Column Store for GWAC: A High Cadence High Density Large-Scale Astronomical Light curve Pipeline and Distributed Shared-Nothing Database

Meng Wan[1,2,3,5], Chao Wu[1,5], Jing Wang[1,5], Yulei Qiu[1,5], Liping Xin[1,5], Sjoerd Mullender[3], Hannes Mühleisen[3], Bart Scheers[3], Ying Zhang[3,4], Niels Nes[3], Martin Kersten[3,4], Yongpan Huang[6], Jinsong Deng[1,5] and Jianyan Wei[1,5]

cwu@nao.cas.cn

ABSTRACT

The Ground-based Wide-Angle Camera array (GWAC), a part of the SVOM space mission, will search for optical transients of various types by continuously imaging a field-of-view (FOV) of 5000 degrees^2 in every 15 seconds. Each exposure consists of 36×4k×4k pixels, typically resulting in 36× ∼175,600 extracted sources. For a modern time-domain astronomy project like GWAC, which produces massive amounts of data with a high cadence, it is a challenge to search for short-timescale transients in both real time and archived data, and to build long-term light curves for variable sources.

Here we develop a high-cadence high-density light curve pipeline (HCHDLP) to process the GWAC data in real time, and design a distributed shared-nothing database to manage the massive amount of archived data, which will be used to generate a source catalogue with more than 100 billion records during ten years of operation. First, we develop HCHDLP based on the column-store DBMS of MonetDB, taking advantage of MonetDB's high performance when applied to massive data processing. To realize the real time functionality of HCHDLP, we optimize the pipeline in its source association function, including both time and space complexity from outside the database (SQL semantic) and inside (RANGE JOIN implementation), as well as in its strategy of **building complex light curves.** The optimized source association function is accelerated by three orders of magnitude. Second, we build a distributed database, using a two-level time partitioning strategy via the MERGE TABLE and REMOTE TABLE technology of MonetDB. Intensive tests validate that our database architecture is able

[1]National Astronomical Observatories,Chinese Academy of Science, 20A Datun Road, Chaoyang District, Beijing 100012, China

[2]University of Chinese Academy of Sciences, Beijing 100049, China

[3]Centrum Wiskunde & Informatica, 1098 XG Amsterdam, The Netherlands

[4]MonetDB Solutions, 1098 XG Amsterdam, The Netherlands

[5]Key Laboratory of Space Astronomy and Technology, National Astronomical Observatories, Chinese Academy of Sciences, Beijing 100012, China

[6]College of Economics, Capital University of Economics and Business, Beijing 100070, China

to achieve both linear scalability in response time and concurrent access by multiple users. In summary, our studies provide guidance for a solution to GWAC in real-time data processing and management of massive data.

Subject headings: source identification, query optimization, astronomical database, light curve pipeline

1. Introduction

1.1. GWAC scientific goals and data challenges

The Ground-based Wide-Angle Camera array (GWAC) is a set of ground based instruments under the framework of the SVOM mission. SVOM is a Chinese-French space mission dedicated to detecting gamma-ray bursts (GRBs), which has been funded by CNSA (the China National Space Administration) and CNES (the Centre National d'Etudes Spatiales) and is planning to launch in 2021(Cordier et al. 2015). GWAC is designed to comprise of 36 cameras, each with an 18 cm diameter and 12.8×12.8 degrees^2 field of view. The 36 cameras will point to the sky in different directions, and totally cover an area of more than 5000 deg^2. Each camera will take an image once every 15 seconds [10 second exposure plus 5 second readout]. GWAC will monitor simultaneously an area of sky within the field of view of ECLAIRs (Cordier et al. 2015), so that GWAC has the potential to catch the prompt optical emission of GRBs. Besides monitoring GRBs, GWAC is also able to search for other optical transients such as supernova and optical counterparts of gravitational-wave bursts.

Thanks to GWAC's 15 s exposure and large FOV, GWAC can also provide light curves with high time resolution of millions of objects in a long time scale. A light curve is a time series of light intensity, representing the magnitude of a celestial object or region, as a function of time. In light curve analysis we are interested in variable objects during periods that they show drastic changes. The huge data in the form of light curves will be used not only for studying variable stars, but also for searching for transient phenomena, such as short time-scale gravitational microlensing events and transits by extrasolar planets. It is worthy of attention that gravitational microlensing events with time scales of less than several hours are unique candidates for searching for interstellar dark objects, for example, free-floating planets. The large amounts of data representing the source catalogue and light curves produced by GWAC are an important motivation for our data analysis system.

To generate the light curves, the GWAC light curve processing system will face stringent demands on data cadence and the rate of data acquisition of GWAC. One GWAC camera will capture an image every 15 seconds. The source extraction and subsequent light curve processing of each image should be finished in a time frame of 15 seconds. **This is due to the fact that the short time-scale observing objects, such as the microlensing events, need to be**

discovered by analyzing light curve data in real time. The data rate of each camera is ∼12,000 source measurements (2.4 MB) per second, which means the total data rate is 85 MB/s for the whole GWAC array system. GWAC will produce ∼ 2.7 TB catalogues data per day, and ∼ 9 PB over the designed 10-years of operation.

1.2. Comparison with other surveys.

The massive catalogue and light curves data produced by GWAC drives us to use relational databases to realize data manipulation and query, which is a popular solution for similar modern time-domain survey projects.

Distributed relational database systems (RDBMS) are widely used to manage large scale observational data produced by large aperture wide-field surveys, such as Pan-STARRS and LSST. Pan-STARRS (Kaiser et al. 2002; Burgett 2012) is designed to collect data at a rate of 3 ∼ 10 terabyte (TB) per night. The observed data of Pan-STARRS are managed in a distributed relational Microsoft SQL server database, which is spatially partitioned into slice databases using a hash function over the spatial location (RA and DEC) of each detection (Simmhan et al. 2011). The raw imaging data of LSST is expected to be about 15 TB per night (Ivezic et al. 2008). Over the ten years of LSST survey operations, LSST will result in over 50 PB for the catalogue databases (Jurić et al. 2015). To manage the massive amount of astronomical catalogue, LSST developed a special distributed shared-nothing SQL database query system, called Qserv, which is independent of a particular RDBMS (Wang et al. 2011; Becla et al. 2013; Becla and Wang 2014).

In addition to managing the large amount of observational data, the RDBMSs also play an important role in the data processing pipeline of radio transient search projects (Norris 2010), such as VAST (Variables and Slow Transients) of Australian Square Kilometer Array Pathfinder (Murphy et al. 2013) and LOFAR (the LOw-Frequency ARray) (Van Haarlem et al. 2013). The transient processing of VAST is implemented as a real-time pipeline, which employs PostgreSQL database with Q3C (Koposov and Bartunov) plugin to optimise coordinate searches and cross-matches (Banyer et al. 2012). Its source data rate is ∼ 12,000 source measurements per second. The transient pipeline TraP (Scheers 2009; Swinbank et al. 2015) of LOFAR is implemented in real-time processing, which is developed on the column-store database MonetDB (Scheers et al. 2012) [1]. Its source data rate is ∼10-10,000/sec. Pioneering the columns-store technology (Abadi et al. 2013) since 1993, MonetDB has achieved significant speedup compared to traditional databases by innovations at all layers of a DBMS (Idreos et al. 2012, 2007; Manegold et al. 2009). In column-store, queries only touch the relevant columns, and when in contiguous memory it allows compression and good cache-hit ratios. Furthermore, MonetDB's kernel is a programmable relational algebra machine operating on "array"-like structures, exactly what CPUs are good at. Thanks to the

[1]Centrum Wiskunde and Informatica (CWI), the Netherlands. www.monetdb.org

high performance computing capability provided by the MonetDB database, the TraP pipeline has successfully realized transient search and light curve generation in real time through technology of table driven logic of MonetDB.

The success of LOFAR motivates us to employ MonetDB as the database platform for the GWAC high cadence high density light curve pipeline (HCHDLP), because the scientific goal and data processing strategy of GWAC are similar to those of LOFAR. This satisfies one of Jim Grays laws: large-scale scientific database should bring computations to data, rather than data to computations (Gray et al. 2005). In addition, the latest distributed technology of MonetDB (see discussion in §3) can support distributed data architecture, which suits our long term data storage and query. **HCHDLP of GWAC is not based on the heritage of either Qserv of LSST or Pan-STARRS because they are not using database-centric computing approaches for their dynamic pipelines during data production.**

The organization of the paper is as follows: Section 2 describes the overview of HCHDLP and optimization techniques to achieve real-time performance for each camera. In section 3, we design a shared-nothing distributed database on top of the GWAC camera array. In section 4, we present the evaluation of the light curve pipeline in terms of functionality and performance. We discuss future research directions in section 5.

2. MonetDB-based HCHDLP

Two main goals of the HCHDLP are: 1) to manage a catalogue that contains all the individual measurements of the sources observed by GWAC. 2) to create cumulative light curves to study variable sources and transients (microlensing events etc.) in real-time.

We design the HCHDLP based on the following requirements and constraints of GWAC: Firstly, the total processing time of an image should not exceed the time between two subsequent images, i.e. 15 seconds. Secondly, the data rate needed to be processed by HCHDLP is $\sim$ 12,000 sources/sec. Finally, the design of HCHDLP should satisfy the boundary conditions given by the scientific requirements and hardware architecture.

