Publications of the Astronomical Society of the Pacific, 131:054501 (10pp), 2019 May https://doi.org/10.1088/1538-3873/ab024c


# mcatCS: A Highly Efficient Cross-matching Scheme for Multi-band Astronomical Catalogs

Bingyao Li[1], Ce Yu[1,3], Chen Li[1], Xiaoteng Hu[1], Jian Xiao[1], Shanjiang Tang[1], Chenzhou Cui[2], and Dongwei Fan[2]
[1] College of Intelligence and Computing, Tianjin University, Tianjin 300350, People's Republic of China; yuce@tju.edu.cn
[2] National Astronomical Observatories, Chinese Academy of Sciences, Beijing 100101, People's Republic of China
*Received 2018 July 1; accepted 2019 January 28; published 2019 March 19*

## Abstract

Multi-band astronomical catalog cross-matching has always been, and will continue to be, indispensable to astronomy research. However, the archived data volume in different wavebands is extremely huge, which results in the cross-matching process having high computational consumption and slow response. The complexity will also be augmented by the continuous growth of observational data. In this paper, we present mcatCS (**m**ulti-band **cat**alog **C**ross-matching **S**cheme), a distributed cross-matching scheme to efficiently integrate celestial object data from billion-row multi-band astronomical catalogs. It is deployed on a cluster of commodity machines and provides a command-line-based interface to the end user. To allow fast cross-matching, the data in catalogs are reformatted into the Grouped Spatial Index File, which is a specially designed multi-band catalog uniform format. Furthermore, a min-conflicts data layout strategy is utilized to maximize the parallelization of cross-matching. Using real data, archived in the National Astronomical Observatories of China, we verify that mcatCS has good capabilities for performing efficient and reliable cross-matching between billion-row multi-band catalogs, and experimental results show that the query response speed is 38% to 45% greater than that of MongoDB and 21% to 32% greater than that of PostgreSQL with the HEALPix B-tree index. Moreover, although Q3C and H3C—the extension index packages for PostgreSQL—offer faster query response speed for less than 85 million sources, mcatCS proves to be advantageous after sources scale up to 100 million, and achieves a time reduction of 30.3% and 30.7% compared to Q3C and H3C for 200 million sources.

*Key words:* methods: data analysis – techniques: miscellaneous – catalogs – surveys

*Online material:* color figures

## 1. Introduction

With the development of space technology and in-depth scientific research, astronomical observations have expanded to gravitational waves and various bands of electromagnetic waves, including visible light, radio waves, infrared, X-ray, etc. In practical astronomy research, to acquire a comprehensive understanding of celestial objects' properties or to forecast various astronomical phenomena, related data from different wavebands usually need to be aggregated. For instance, Maselli et al. (2015) found new blazars by aggregating recent multi-frequency catalogs. The LIGO–Virgo detector network observed a gravitational-wave signal in 2017 by the identification of transient counterparts across the electromagnetic spectrum in the same location (Abbott et al. 2017). As stated above, astronomical data fusion is of great significance.

A common challenge astronomical data fusion faces is how to find the corresponding data of the same object or area in multiple data sets. A direct method is to compare each item in the data sets, and find out which objects have the same name or coordinates. Unfortunately, astronomical data are archived using different systems, which implies that data sets have various recording formats, naming schemes, and storage modes. The complexity is also augmented by the existence of a thousand or more features of each recorded item. Moreover, the current astronomical observation data volume has experienced continuous growth. For instance, the latest data release of the SDSS, DR13, reached 125 TB in size (SDSS 2015); the Five hundred meter Aperture Spherical Radio Telescope in Guizhou China (XHWebsite 2016) is expected to generate 3 TB of data each day; the Large Synoptic Survey Telescope will hold over 55 PB of data in the next decade (Becla et al. 2008). It is impractical to process every source in all data sets by pairwise comparison. Therefore, an efficient and prompt mechanism is required for high-volume and high-variety astronomical data fusion.

