nightly ephemerides"?)

jmyers: we could get by using 2-3 nights of ephemerides at any given time, but we'd prefer to hold on to nightly ephemerides for a full 1 year - that way we can test new OpSim output without re-generating these ephemerides (which is quite slow and painful).

LJ : potential terminology confusion here ... we generate 'rough nightly ephemerides' (at midnight of each night) for all the objects, and use those to help calculate the locations of the actual DiaSources. We want to keep the rough nightly ephemerides around, but not necessarily at NCSA (depends on ImSim/catalog sim). We definitely do need to keep the DiaSources for the entire year around ... but these will be produced by the pipelines working on the imsim images. For DC3b, I think the plan is NOT to run MOPS in a "check completeness" mode (i.e. generating test DiaSources without going through images), so we should not need to keep a copy of those around either. check with Tim?.

Question: should ephemerides be kept "near" the MovingObject table (eg on lsst10)? We are talking about ~ 500 GB (68 bytes/row x 4 billion worst case = 272 GB + indexes)

Proposed solution: use MySQL partitioning, put each night in a different partition. 365 partitions is a reasonable number (the hard limit is 2,000). Creating many more partitions is not recommended, e.g., if we were to run production today, it'd be best to stick with 365 partitions and keep 10 nights/partition.