The GWAC hardware architecture is illustrated in Figure 1. There are 9 mounts totally, and each mount consists of 4 CCD cameras, where each camera uses a dedicated database server. The HCHDLP database is temporary; the long-term database is a distributed shared-nothing database, which is illustrated in Figure 6 in §3.

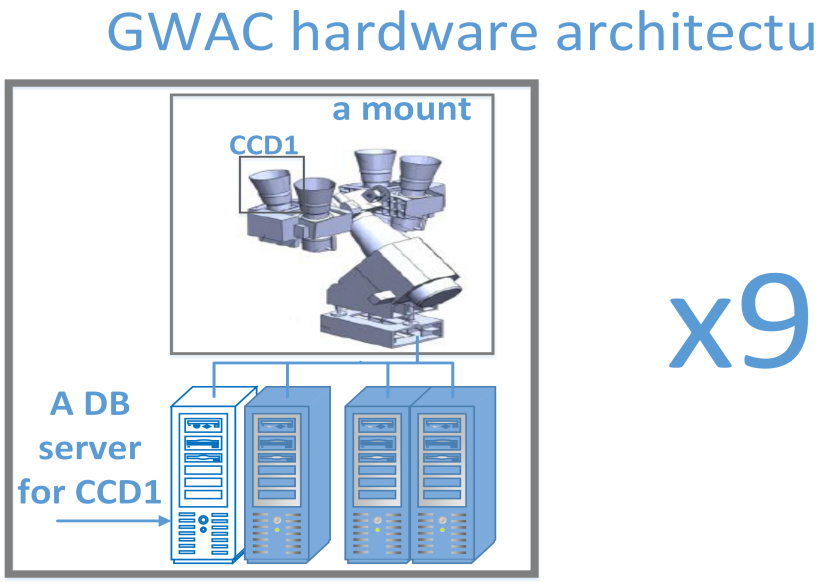

Fig. 1.— GWAC telescope array and its computing cluster, in which each CCD (telescope) has a dedicated database server, e.g., CCD1 has an accompanying database server displayed in white, and so on. The HCHDLP database is temporary where the bulk of in-database processing takes place to achieve real-time data processing, with only limited disk capacity. The catalogue and light curve production of the temporary database will be periodically exported to a long-term archival catalogue database. The long-term database is a distributed shared-nothing database illustrated in Figure 6.

The data flowchart of GWAC is shown in Figure 2. The input to HCHDLP is a stream of catalogue data after image preprocessing, quality control, source extraction and flux calibration of pipeline 0. Firstly, the CCD reductions and astrometry calibration are processed in the image preprocessing. Then, the quality control filters out the bad quality image according to star profiles, star numbers and astrometry accuracy. After quality control, the images of good quality are taken to source extraction procedure. Then, zone ID and Cartesian coordinates are calculated from equatorial coordinates. Finally, the flux calibration is carried out through comparison to standard stars in the same frame. The standard stars are referenced from the UCAC4 catalogue (Zacharias et al. 2013). The resulting source attributes of the final calibrated catalogue are listed in table 1. All sources are in the form of point-like sources and will be loaded into the *target* table in Figure 3.

Name	Type	Description
ID	long int	Every inserted source/measurement gets a unique id, generated by the source extraction procedure.
imageid	int	The reference ID to the image from which this sources was extracted.
zone	smallint	The zone ID in which a source declination resides, calculated by the source extraction procedure..
ra	double	Right ascension of the a source (J2000 degrees), calculated by the source extraction procedure.
dec	double	Declination of a source (J2000 degrees). as above.
mag	double	The magnitude of a source.
mag_err	double	The error of magnitude.
pixel_x	double	The instrumental position of s source on CCD along x.
pixe_y	double	The instrumental position of s source on CCD along y.
ra_err	double	The 1-sigma error on ra (degrees).
dec_err	double	The 1-sigma error on declination (degrees).
x	double	Cartesian coordinates representation of RA and declination, calculated by the source extractor procedure.
y	double	Cartesian coordinates representation of RA and declination, as above.
z	double	Cartesian coordinates representation of RA and declination, as above.
flux	double	The flux measurements of a source, calculated from the mag value.
flux_err	double	The flux error of a source.
calmag	double	calibrated mag.
flag	int	The source extraction uses a flag for a source to tell for instance if an object has been truncated at the edge of the image.
background	double	The source extraction estimates the background of the image.
threshold	double	The threshold indicates the level from which the source extraction should start treating pixels as if they were part of objects.
ellipticity	double	Ellipticity is how stretched the object is.
class_star	double	The source extractions classification of the objects.

Table 1: Measured attributes of each source.

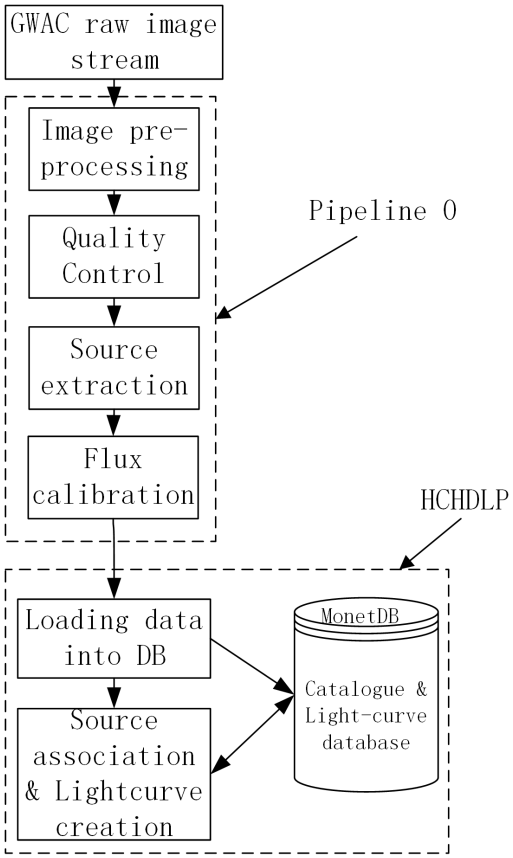

Fig. 2.— The data flowchart of the GWAC. After the preceding "source extraction" module is applied, detections and measurements related to celestial objects in the images are extracted into catalogue files. **In the last procedure of pipeline 0, flux calibration is carried out through comparison to standard stars in the same fields. The standard stars are selected from the UCAC4 catalogue.** Then in the HCHDLP phase, the catalogue files are inserted into the HCHDLP database, where they are associated with an existing sky-model *uniqcatalog* table one by one to form light curves.

The HCHDLP consists of two procedures, i.e., data loading, and source association & light-curve creation. In order to load large amounts of data quickly, binary bulk loading is adopted to ingest the point-like source catalogues into MonetDB using a parallel data insertion command. The source association identifies every source detected by GWAC, and concatenate all current and archived measurements of each identified source in time series, resulting in the light curves.

Due to the high cadence, a lot of the data processing is shipped to the database engine. Database algorithms take care of source associations. Figure 3 shows the ERD (Entity Relationship Diagram) of five key tables and their one_to_many relationships.

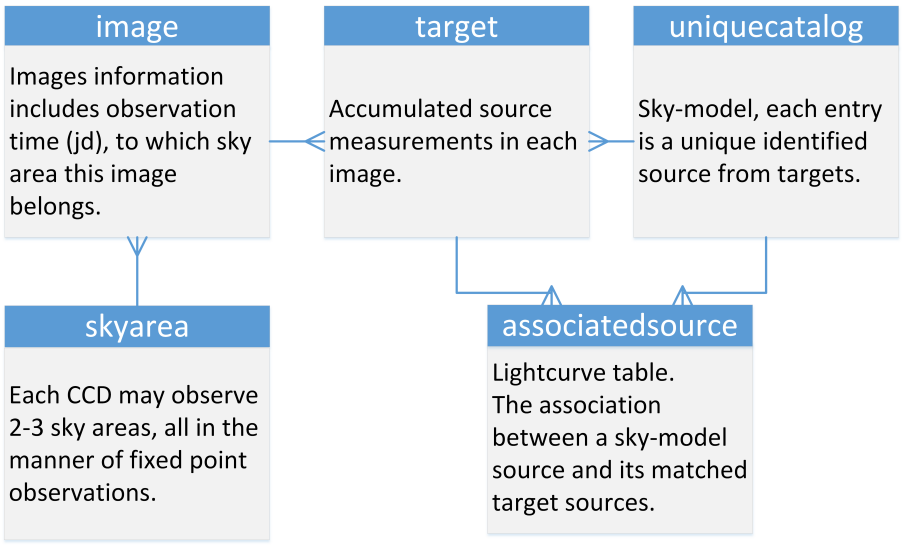

Fig. 3.— Simplified Database Schema.

It is worth noting that the measured coordinates of sources are taking two forms simultaneously: two components in the form of spherical coordinates representing R.A. and DEC and three components of Cartesian coordinates in the form of unit vectors representing x,y, z, in order to save time during complicated source association calculations.

In practice, when a new source is loaded into the *target* table, it is assigned a new permanent ID *ID_target* by primary key as its unique identifier. Then the new source is associated with previous records of sources in the *uniquecatalog* table, in which each source has been assigned an *ID_uniq* . If the new source is a good match with a record in *uniquecatalog*, the associated IDs of *ID_target* and *ID_uniq* are appended into the association table *associatedsource* which is a de-facto light curve table. If the match failed a new record is created in uniquecatalog. The related auxiliary information is recorded in the table *image* and *skyarea*.