The work area of astronomical data fusion has covered a variety of domains including astronomical images, astronomical catalogs, and some virtual observatories. Specifically, an astronomical catalog is the standard quantitative data generated

[3] Corresponding author.

after calibration and photometry on the original observational data, and has become the most commonly used format in astronomical research. More importantly, the Chinese Virtual Observatory (China-VO) is in urgent need of an efficient catalog fusion tool (Cui et al. 2013, 2017). Therefore, the work in this paper arises from the requirements of China-VO and mainly involves astronomical catalog fusion.

The most important step in catalog fusion is to perform cross-matching among heterogeneous catalogs, which refers to comparing and identifying the same celestial object data in multiple data sets based on the approximate coincidence of the source coordinates. Over the past decade, several methods have existed for cross-matching catalogs on a single node (e.g., Zhao et al. 2009; Wang et al. 2013; Budavari & Lee 2013). However, some research, such as the generation of light curves in time-domain astronomy, needs to process the catalogs generated by the accumulated historical observational data. In this case, the number of catalogs increases linearly with the accumulation of observational data. A single node can hardly store that much data, and will limit the further improvement in performance. A distributed-computing-based approach is a better solution. Thus, in this paper, we propose a faster and command-line-based stand-alone scheme, mcatCS (**m**ulti-band **cat**alog **C**ross-matching **S**cheme), to fuse data from the same celestial object from billion-row multi-band astronomical catalogs, which can be run in a distributed computing environment as well as on the end-user machine.

Considering the large number of catalogs, mcatCS is deployed in a distributed environment consisting of commodity machines, which allows easy appending of new records into existing data and also scales nicely for large data sets. Because of the different recording formats and data attributes of each astronomical catalog generated by multiple observers, inconsistent fields complicate the preliminary cross-match work. Therefore, the Grouped Spatial Index File—a uniform format for multi-band astronomical catalogs—has been designed, and its special structure can reduce the time complexity of the cross-matching. Besides, a min-conflicts data layout strategy is presented to maximize the parallelism of the cross-matching process in the distributed environment. Thus, the cross-matching time is further reduced and we are able to achieve billion-row multi-band astronomical catalog fusion with ease.

The rest of this paper is organized as follows. Section 2 presents the details of data fusion in the field of astronomy and existing work related to the methods of astronomical catalog cross-matching and some data layout strategies. Section 3 focuses on the design of mcatCS, and the evolution of the experiment is described in Section 4. The last section contains our conclusion and discusses future work.

## 2. Related Work

Astronomical data fusion plays an important role in astronomical discovery. The biggest challenge for astronomical data fusion is to provide astronomers with thorough information from multiple data sets in a short period. Several methods exist for large-scale cross-matching to reduce the fusion complexity. Meanwhile, index structure and storage layout are increasingly important factors that affect the efficiency of astronomical data fusion. Therefore, a preliminary evaluation of these methods is needed.

### *2.1. Data Fusion in Astronomy*

In the Big Data era, what is really needed is both access to and analysis of data in order to exploit their value. Therefore, the gathering, fusion, and processing of massive data sets from multiple sources to extract valuable information is desirable, especially in the field of astronomy.

The whole-wavelength astronomy era has given rise to an increase not only in data volume, but also in data attributes. Complete observable parameter space axes include quantities such as the object coordinates, velocities or redshifts, sometimes proper motions, fluxes at a range of wavelengths, surface brightness and image morphological parameters for resolved sources, and variability over a range of timescales (Zhang et al. 2008). The data in the same area are derived from different surveys, projects, or apparatus of different types: unstructured, semi-structured, structured, and mixed. All these characteristics render it difficult to cross-match sources from different wavebands. Therefore, one needs to examine first what kind of features the fusion process is expected to have to be accurate, and identify which data elements are relevant for each data set. Then, from other fields of information fusion, various matching algorithms relying on specific types of data and tasks should be constructed.

The advent of astronomical data fusion technology provides great new opportunities for astronomy research. Cone search is a common query method in astronomy which queries the information related to a cone in a circular region of the sky defined by a sky position and a radius around that position (Williams et al. 2011). Cone search among multi-band astronomical data is one type of astronomical data fusion in a sense, and it is generally the first step in celestial object study, such as supernova discovery or drawing a light curve. Our research group (Li et al. 2017) has proposed a distributed cone search indexing system (DCSIS), which is aimed at finding a certain area of data from massive multi-band astronomical catalogs. DCSIS offers astronomers a comprehensive view of what ocurred in a certain area of the sky at multi-wavelengths, which is of great importance in astronomical research. Apart from this, astronomical data mining and knowledge discovery from astronomical databases have become a hot spot in astronomical research. With such technology, the functions of correlative prediction, classification, clustering, and novelty discovery can be achieved, and data fusion plays a significant role in this process. For example, 14 new variable stars were

discovered by data mining the image collection taken during the 2015 asteroid photometric observations (Papini et al. 2015), and the image collections are exactly the fusion result of time-series images on a single field for the whole night.

### 2.2. Astronomical Catalog Fusion

#### 2.2.1. Cross-matching of Celestial Objects

Cross-matching is the key technology in astronomical catalog fusion. Because of the differences in observation instruments, data acquisition, and calibration methods, the same astronomical object might have slightly different coordinates in different catalogs (Nietosantisteban et al. 2006a). Therefore, a distance threshold based on the calibration errors is the condition for identifying whether two celestial objects are the same. Generally, the calculation rule of angular distance between two objects $d$ is (where two objects $O_1$ and $O_2$ are represented as being at coordinates ($ra_1$, $dec_1$) and ($ra_2$, $dec_2$))

$$d = \arccos(\sin(dec_1)\sin(dec_2) + \cos(dec_1)\cos(dec_2)\cos(|ra_1 - ra_2|)). \quad (1)$$

When the two objects are close together, the angular distance $d$ can be approximately computed as

$$d = \sqrt{((ra_1 - ra_2) \times \cos\delta)^2 + ((dec_1 - dec_2))^2} \quad (2)$$

$$\delta = (dec_1 + dec_2)/2. \quad (3)$$

The distance threshold $\gamma$ used in our paper is defined as

$$\gamma = 3 * \sqrt{r_1^2 + r_2^2} \quad (4)$$

which is the most widely used formula. Here, $r_1$ and $r_2$ are the error radius of the two astronomical catalogs. In this paper, the pairs of celestial objects that satisfy the formula $d(O_1, O_2) \leqslant \gamma$ will be considered as the same object.

#### 2.2.2. Existing Astronomical Catalog Cross-Matching Methods

Since an exhaustive pairwise-based match on each catalog requires many unnecessary comparisons, several partitioning and parallelization methods have been proposed to speed up cross-matching. Zones, an early cross-matching method proposed by Gray et al. (2007), involved the division of the spherical space into zones, which are declination stripes of equal height to perform cross-match in related zones. Nietosantisteban et al. (2006a, 2006b) parallelized zone-based cross-matching computations by distributing the data and workload among clusters of the Relative Database Management System (RDBMS). However, classical indexation of the RDBMS shows very poor performance, especially in the updating phase; the addition of a new catalog can require up to 4.6 million modifications or additions, which becomes dramatically slow (Ochsenbein et al. 2000). On this basis, Wang et al. (2013) proposed two parallel algorithms with the reference table indexed by zones and implemented a zone index on the GPU, and Budavari & Lee (2013); Lee (2013) adapt it on multiple GPUs.