An association pair is acceptable if the angular distance between the two sources is smaller than a given tolerance radius. Generally speaking, the choice of the tolerance radius is mainly based on the uncertainty of the astrometry calibration. However, for the GWAC system, due to its large pixel scale of ∼12 arcsecond per pixel), the crosstalk between sources shall be a problem when getting positions of these sources in the CCD image, especially for those objects with low signal to noise ratio (Dudik et al. 2012). In these sub-sampled images, the point spread function (PSF) of the sources would be relatively stable since it is less sensitive to variation of the environment, like the seeing or temperature . Thus, in our practical experiments, 2.5 pixels , which is ∼1.4 times of the typical PSF value (∼1.8 pixels), is a proper choice for GWAC images when doing the cross-match.

From our analysis, the mismatch rate is ∼ 0.05% (considering the flux constraint) with our above tolerance radius, while the mismatch rate is <0.1% if not considering

the flux constraint. This mismatch rate is acceptable for GWAC taking into account the balance with data processing speed. Additionally, the color constraint in cross-match can be ignored since all the data used for the association are acquired from same camera and same band.

The HCHDLP database uses the popular zone algorithm (Gray et al. 2007) to speed up processing. Its basic idea is to map a sphere into equally spaced declination zones. With the filter of ZoneID, the strength of the zone algorithm is from its simplicity and the locality it produces: a zone only has two neighbours when matching two datasets. We only need to look for matches within the same ZoneID and its neighbours. The optimal zone height is set to the value of tolerance radius. Either a "tall" or "short" zone heights will cause more neighbors that need to be joined with the center zone. These extra areas add extra costs that outweigh the savings in pair-wise comparisons (Gray et al. 2004). The search radius can be easily set at any time as an input parameter of the association function.

2.1. Optimization of source association

A straightforward zone-based source association query that joins two tables by using the Euclidean distance would run for a very long time on high cardinality datasets. Among all the relational algebra operators in the association, the *Join* operator is the most expensive one. In the *Join* operation, each zone in the left reference table "uniquecatalog" needs to be compared with all the zone values in the right "target" table one by one. The zone search results must be further reduced by an *ra* filter of the *Alpha(theta, decl)* computation. The inflated radius *Alpha* function can compute the limiting *ra* ranges of points in all regions both near the equator and near the poles (see details in (Gray et al. 2007)). Then a quick filter on *dec* is tested and finally a careful Euclidean distance is computed. The time complexity of this operation is $\mathcal{O}(n^2)$. For example, this straightforward query takes several hours to associate two tables each with ∼175,600 tuples. The SQL extract below is the straightforward source association query.

```
SELECT u0.id AS uniqueid, ...
, t0.id AS targetid, ...
3600*DEGREES(2*ASIN(SQRT( (u0.x-t0.x) * (u0.x-t0.x)+(u0.y-t0.y)*(u0.y-t0.y)+
(u0.z-t0.z) * (u0.z-t0.z))/2)) AS distance_arcsec
FROM uniquecatalog as u0, targets as t0
WHERE u0.zone BETWEEN cast(floor((t0."dec" - radius )/ zoneheig) as integer)
AND cast(floor((t0."dec" + radius )/ zoneheig) as integer)
AND u0.ra_avg between t0.ra - alpha(t0."dec", radius) and t0.ra + alpha(t0."dec", radius)
AND u0.decl_avg between t0."dec" - radius and t0."dec" + radius
AND u0.x*t0.x+u0.y*t0.y+u0.z*t0.z > cos(radians(radius));
```

To significantly reduce the long time required for source association, we optimized the association for both time and space complexity by two means: from outside the database (SQL clauses) and inside the databases (query optimizer).

From outside database engine, in a SQL perspective, since the sources stored in the table *target* are unordered, this is also true for the zoneid column, which leads to the zoneid lookup being a random access pattern. In case that the randomly accessed data are too large for the CPU caches, the random access will cause cache misses and performance degradation (Boncz et al. 1999). For this reason, we create a sorted version of the inner relation *uniquecatalog* ordered by the predicate column "zone" to mimic a clustered index before the join phase:

```
CREATE TABLE u0_zone AS SELECT * FROM uniquecatalog ORDER BY zone with data.
```

This has the advantage of scanning the outer relation sequentially. The time complexity of sorting the outer relation is $\mathcal{O}(nlogn)$, where n is the size of the outer relation.

To reduce the amount of calculation and the space complexity, we use the standard SQL "WITH" clause, to simplify complex SQL by materializing subqueries, which saves the MonetDB from recomputing multiple times. Although both the "WITH" clause and a temporary table can improve query speed for complex subqueries, the former has some advantages over the latter : 1) "WITH" queries are treated as inline views without extra effort to remove the temporary tables after usage; and 2) WITH supports multiple subquery and mutual references between subqueries. WITH allows assigning a name to a subquery block. This name can be referenced in multiple places in the main query or even in the following WITH subquery. By using WITH clauses, we materialize the intermediate results $target.\text{“}dec'' - \ radius\ as\ decmin$ and $target.\text{“}dec'' + \ radius\ as\ decmax$. The derived decmin/decmax are used in the next "WITH" subquery to compute the zone range: $decmin/zoneHeight\ as\ zonemin$ and $decmax/zoneHeight\ as\ zonemax$. Zonemin and zonemax together define a zone range. The search radius can be flexibly changed according to the various different sky region. **The core SQL extract of *associates* operator is listed below.** Our test (§4.3.1) shows that the "WITH" optimization can bring the source association procedure for two tables with 175,597 rows × 175,540 rows from resource exhaustion (running out of disk space) to 4m 2s.

```
CREATE FUNCTION Alpha(theta double, decl double) returns double
BEGIN
    IF abs(decl)+theta > 89.9 then return cast(180.0 as double);
    ELSE
    RETURN (degrees(abs(atan(sin(radians(theta)) /
                             sqrt(abs( cos(radians(decl-theta))
                                     * cos(radians(decl+theta))
           )       )   )   )     ));
       END IF;
END;
```

```
CREATE FUNCTION associates(imageno int, radius double)
RETURNS TABLE (uniqueid bigint, targetid bigint, distance_arcsec double...)
BEGIN
  DECLARE TABLE u0_zone (LIKE uniquecatalog);
  DECLARE zoneheig double;
  SET zoneheig=1e1/3600;
  INSERT INTO  u0_zone SELECT id,targetid,... FROM uniquecatalog  ORDER BY zone;
  RETURN TABLE(SELECT uniqueid,targetid,distance_arcsec...FROM (
 WITH x as (SELECT target.id,
    target."dec" - radius as  decmin,
    target."dec" + radius as  decmax, ...
  FROM target4 as target
  WHERE target.imageid = imageno),
 smallt as (select x.id, x.decmin, x.decmax,
    cast(floor( x.decmin / zoneheig) as integer) as zonemin,
    cast(floor( x.decmax / zoneheig) as integer) as zonemax, ...
 FROM x)

SELECT u0.id as uniqueid,t0.id as targetid ....
  FROM  u0_zone as u0, smallt as t0
        --The ‘‘implicit join notation’’ using commas to separate tables
        --and the CROSS JOIN are semantically identical.
 WHERE u0.zone BETWEEN zonemin AND zonemax
    AND u0.ra_avg BETWEEN t0.ra-alpha(t0."dec", radius) AND t0.ra+alpha(t0."dec", radius)
    AND u0.decl_avg BETWEEN t0.decmin AND t0.decmax
    AND u0.x*t0.x+u0.y*t0.y+u0.z*t0.z > cos(radians(radius))
) AS ut);
end;
```

From inside the database engine, the optimization of database implementation can speedup a RANGE-JOIN significantly. There are three expensive "range-joins" predicates in the above *associates* function i.e., the three BETWEEN...AND in WHERE clause. RANGE-JOINs are queries with inequality predicates (greater than, less than, or between) on which a column from the left table is restricted to be in a range specified by two columns of the right table. Because the condition with the highest discarding rate is a RANGE-JOIN on the zone column, it should be given high priority in terms of optimization.

We have optimized the RANGE-JOIN implementation of the MonetDB by employing a quick binary search and compressed imprints index. The optimization has been introduced specially for this work, however, it is generally applicable for all similar types of queries. The imprints index is already available in MonetDB but was not applied to RANGE-JOIN operations. This work has extended the imprints index to also work with RANGE-JOINs. (Sidirourgos and Kersten 2013) developed a novel cache-conscious secondary index structure called *imprints* to speed up scans

over large tables stored in MonetDB. It is designed such that any clustering or partial ordering is naturally exploited without the need for extra parametrization. If the left column is sorted, we use binary search, which allows the majority of the table to be skipped to reduce join time by orders of magnitude instead of "brute-force" for large tables. Figure 4 illustrates an example of the binary search optimization. The table *u0_zone* with a column *zone*, joins with table *smallt* using a BETWEEN predicate (zone BETWEEN zone_min AND zone_max). When the zone range of the BETWEEN predicate is (790, 1001), only the tuples of zone values 800, 900 and 1000 (within the zone range) are quickly selected and all the other tuples are skipped. If the left column is unsorted, we use imprints under three conditions: 1) the data type is right for imprints. All numeric types (integers of all widths and floating points) and types that are internally represented as integers, such as dates and timestamps are suitable for imprints; 2) the left column is either persistent or already has imprints; 3) the right column is long enough so that it is worth the effort of creating imprints. Without sorting the order on the ra_avg and decl_avg columns, the other two range join predicates: *u0.ra_avg between t0.ra-alpha(t0."dec", radius) and t0.ra+alpha(t0."dec", radius)* AND *u0.decl_avg between t0.decmin and t0.decmax*, will use imprints to minimize data access.