Apart from the zones algorithm, the hierarchical triangular mesh (HTM) proposed by Kunszt et al. (2001) and HEALPix proposed by Górski et al. (2005) are two mainstream spatial indices for cross-matching, which partition the sphere into triangles and diamond-shaped cells, respectively, and each cell has a unique ID from their coordinates and hierarchy. Peng et al. (2014) combined HTM and HEALPix to solve the block-edge problem and simultaneously speed up the cross-matching. Zhao et al. (2009) designed a parallel cross-matching function using HEALPix on a single SQL server and cross-matched two catalogs with 470 million sources and 100 million sources in 32 minutes. Jia et al. (2015) accelerated the cross-matching by adopting HEALPix and performed cross-matching of 1.2 billion sources on CPU–GPU clusters with seven nodes in 10 minutes. Pineau et al. (2011) partitioned the sky using the HEALPix scheme and finished the cross-matching of 1 billion sources on a single computer in 30 minutes. Recently, Fan et al. (2015) employed a Bayesian model to automatically cross-match radio sources, which can determine whether two sources are associated or double-lobed radio galaxies. Jia & Luo (2016) adopted a multi-assignment single-join method for cross-matching on heterogeneous clusters consisting of CPUs and GPUs. Riccio et al. (2017) proposed a new partitioning method that partitioned the sky into cells and dimensions determined by the maximum value assumed by the main dimension of the matching area or by the minimum partition cell-size parameter.

Another sky-indexing method, quad tree cube (Q3C), was proposed by Koposov & Bartunov (2006). The strategy of Q3C is similar to other sky-indexing schemes, but partitioned into a cube. Each face of the cube is a quad-tree structure. Special look-up tables are used to speed up the computations, which make it faster than HTM in the case of high depth of segmentation. The Q3C is mainly used in the PostgreSQL database. More recently, Landais et al. (2013) were largely inspired from Q3C to build the 2D PostgreSQL library HealpiX-tree-C (H3C). This has the same functionality as Q3C, but works with the HEALPix algorithm.

In addition, there are several ready-made tools for cross-matching. SIMBAD (Wenger et al. 2000) provides a multi-source querying function of astronomical catalogs based on cross-identifications. TOPCAT (Taylor 2011) is a widely used and feature-rich tool, including viewing, editing, and analyzing catalog records, as well as a cross-matching function, and usually a few million rows and hundreds of columns can be easily processed. The NASA/IPAC Extragalactic Database (NED; Helou et al. 1990)—an extragalactic database—contains names, positions, bibliographic references, and a variety of other data. It is similar to SIMBAD in function, although mainly for extragalactic objects. VizieR (Ochsenbein et al. 2000), the current mainstream tool, is a database in which most

of the catalogs are managed by a relational database while the larger catalogs containing over 10 million rows are stored as compressed binary files. It offers the functions of querying observation data, cross-identification of small data sets, and cross-matching of self-uploaded and existing data sets. Recently, other web applications have been developed, such as catsHTM (Soumagnac & Ofek 2018) and ARCHES (Motch et al. 2016). However, for most web-based applications, computation of cross-matching consumes much server resource, thus limiting the current number of jobs and users. For example, according to CDS xMatch service documentation (Boch et al. 2014), the total size of uploaded tables is limited to 100 MB for anonymous users, 500 MB for registered users, and all jobs are aborted if the computation time exceeds 100 minutes.

### 2.3. Data Layout Optimization

Striping and de-clustering are two traditional methods for data layout in parallel storage systems. The data are partitioned into separate regions and distributed in independent storage nodes with a view to retrieving requests spanning different nodes in parallel. RAID (Patterson et al. 2002), which used the data striping technique, was originally designed for statically distributed data over disks. Further, numerous efforts (Song et al. 2011; Liu et al. 2017) have been devoted to parallel data layout optimizations using data striping. Several advanced de-clustering methods (Atallah 2003; Altiparmak & Tosun 2014) have also been proposed for better parallelism. However, they are optimized exclusively for range queries. If the blocks of the request are spread across the disks in a balanced way, such a layout strategy is expected to perform well. However, when an arbitrary query which requires data stored on the same disk arrives, the retrieval of this request cannot be performed in parallel. Thus on this basis, placing the correlated data separately was taken into account by Li et al. (2004), Bhadkamkar et al. (2009), and Hsu et al. (2005). Both these strategies attempted to mitigate the aforementioned problem, yet were used primarily in a single disk. Recently, Rush et al. (2017) proposed a layout algorithm that redistributes the correlated data into separate nodes combining graph coloring and bin-packing techniques. They constructed an undirected correlation graph and used the soft coloring algorithm to color adjacent nodes in different colors and the traditional bin-packing algorithm to handle disk capacities. Yasar et al. (2017) designed a distributed data layout technique for graphs; they assigned a rank label to each block and ordered them based on rank, which aims at reducing the I/O cost. In addition, heuristic algorithms, such as the genetic algorithm (e.g., Zhao et al. 2013; Fan et al. 2016), and particle swarm optimization (Wang et al. 2014), have also been applied to data layout problems. Unfortunately, these methods are not suitable for numerous astronomical catalogs. Their huge numbersof iterations and calculations take the time consumption to an unacceptable level in finding the optimal data layout. Thus, a specifically designed data layout strategy is featured in mcatCS in order to achieve better performance in correlated data layout.

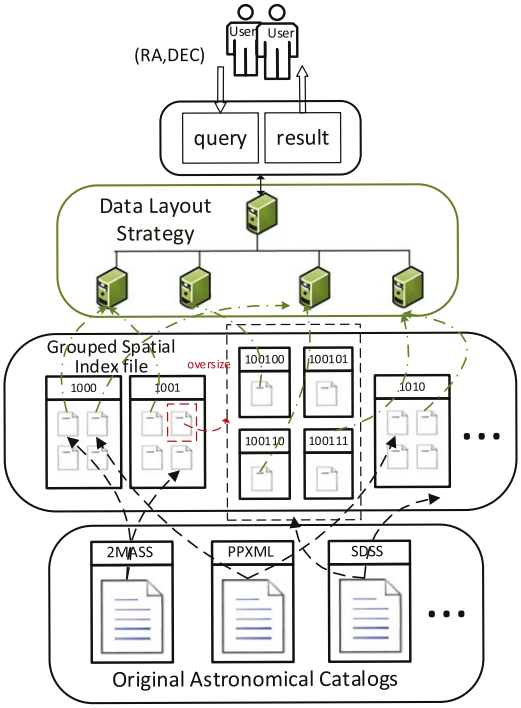


**Figure 1.** Overview of mcatCS.
(A color version of this figure is available in the online journal.)

## 3. Multi-band Astronomical Catalog Cross-Matching Scheme

### 3.1. Problem Definition

This work focuses on cross-matching certain celestial object data from billion-row multi-band astronomical catalogs efficiently; the problem can be expressed formally as follows.

*Problem*. Given a query's two input parameters: right ascension ($T_{ra}$), declination ($T_{dec}$), find all the objects ($S_{ra}$, $S_{dec}$) that satisfy Equation (4). ($S_{ra}$, $S_{dec}$) represent the celestial object's coordinates that matched each other. The query request will be submitted to the server, which will return the target object data from different waveband astronomical catalogs.

The architecture of our cross-match scheme is shown in Figure 1. The original astronomical catalogs in different wavebands would be divided into smaller blocks and recombined to form the grouped spatial index file. Then, according to the data layout strategy, the index file would be distributed over multiple servers. Ultimately, a command-line-based query interface can take the specific coordinates as

inputs, and return all data that matched with the target in multi-band astronomical catalogs as the result.

From the discussion in Section 2, it is clear that cross-matching billions of records in astronomical catalogs is prohibitively time-consuming. To accelerate this process, we proposed a grouped spatial index file and a min-conflicts data layout strategy to match and query the target efficiently.