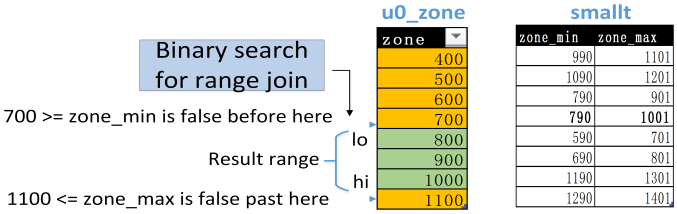

Fig. 4.— optimized RANGE-JOIN in MonetDB.

The time complexity includes two parts: the complexity of sorting the left column is $\mathcal{O}(|l|\cdot\log|l|)$, and the complexity of range join:

$$T(|l|,|r|) = \begin{cases} |r|\cdot\log|l|, & \text{if } l_{sorted} \\ C\cdot|r|\cdot|l|, & \text{if } l_{imprints} \\ |r|\cdot|l|. & \text{if nested loop} \end{cases}$$

where C is the size of imprints of the left column. C is $\ll 1$ in a typical case, and $\frac{1}{16}$ in the worst case. Without optimization, direct scan is employed using a nested loop, so C=1. As shown by our test, the optimization of the RANGE-JOIN implementation can reduce the source association procedure from 4m 2s further down to a few seconds, giving a speedup by a factor of 220. (see §4.3.1 for the detailed performance testing).

2.2. Optimization of One_to_many Match Type

Figure 5 illustrates a simplified version of HCHDLP, which includes source associations and subsequent processing of associated relationships in four types. Among these types, the one_to_many relationship is the slowest one. This relationship is caused by either a new image with higher spatial resolution or a newly detected source. When a one_to_many relationship occurs, the *associatedsource* table is traversed twice originally in TraP (Scheers 2011). The size of the *associatedsource* table is a function of number of sources of the sky-model *uniquecatalog* table : $D_{assoc} = 175600 \times 26 \times 2400 \times 36 \times n = 367 \times n$ (GB), where n is the number of observation days. For instance, at the epoch of image 5000, the *associatedsource* table has 1+ billion rows. The EQUI-JOIN with the *associatedsource* table on a predicate of equal *uniqueid* takes 20.1s to return 41,894 rows.

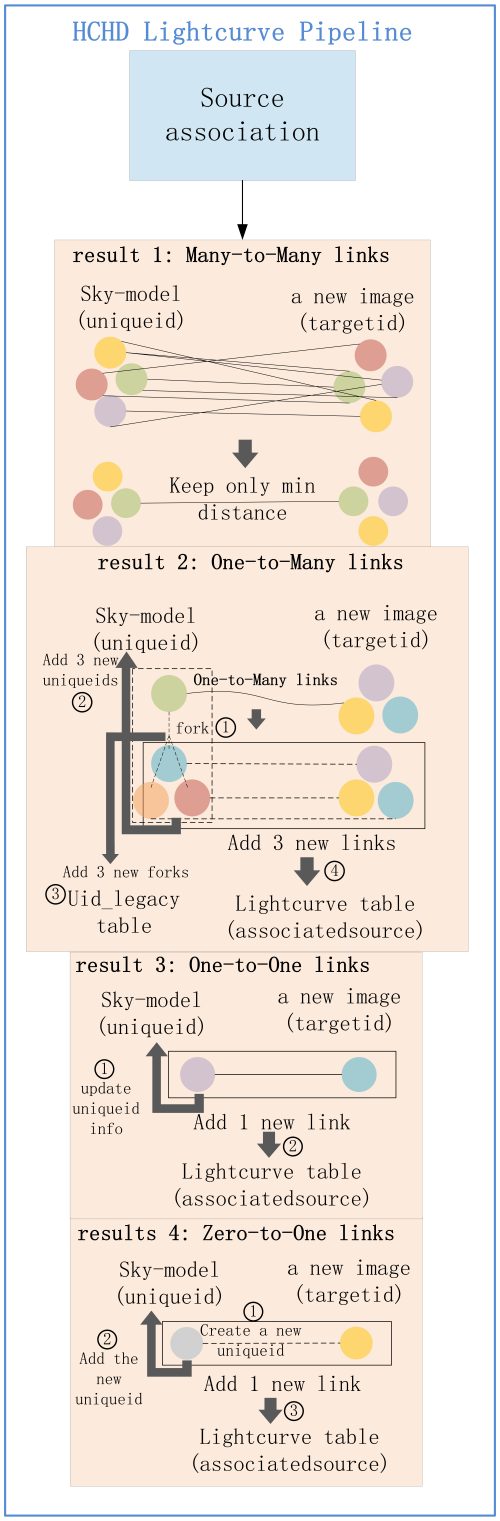

Fig. 5.— HCHDLP displays how to process four types of source association results. Many_to_many links are replaced with the link with minimal distance to reduce overhead. One_to_many links are replaced with many new links, which are inserted into the the light curve table. Old *uniqueids* and their successors relationships are inserted into the *uid_legacy* table. One_to_one links are previously found relationships. Zero_to_one links are newly detected targets which have no previous assigned *uniqueid*.

Since the EQUI-JOIN with *associatedsource* is so expensive, we have changed the strategy of forming light curves of the one_to_many type to build the pipeline that can operate in real time .

1) We drop uniqueness constraint by removing the primary key and foreign key on the large tables during the run time of HCHDLP [2]. During idle time, the check is instead carried out by an SQL query, and the primary and foreign keys are added back to maintain the database consistent.

2) In the one_to_many case, the old relationship between the *uniquecatalog* and the *target* table is replaced by many new ones through "forking" (see figure 5 for the details). However, during the replacement, DELETE DML (deleting the old relationships from the big table *associatedsource*) is expensive. We avoid DELETE [3] by INSERT the old relationships to a small auxiliary table, *uid_legacy*. The *Uid_legacy* table stores deleted old uniqueid and their children (replacement) uniqueids in the one_to_many scenario and it will exist for all time as a historical record of changes.

3. Shared-nothing Distributed HCHDLP Database Architecture

The catalogue and light curve production of the temporary HCHDLP database will be periodically exported to the long-term distributed database. The shared-nothing distributed database is the best solution to store the final products of HCHDLP over a long term thanks to an independent hardware and software architecture. The architecture is adopted by taking into account of the GWAC's top level pointing strategy: there is no source association between the available sky regions of different CCDs, which is guaranteed by the GWAC's pointing strategy.

---

[2]applicable to all the match types of relationships

[3]Frequent UPDATE and DELETE DML operations have been shown to be very expensive on a petabyte system (Becla et al. 2010), so we tried to constantly apply the "write-once and never-update" and "append-only" principle to the big light curve table which greatly optimize HCHDLP throughput and latency.

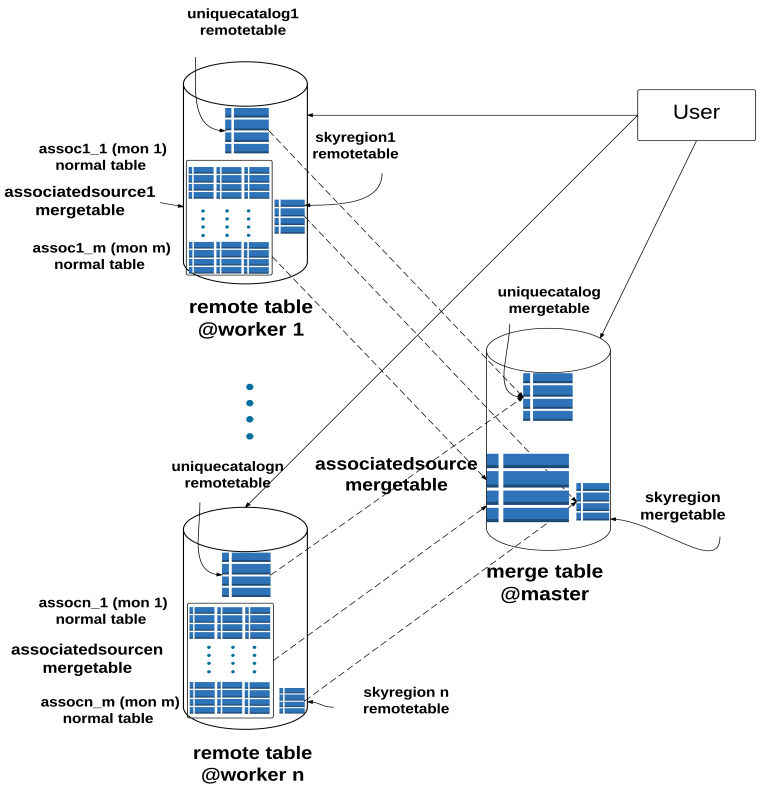

Fig. 6.— The distributed GWAC database architecture is comprised of one master node and multiple worker nodes. Remote tables associatedsource1..n, uniquecatalog1..n on individual worker nodes are mapped to the mergetable associatedsource and uniquecatalog on the central server. Within a worker node, associatedsource1..n table can be further partitioned by time, i.e., by months or weeks or a couple of days. This time partitioning of light curve table provides coarse temporal indexing and finer data locality, so that light curve queries involve only the relevant temporal partitions.