### 3.2. Design of the Grouped Spatial Index File

Astronomical catalogs generated by different waveband observers have different recording formats and data attributes, and inconsistent fields make the cross-matching complicated. Furthermore, there are millions, even billions, of celestial object records in each original catalog, which makes it unsuitable to match them with each other exhaustively. Considering the "completeness" requirement of astronomical data, we cannot directly divide the original catalogs, and what really need to be aggregated are the catalog records rather than the catalog itself. Thus, grouped spatial index file—a unified file structure for astronomical catalogs from different wavebands—has been designed.

In the first step in the index file build, celestial coordinate information (right ascension and declination) and the celestial line number in each catalog are extracted from the original catalog and recombined into a unified temporary file. Then we divide the whole sky into a mesh on the basis of HEALPix. Thus, each block of the sky has its own HEALPixID, so that each record in the temporary file has a HEALPixID corresponding to the sky block. Next, the records with the same HEALPixID are grouped together and combined into new files. By now, the temporary files have been split into many smaller partition files. We perform this operation on each astronomical catalog from different wavebands.

Since astronomical catalogs come from different bands of telescopes, the number of celestial object records is likely to be different in a sky block with the same HEALPixID. To avoid large partition files increasing the overall cross-matching time, the size of the partition files should be controlled within a certain range. Thus, a strategy similar to adaptive mesh refinement is applied to this target. Our strategy is as follows.

(1) If the size of the partition file exceeds a certain threshold, divide it into four sub-files.

(2) If the total size of the four sub-files is less than the threshold, combine them into a single unit.

(3) Repeat (1) and (2) until all partition file sizes remain unchanged.

Next, we add redundancy data to partition files. The existence of the error radius between different astronomical catalogs can lead to two records belonging to the same celestial object while being divided into different blocks. If we want to find the object in a certain sky block whose center is very close to the block boundary, the target in the other catalog located in the adjoining block might be missed. Therefore, we expand the scope of its blocks by adding four outer borders to ensure the completeness of the query results.

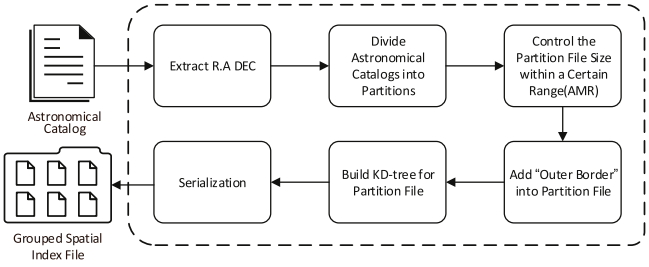


**Figure 2.** Generation process for grouped spatial index file.

After that, we employ all (R.A., decl.) as coordinate points to build a K-dimensional tree (KD-tree) for every partition file. The KD-tree is useful in range searches; it conforms to the requirements of cross-matching and the time complexity will be down to log($N$) ($N$ is the total number of points). However, the KD-tree is a type of data structure just to be stored in memory. Therefore, the last step of generating the index file is serializing the KD-tree into a binary file using Protocol Buffer (Google 2008).

Thus, the final grouped spatial index file is generated. The entire process of generating the index file for a single catalog is shown in Figure 2. When a query arrives, the corresponding HEALPixID of the input target is calculated first and submitted to the servers, which will then load the corresponding index files into the memory to conduct cross-matching.

### 3.3. Min-conflicts Data Layout Strategy

Using the grouped spatial index, cross-matching via large astronomy catalogs has become less of a burden. Nevertheless, with terabytes of original catalogs and millions of query requests per second, there is still an obstacle affecting the response time through only one thread or node. As stated above, it is easy to find that cross-matching can be handled in parallel. Thus, we can use multiple server nodes to load index files and return the result.

In a clustered environment, the communication rate between servers is far more than that of the servers. Optimizing the data layout of each node is a better method for reducing server communication and making full use of the computing nodes. Thus, we present a min-conflicts data layout strategy (MCDL), in which we convert the data layout problem into a quad-tree model and optimize it with an improved Trie tree structure and lazy operation, to minimize the conflict value in each server and scale up the parallelization of the cross-matching process.