Figure 6 illustrates the shared-nothing distributed database architecture comprised of one master node and multiple worker nodes. The initial number of worker nodes should be the same as the number of CCDs. Each remote table on a worker node is mapped to the merge table with the same name as the master. In terms of the MERGE TABLE+REMOTE TABLE of MonetDB, MERGE TABLE technique is a horizontal data partitioning method. The technique allows a table to be defined as the union of its partitions, and enables finer control of data locality during query evaluation. As complements of the merge tables, the REMOTE TABLE technique allows the partitions of a merge table to reside on different nodes. Queries involving remote tables are

automatically split into subqueries by the merge table and executed on the remote tables. REMOTE TABLE adopts a straightforward master-worker architecture: one can place the partition tables in different databases, and then concatenate everything together in a MERGE TABLE in the master database. Not only REMOTE TABLEs on another node, but also local normal tables can be added to be part of a MERGE TABLE. MERGE TABLE supports nested merge tables, i.e., a MERGE TABLE can also contain other MERGE TABLEs. The MERGE TABLE and its partitions can be queried both individually and jointly.

In order to improve the performance of data querying, a remote table within a worker node can be further partitioned according to the ratio of the real RAM size and the data product size. The time span of each partition ranges from a couple of days to months.

There are a few sources that fall out of the boundaries of the observation areas, but the marginal sources are few. Firstly, according to the share-nothing design, sources of one CCD camera will not be associated with those of other cameras. Secondly, GWAC survey strategy and its pointing and tracking system can ensure that each CCD observes a few fixed sky areas in the long term and the drift of pointing center is controlled within 10 pixels. Therefore, if a marginal source is detected in the area of a CCD, it will be added to its catalogue, and if the source moves out, it might appear on another area, it will be added to another catalogue and traced. It will be flagged in the *flag* attribute of the sources catalogue (see table 1).

4. Experiments and evaluations

This section describes our tests on the performance of both HCHDLP and shared-nothing distributed database, including source association optimization on MonetDB and its comparison with PostgreSQL, HCHDLP running and light curve queries.

4.1. Hardware and software configuration for HCHDLP

Our experiments run on a cluster of six server nodes of the CWI Scilens cluster platform interconnected via InfiniBand 40 Gb/s links. Each node has two sockets 32 hyper-threaded cores that use Intel Xeon CPU E5-2650 v2 @ 2.60GHz, with 256 GB main memory and 5.4 TB of storage on software RAID0 containing 3 disk drives per server. Tests are conducted on MonetDB default (development) branch v11.22.0, hg id 3603a1af9790. Optimized compilation of MonetDB is activated in our tests.

4.2. Data and Loading

A hindrance for large scale adoption of DBMSs in handling astronomical catalogues is the time it takes to bulk-load the data into the databases. Our experiments are based on the *target* table. It is ingested by simulated catalogue files, whose size are proportional to the observation time. Simulated catalogue files represent pseudo-sources extracted after pipeline 0. The catalogue files are synthesized by adding positions and flux noises to a template catalogue file extracted from the UCAC4 catalogue (Zacharias et al. 2013). Typically each image is set to have $\sim$ 175,600 sources, and each source has 22 column attributes, which means an increment of *target* table is $\sim$ 79 GB per night (with a cadence of 15 s and 10 hr of observation per night) per CCD. We produce simulated catalogue files in batches and load them into the *target* table before the HCHDLP system starts testing.

Binary bulk loading (MonetDB 2016) is used to load simulated catalogues into MonetDB. The SQL COPY command can take a complete ASCII file and insert the data in one go using all system cores in parallel. Furthermore, MonetDB created a binary bulk loading method, the binary version of the COPY command: COPY BINARY. When large tables are needed to load into a database, the binary bulk loading is slightly faster. This saves rendering of data into ASCII and subsequent parsing of the data being exchanged, and 'attach' it to the SQL table. To illustrate, the SQL query below is used to load the binary column files into *target* table.

```
COPY BINARY INTO target FROM ('path\_to\_column\_file\_i', ..., 'path\_to\_file\_f').
```

Each attachment file is produced by our simulator program that writes the binary version of the columns files directly into the disk. All the binary column files are aligned, i.e. the i-th value in each file corresponds to the i-th record in the table. The files with numeric data are moved into place to avoid copying (MonetDB 2016).

4.3. Performance of HCHDLP

4.3.1. Source Association

At the beginning, with the outside database optimization (i.e., sorted zone and "WITH" clause), the execution time of the source association for two tables of $\sim$175,600 rows is as short as 4m 2s [4]. Then, the inside database optimization further reduces the time of source association from 4m 2s to 1.1s, with a 220x speedup. The association radius is set to be 6 arcsec, the same

[4]Without the optimization, the source association failed in the same environment due to a huge crossproduct running out of disk space, and succeeded in a more powerful environment with 20TB disk space in 59m.

as the zone height that is a parameter in the zone algorithm. [5] Since the highest discarding rate comes from the RANGE-JOIN on zone column, the 220x speedup is mostly from the binary search on the sorted zone column. The following is the source association query by calling the *associates* function.

```
sql> INSERT INTO tempuniquecatalog SELECT * FROM associates(2, 0.0016666666666666666);
179,769 affected rows (1.1s)
```

The accuracy validation of our optimization is performed by both self-join and cross-join of source association and by checking if the correct number of new sources could be found. For a self-join with a radius of 0.36 arcsec, the sources in a simulated catalogue with 175,597 rows all match themselves. The related code is listed as follows:

```
--self-join
sql> INSERT INTO tempuniquecatalog SELECT * FROM associates(1, 0.0001);
175597 affected rows (663.816ms)
```

For cross-join, the sources in two simulated catalogues with 175,597 rows × 175,540 rows are cross-matched with a radius of 5 arcsec. The cross-join returns 27 unmatched sources because their distance from other sources are above 40 arcsecs. Again, the related code is listed as follows:

```
--cross-join
sql> INSERT INTO tempuniquecatalog
       SELECT * FROM associates(2, 0.0013888888888888888888888888888889);
176539 affected rows (1.1s)
--sanity check
sql>INSERT INTO newsrc SELECT t.id AS targetid
      FROM target t LEFT OUTER JOIN tempuniquecatalog tuc
      ON t.id = tuc.targetid
      WHERE t.imageid = 2 AND tuc.targetid IS NULL ORDER BY t.id;
27 affected rows (96.168ms)
```

[5]This association radius here is set to be half pixel (11.7 arcsecond/pixel). **The uncertainty of object position is 0.1 pixel for our simulated catalogues. We employ 0.5 pixel (5 σ) as the tolerance radius here, which is consistent with our above PSF method of the tolerance radius selection in practice (see §2).**

4.3.2. Comparison With PostgreSQL

The performance of source association based on the column-store database (MonetDB) is compared with a traditional row-store database (PostgreSQL) involving PostGIS. PostGIS [6] is an extension to PostgreSQL which allows it to handle and process geographic data through GiST-based R-Tree spatial indices. Gist (Generalized Search Tree) index is the most suitable for querying spatial data. GiST is used to speed up searches on all kinds of irregular data structures (integer arrays, spectral data, etc) which are not amenable to normal B-Tree indexing (Hellerstein et al. 1995). Although PostgreSQL supports three kinds of indices by default: B-Tree, R-Tree, and GiST, the GiST index is adopted in the current study. At first, B-trees are hard to deal with when applied to a two dimensional sky image since they were originally designed for one dimensional, linearly ordered key spaces. Secondly, R-Tree implementation is not as robust as the GiST. For example, GIS objects larger than 8K will cause the building of R-Tree index to fail. Finally, GiST index can support nearest-neighbor search over large datasets, which is close to the purpose of source association (Kornacker 1999). The queries run on PostgreSQL are identical, with modulo syntax differences. The code below shows the experimental procedures using PostgreSQL.

```
associates function experiment on PostgreSQL 9.4.4.

0. prepare data (the source lists are produced )
gwacdb=# COPY targets(imageid,zone,ra,dec,mag,pixel_x,pixel_y,ra_err,dec_err,x,y,z,flux,
flux_err,normmag,flag,background,threshold,mag_err,ellipticity,class_star,orig_catid)
from '/scratch/meng/gwac/RA240_DEC10_sqd180-ccd4-86401.cat'  delimiters ' ' CSV;
COPY 175540
gwacdb=# COPY uniquecatalog(id,targetid,ra_avg,dec_avg,flux_ref,datapoints,zone,x,y,z,
INACTIVE) from '/scratch/meng/gwac/RA240_DEC10_sqd180-ccd4-86402.cat'  delimiters ' ' CSV;
COPY 175591

1. without tuning
gwacdb=# INSERT INTO tempuniquecatalog SELECT * FROM associates(2, 0.0016666666666666666);
INSERT 0 175123
Time: 4456163.234 ms

2. with tuning
By default, postgresql uses only a single core and only 128MB disk cache is
available to a single query.
## tuning for our machine 256 GB of RAM
#shared_buffers       = 170GB
#effective_cache_size = 254GB
#maintenance_work_mem = 42GB
#work_mem             = 160GB
```

[6] http://postgis.net/

```
## The linux system shared memory parameters are configured to
#kernel.shmmax=188978561024 (176GB)
#kernel.shmall=188978561024
#kernel.shmmni=22528.

gwacdb=# INSERT INTO tempuniquecatalog SELECT * FROM associates(2, 0.0016666666666666666);
INSERT 0 175123
Time: 2958910.701 ms

3. with GIST index and the tuning above
3.1 create geometry columns
alter table target add column geo geometry;                                    0.0179
alter table uniquecatalog add column geo geometry;                            0.01395
update target set geo=st_makepoint(x,y,z);                                      3.52
update uniquecatalog set geo=st_makepoint(x,y,z);                               1.46
3.2  create gist indices
create index target_gist on target using gist(geo);                             3.37
create index uniquecatalog_gist on uniquecatalog using gist(geo);               1.73
3.3 source association with radius=6 arcsec
gwacdb=# select count(*) from uniquecatalog c, target t                         3.89
where ST_3DDFullyWithin(c.geo, t.geo, radians(0.0016666666666666666)) and t.imageid=2;
Total                                                                          14.00
```

Table 2 compares the runtimes of source association in MonetDB and in PostgreSQL. All tests are based on the cross-match of two tables with 175,597 and 175,540 rows. The tuning parameters for PostgreSQL database server are $shared\_buffers = 170GB, effective\_cache\_size = 254GB$ and $work\_mem = 160GB$. The linux system shared memory parameters for PostgreSQL are $kernel.shmmax = kernel.shmall = 188978561024(176GB)$ and $kernel.shmmni = 22528(B)$. Note that only one process per database session can be utilized by PostgreSQL, so a single complex and CPU-intensive query is unable to use more than one CPU. In order to do a fairer comparison to PostgreSQL, a single-core performance test of MonetDB is done by setting "sequential_pipe" as the SQL optimizer pipeline. Sequential_pipe is to let mserver5 avoid using parallelism. The average runtime: 1.7 second. The single core performance shows MonetDB is faster than PostgreSQL not only due to MonetDBs ability to harness multiple cores, but mainly due to the Range-Join optimization we applied for it.

Source Association Queries	Mean time (second)	
MonetDB with "WITH" clause	242	
MonetDB with Range-Join optimized	1.1	
PostgreSQL	4456	
PostgreSQL tuned	3074	
PostgreSQL tuned and with GiST index	create index	10.08
	query	3.89
	sum	13.97

Table 2: Source association query speed comparison between MonetDB and PostgreSQL. Both under optimization or indexing, MonetDB is 3.54x faster than PostgreSQL even though extra time for creating GiST index is not taken into account in PostgreSQL. It's important to note the GiST index will be updated every time new data is inserted, so data loading performance will become slower and hence this will significantly delay the whole HCHDLP.

By comparison, MonetDB (1.1s) is 3.54x faster than PostgreSQL (3.89s) even though the time for creating index in PostgreSQL is not included. It's worth noting that the GiST index is needed to be updated when new data are inserted every 15 seconds (the cadence of GWAC). Such a frequent update causes that the index building and data loading become slow, which finally significantly degrades the pipeline performance.

4.3.3. Performance of HCHDLP

In principle, the sky-model *uniquecatalog* table grows over time due to new detections of events, e.g., optical transients, asteroids and artificial objects, which will increase the execution time of HCHDLP. In order to test how the HCHDLP will perform in an extreme case, we carry out stress tests on HCHDLP by increasing the *uniquecatalog* table size through an enhanced artificial mismatch rate. The artificial mismatch is created by displacing the position of each source in *uniquecatalog* through a random offset with a Gaussian distribution to produce simulated catalogues. Therefore, an extremely high number (∼13,000) of new simulated sources is added to *uniquecatalog* per day. In practice, taking into account optical transients or other moving objects, all noises and false positives, the upper limit of new objects of GWAC is less than 1000 per day. Since the HCHDLP's main scientific goal is to manage light curves of known objects, we will clean all of above kinds of sources out of the *uniquecatalog* table every week. Hence, there will be very few new sources per day. During the 10 years operation of GWAC, the HCHDLP performance should be stable.

Figure 7 shows the running time of HCHDLP and increase of the light curve data size per day as a function of time over a period of nineteen days. Each point in the figure represents the time taken by the pipeline to produce the light curve data for the sources ($2400 \times 175600 = 0.42 billion, \approx 79GB$)

observed in each night. 19 days of observations resulted in a target table of 8 billion rows (1.5 TB) and a light curve associatedsource table of 12.7 billion rows (318 GB). The scripts we used to arrive at the results of this section are available on Github. [7]

We argue that the HCHDLP performance is acceptable in the context of GWAC based on our stress tests. On one hand, the execution time roughly goes linearly with the increase of light curve data per day, which is implied by the fact that both execution time and data volume increase with date in parallel (see Figure 7). On the other hand, the maximum execution time to process one image is derived to be less than 7 seconds from Figure 7, which is less than the GWAC cadence of 15 seconds.

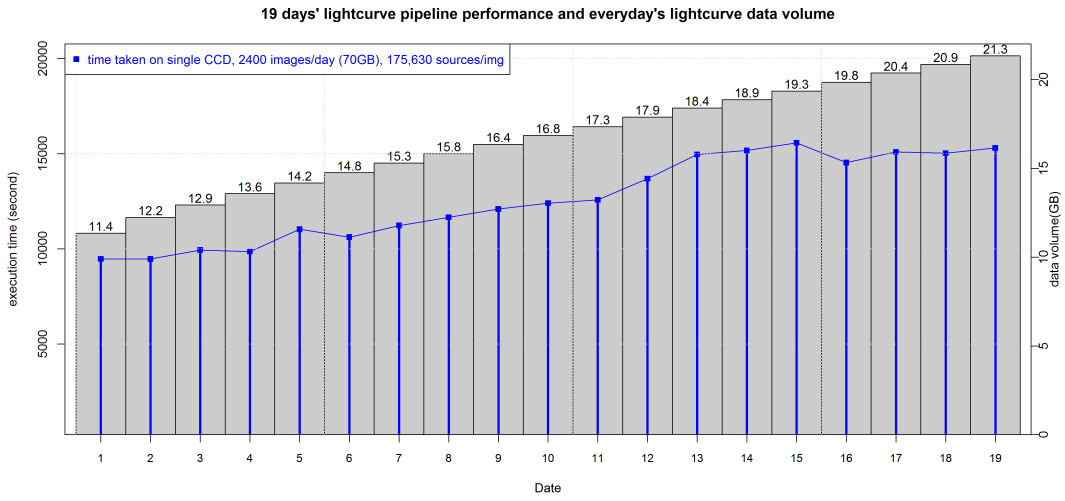

Fig. 7.— Light curve pipeline performance and the light curve data generated everyday over a period of nineteen days, the execution time of the stress test goes up linearly with everyday's increasing data size. For the first day's data volume that is equivalent to GWAC daily data volume, the average time to process one image is 3.95 seconds, which is much faster than the speed of GWAC high-cadence image generation of 15 seconds.

4.4. Performance of Shared-Nothing Distributed Database

4.4.1. Light Curve Retrieval on Partitioned Tables

If the data volume of the light curve table *associatedsource* increases continuously night by night, queries on the huge table would be very slow. However, astronomers are usually interested

[7] https://github.com/wan-meng/gwac_pipeline, https://github.com/wan-meng/lightcurve-chunksize, https://github.com/wan-meng/concurrency

in data within certain time slices, so in many cases, the results of a query can be achieved by accessing a temporal subset of partitions, rather than the entire table, a technique that is called **Partition Pruning**. Partition pruning splits a large table into smaller, individual tables, so queries that access only a fraction of the data can run faster because there is less data to scan. Partition pruning dramatically reduces the amount of data retrieved from disk and shortens processing time, thus improving query performance and optimizing resource utilization (Herodotou et al. 2011).

In terms of the number of partitions, a larger number of smaller partitions provides finer granularity and causes less I/O but also increases management overhead. The more partitions we generate, the more partitions we have to deal with. In consideration of these trade-offs, we need to test which chunk size can provide the best response time for our testing system. In the test of determining the optimal chunk size for the *associatedsource* table, it is horizontally partitioned on ranges of ROW_NUMBER. Four partitioning ranges, i.e., chunk sizes, are tested to find which provides the best response time. Each chunk is created by accumulating the ROW_NUMBER of the *associatedsource* table to reach the size of 6×10^9, 12×10^9, 24×10^9 and 48×10^9 rows respectively. The SQL extract below is used to generate the timing information of figure 8.

For each chunk size, the following queries of light curve retrieval are launched to search for all time series of flux measurements for a source of uniqueid=1 from all partitioning chunks, so the WHERE clause does not specify which partitions are relevant for the query.

```
SELECT i.jd, t1.flux
FROM (SELECT flux,imageid FROM
             (SELECT targetid
                FROM associatedsourceX
               WHERE uniqueid=1)
              t0, targetX t
         WHERE t0.targetid=t.id
      ) t1,
      image i
WHERE t1.imageid=i.imageid;
```

The response times are measured for both hot and cold runs. The hot runs refer to all the needed data is already loaded into main memory, while the cold one is that data need to be loaded into memory before the queries start running. MonetDB is a main-memory database, the entire main-memory of which can be viewed as cache for disk IO access (Manegold et al. 2002). MonetDB aggressively uses as much memory as available, as many cores as possible in parallel without many tuning knobs, and tries to avoid going to a slow disk, so caching effects have a significant impact on performance.

For chunk sizes of 6B to 48B, the tested data size ranges from 710GB for a period of 8 days to 3TB for a period of 26 days. The dataset includes the light curve table *associatedsource* and

the *target* table loaded into MonetDB beforehand. Then, we ran the above light curve retrieval query using uniqueid=1 and measured the (wall clock) time it took to produce the result. In cold runs, the database is stopped and all file system caches are emptied. In hot runs, we start up the database, and run the query two times to warm up caches. For both cold and hot runs, every query was ran five times to control for random fluctuations in system IO and background activity.

Figure 8 shows the **normalized** response times of returning a row as a function of partitioning chunk size. It is obvious that the response time of the hot runs is much shorter than that of the cold runs, as expected. As we can see, for all cold runs, response times are almost unvaried. For hot runs and chunk size between 6B and 24B, MonetDB maps all needed data into memory via the disk cache to greatly increase the performance by avoiding disk IO. When the chunk size grows to 48B, related columns of the *associatedsource* table and *target* table touched by this query add up to 1035GB, $\sim$ 4 times the RAM capacity of 256GB. Therefore, the kernel is swapping data out to free up some memory, which causes severe performance degradation. So, our tests suggest that the chunk size of 24B is an optimal trade-off for our testing system. It is a good balance between partitioning size of tables and performance.

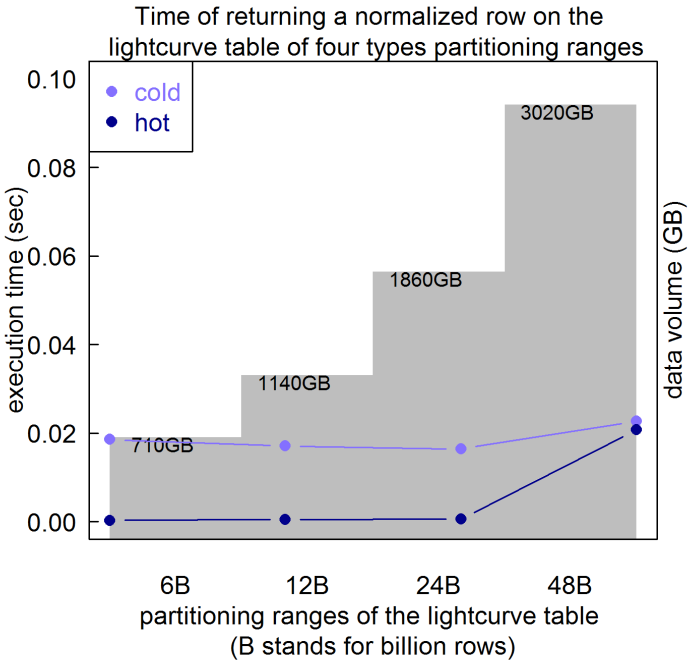

Fig. 8.— Performance of light curve retrieval over different partitioning chunk sizes.

4.4.2. Scalable Concurrency

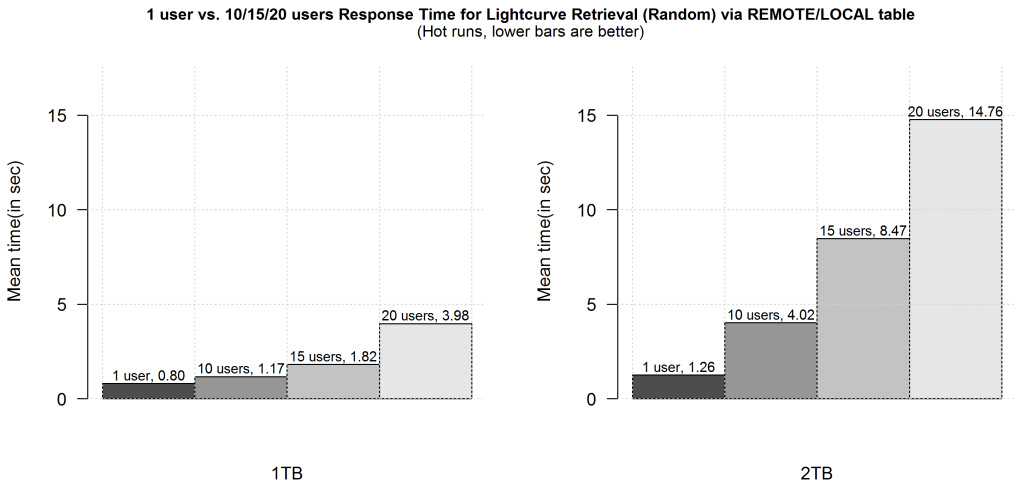

Fig. 9.— Single user vs. 10 vs. 20 users Response Time on 1TB/2TB datasets

Good concurrency is highly desirable. A database with good concurrency allows a large number of users to access a database without any noticeable impact on performance (Bernstein and Goodman 1981). Appropriate concurrent query execution facilitates improved resource utilization and aggregate throughput, while too much concurrency makes it a challenge to overall query performance (Duggan et al. 2011).

To provide both the fastest speed and highest concurrency of our database, the goal of this experiment is to find, at which degree of concurrency the light curve retrieval query is fastest. In the experiment, we increase the number of concurrent users and measure the response time of random light curve retrievals on the distributed database, which is accessed either via the REMOTE table on the master node or via the local table on the worker nodes. We fire up (1/10/15/20) parallel queries at a time to represent multiple-users accessing the database with data volumes of 1TB and 2TB. The test result is shown in Figure 9. In this figure, it is obvious that the response time of concurrent queries for 1TB is much shorter than that for 2 TB. In the 1TB case, the average response time of a single query is 0.8/0.117/0.121/0.199 for a concurrency of 1/10/15/20. So, if we fire up 10/15/20 queries at a time every 10 seconds, 60 queries will take a total of 7.02/7.26/11.94 seconds instead of 48 seconds for 1 user. In one minute, we will get throughputs of 60/90/120 queries/minute. This test shows that scheduling appropriate concurrency can boost performance and throughput by efficient use of hardware resources. If we emphasize response time performance over throughput as a scientific program, there should be no more than 15 concurrent users. But when more throughput is the evaluation criterion, 20 users is a better choice.

5. Conclusion

We develop a real time light curve processing pipeline HCHDLP for the GWAC project. It is based on the column store database engine, MonetDB, and has been optimized for the large amounts of data of GWAC. From outside and inside the database engine, MonetDB has been improved to reduce the time consumed in the source association procedure. Outside the database engine, a sorted version of the inner relation ordered by zone column is created before the join phase, and a SQL *WITH* clause is adopted to avoid multiple recomputations by MonetDB. For inside the database engine, the RANGE JOIN query is implemented through quick binary search and compressed column imprints. Furthermore, in building light curves of one_to_many match type, HCHDLP drops uniqueness check during running and replaces DELETE with INSERT (i.e., insert the old relationships to a small auxiliary table).

As a result of optimization, our HCHDLP can process one catalogue in 3.95 seconds on average, significantly shorter than the image cadence of 15 seconds of GWAC. Our tests show that the source association is sped up by a factor of 220x relative to that before optimization. According to theoretical analysis, the time complexity is reduced from $O(|r|\cdot|l|)$ to $O(|r|\cdot\log|l|)$ (binary search) or $O(C\cdot|r|\cdot|l|)$ (imprints, C $\ll$1). We have also made comparisons between MonetDB and PostgreSQL in performance of source association queries and find that the former takes 1.1s on two tables with 175,597 rows $\times$ 175,540 rows, 3.54$\times$ faster than the row-store disk-based database PostgreSQL.

We also built a shared-nothing distributed database to manage long-term light curves using a two-level time partitioning strategy via the MERGE TABLE and REMOTE TABLE technology of MonetDB. The optimal partitioning chunk size should give both short response time and fewer partitions, and is determined through tests. We find that the response time of MonetDB scales linearly with the number of users. In this scalable solution, both short and long-running queries on large data sets are available, which provides guidance for a solution to GWAC in the management of massive data. Finally, estimating the scalability of HCHDLP performance over the full survey duration with realistic new source counts is key for a long term astronomical project like GWAC. The upper limit of real source counts of GWAC is less than 1000 per day, taking into account optical transients or other moving objects, all noises and false positives. In practice, we will clean all of above kinds of sources out of the *uniquecatalog* table every week. Hence, new sources per day is very few and the HCHDLP performance over the survey duration should be stable in the long term.

In our future work, the HCHDLP may be further optimized in processing speed and in match criteria. The GWAC real-time image processing is required to be completed in 15 seconds for one image, but the current implementation of pipeline 0 (see figure 2) takes almost 10 seconds and HCHDLP takes 5.8 seconds (including both data loading and the cross-match). We note that pipeline 0 can be sped up through GPU implementation and advanced algorithms based on our current optimization are available for source association of HCHDLP. Moreover, the match criteria of source association will take into account adding a flux factor and using variable tolerance radius

with brightness of objects to increase accuracy of cross-matching. Our designed shared-nothing distributed database can provide one preliminary solution to managing light curves in the long term. It is certain that with technology advancing, there are other possibly better solutions that may overcome the data challenge. Additionally, there are some optimization options to reduce light curve data volume, such as aggregate source characterization or reducing time resolution of the data.

6. Acknowledgement

The authors would like to thank the anonymous referee for providing constructive comments which substantially helped improving the quality of the paper. We would also like to thank Prof. Yanxia Zhang, Dr. James E. Wicker, Dr. Xianmin Meng, Dr Xiaomeng Lu, Mr. Yang Xu, Dr. Huali Li, Dr. Qichen Feng and Dr. Lingjun Wang for their helpful discussions and modifications. This work was supported by a grant from the Sino-Dutch Scholarship Programme of the China Scholarship Council. It was also supported by the National Basic Research Program of China (973-program, Grant No. 2014CB845800), and the National Natural Science Foundation of China (Grant Nos. U1331202, 11533003 and U1431108).

REFERENCES

Daniel Abadi, Peter Boncz, Stavros Harizopoulos, Stratos Idreos, Samuel Madden, et al. *The design and implementation of modern column-oriented database systems*. Now, 2013.

Jay Banyer, Tara Murphy, et al. Vast-a real-time pipeline for detecting radio transients and variables on the australian ska pathfinder (askap) telescope. *arXiv preprint arXiv:1201.3130*, 2012.

Jacek Becla and DL Wang. Enabling scalable data analytics for lsst and beyond. In *Exascale Radio Astronomy*, volume 1, page 30303, 2014.

Jacek Becla, Kian-Tat Lim, and Daniel Liwei Wang. Report from the 3rd workshop on extremely large databases. *Data Science Journal*, 8:MR1–MR16, 2010.

Jacek Becla, Daniel Wang, Serge Monkewitz, KT Lim, Douglas Smith, and Bill Chickering. Lsst database design. 2013. URL `http://ls.st/LDM-135`.

Philip A Bernstein and Nathan Goodman. Concurrency control in distributed database systems. *ACM Computing Surveys (CSUR)*, 13(2):185–221, 1981.

Peter A Boncz, Stefan Manegold, Martin L Kersten, et al. Database architecture optimized for the new bottleneck: Memory access. In *VLDB*, volume 99, pages 54–65, 1999.

William S Burgett. Ps2: managing the next step in the pan-starrs wide field survey system. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 84490T–84490T. International Society for Optics and Photonics, 2012.

B Cordier, J Wei, J-L Atteia, S Basa, A Claret, F Daigne, J Deng, Y Dong, O Godet, A Goldwurm, et al. The svom gamma-ray burst mission. *arXiv preprint arXiv:1512.03323*, 2015.

Rachel P Dudik, Margaret E Jordan, Bryan N Dorland, Daniel Veillette, Augustyn Waczynski, Benjamin F Lane, Markus Loose, Emily Kan, James Waterman, Chris Rollins, et al. Interpixel crosstalk in teledyne imaging sensors h4rg-10 detectors. *Applied optics*, 51(15):2877–2887, 2012.

Jennie Duggan, Ugur Cetintemel, Olga Papaemmanouil, and Eli Upfal. Performance prediction for concurrent database workloads. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 337–348. ACM, 2011.

Jim Gray, Alexander S Szalay, Aniruddha R Thakar, Gyorgy Fekete, William O'Mullane, Maria A Nieto-Santisteban, Gerd Heber, and Arnold H Rots. There goes the neighborhood: Relational algebra for spatial data search. *arXiv preprint cs/0408031*, 2004.

Jim Gray, David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt, and Gerd Heber. Scientific data management in the coming decade. *SIGMOD Rec.*, 34(4):34–41, December 2005. ISSN 0163-5808. doi: 10.1145/1107499.1107503. URL `http://doi.acm.org/10.1145/1107499.1107503`.

Jim Gray, María A Nieto-Santisteban, and Alexander S Szalay. The zones algorithm for finding points-near-a-point or cross-matching spatial datasets. *arXiv preprint cs/0701171*, 2007.

Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized search trees for database systems. In *IN PROC. 21 ST INTERNATIONAL CONFERENCE ON VLDB*, pages 562–573, 1995.

Herodotos Herodotou, Nedyalko Borisov, and Shivnath Babu. Query optimization techniques for partitioned tables. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 49–60. ACM, 2011.

Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Database cracking. In *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 68–78, 2007.

Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, Sjoerd Mullender, and Martin Kersten. Monetdb: Two decades of research in column-oriented database architectures. *Data Engineering*, page 40, 2012.

Zeljko Ivezic, JA Tyson, B Abel, E Acosta, R Allsman, Y AlSayyad, SF Anderson, J Andrew, R Angel, G Angeli, et al. Lsst: from science drivers to reference design and anticipated data products. *arXiv preprint arXiv:0805.2366*, 2008.

Mario Jurić, Jeffrey Kantor, KT Lim, Robert H Lupton, Gregory Dubois-Felsmann, Tim Jenness, Tim S Axelrod, Jovan Aleksić, Roberta A Allsman, Yusra AlSayyad, et al. The lsst data management system. *arXiv preprint arXiv:1512.07914*, 2015.

Nicholas Kaiser, Herve Aussel, Barry E Burke, Hans Boesgaard, Ken Chambers, Mark R Chun, James N Heasley, Klaus-Werner Hodapp, Bobby Hunt, Robert Jedicke, et al. Pan-starrs: a large synoptic survey telescope array. In *Astronomical Telescopes and Instrumentation*, pages 154–164. International Society for Optics and Photonics, 2002.

S Koposov and O Bartunov. Jul. 2006. q3c, quad tree cube–the new sky-indexing concept for huge astronomical catalogues and its realization for main astronomical queries (cone search and xmatch) in open source database postgresql. In *Astronomical Data Analysis Software and Systems XV*, volume 351, page 735.

Marcel Kornacker. High-performance extensible indexing. In *VLDB*, volume 99, pages 699–708, 1999.

Stefan Manegold, Peter Boncz, and Martin L Kersten. Generic database cost models for hierarchical memory systems. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 191–202. VLDB Endowment, 2002.

Stefan Manegold, Martin L Kersten, and Peter Boncz. Database architecture evolution: Mammals flourished long before dinosaurs became extinct. *Proceedings of the VLDB Endowment*, 2 (2):1648–1653, 2009.

MonetDB. binarybulkload. Website, 3 2016. URL `https://www.monetdb.org/Documentation/Cookbooks/SQLrecipes/BinaryBulkLoad`.

Tara Murphy, Shami Chatterjee, David L Kaplan, Jay Banyer, Martin E Bell, Hayley E Bignall, Geoffrey C Bower, Robert A Cameron, David M Coward, James M Cordes, et al. Vast: an askap survey for variables and slow transients. *Publications of the Astronomical Society of Australia*, 30:e006, 2013.

Ray P Norris. Data challenges for next-generation radio telescopes. In *e-Science Workshops, 2010 Sixth IEEE International Conference on*, pages 21–24. IEEE, 2010.

B. Scheers. Database techniques within lofar's transients key project. In *Astronomical Data Analysis Software and Systems XVIII*, volume 411, page 143, 2009.

B. Scheers, F. Groffen, TKP Team, et al. Towards dynamic catalogues. In *Astronomical Data Analysis Software and Systems XXI*, volume 461, page 757, 2012.

Bart Scheers. *Transient and variable radio sources in the LOFAR sky.* PhD thesis, PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 2011.

Lefteris Sidirourgos and Martin Kersten. Column imprints: a secondary index structure. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 893–904. ACM, 2013.

Yogesh Simmhan, Catharine van Ingen, Jim Heasley, and Alex Szalay. Stargazing through a digital veil: managing a large scale sky survey using distributed databases on hpc clusters. In *Proceedings of the first annual workshop on High performance computing meets databases*, pages 33–36. ACM, 2011.

John D Swinbank, Tim D Staley, Gijs J Molenaar, Evert Rol, Antonia Rowlinson, Bart Scheers, Hanno Spreeuw, Martin E Bell, Jess W Broderick, Dario Carbone, et al. The lofar transients pipeline. *Astronomy and Computing*, 11:25–48, 2015.

MP Van Haarlem, MW Wise, AW Gunst, G Heald, JP McKean, JWT Hessels, AG De Bruyn, R Nijboer, J Swinbank, R Fallows, et al. Lofar: The low-frequency array. *Astronomy & astrophysics*, 556:A2, 2013.

Daniel L Wang, Serge M Monkewitz, Kian-Tat Lim, and Jacek Becla. Qserv: A distributed shared-nothing database for the lsst catalog. In *State of the Practice Reports*, page 12. ACM, 2011.

Norbert Zacharias, CT Finch, TM Girard, Arne Henden, JL Bartlett, DG Monet, and MI Zacharias. The fourth us naval observatory ccd astrograph catalog (ucac4). *The Astronomical Journal*, 145(2):44, 2013.

This preprint was prepared with the AAS LaTeX macros v5.2.

View publication stats