Our goal is as follows.

(1) For each task, all the index files involved in a task are distributed on different nodes as much as possible.

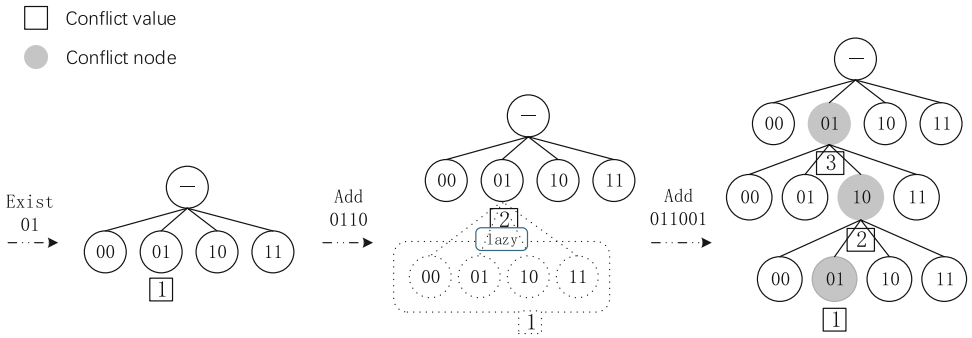


**Figure 3.** Sample graphical representation of the lazy operation process.

(2) For a single node, each node contains the index files with the same sky block ID as little as possible.

We build a quad-tree based on the HEALPix tree structure to organize index files in each server. If an ancestor node and a descendant node exist in the same server simultaneously, we call it a conflict (Figure 3). Therefore, the goal can be summarized as follows, where Avg and Var denote the average and variance conflicts of each server node, respectively:

$$\text{Min}(\text{Avg}(\text{Conflict}(\text{Block}_{id}))$$
$$\text{Min}(\text{Var}(\text{Conflict}(\text{Block}_{id})).$$

The layout procedure of the index file is as follows.

(1) Conduct a depth-first search from root node.

(2) Find the index file that has a conflict with the current node.

(3) Move this index file to the other server with the minimum conflict value.

(4) Continue depth-first search until all files are traversed.

The minimum conflict value of each server is obtained from top to bottom throughout the tree traversed. However, each node's conflict value needs to be calculated repeatedly, which causes the calculations to increase dramatically during the layout phase. Therefore, we present an improved Trie tree structure, taking advantage of public prefixes of strings to minimize unnecessary string comparisons. Meanwhile, lazy operation is introduced to deal with the leaf node value updated.

The Trie tree, also called the prefix tree, is one of the *n*-fork trees. The basic nature of the Trie tree is that the root node stores no characters, and each of the child nodes stores one character; the characters on the path from the root node to one node are connected to form a string corresponding to this node. In our method, each server possesses a Trie tree to count the number and conflict value of the existing index files, as shown in Figure 3. Specifically, the characters on a path from the root node to leaf node are connected to form the index file name-string, and the conflict values are recorded in each node on the path. When a new index file is added to the system, each server's Trie tree is traversed to find the leaf node of the path matched with the index file name-string, and compared with the conflict value of this leaf node's father node. Once it is determined to be placed in one server, the conflict value recorded by each node corresponding to the index file name-string is incremented by one.

As each index file is moved and placed, all the ancestor and child nodes related to the new file need to be updated, which will lead to the problem of repeated update. To reduce the update and traversal times, lazy operation is introduced. This means that when a leaf node's conflict value needs to be updated, we first record this operation on the father node until the next update occurs in this node's child node, then pass the update operation to the child node and change each node's conflict values. A graphical representation of the lazy operation process is shown in Figure 3.

Through MCDL, the grouped spatial index file would be distributed into multiple servers with minimum conflict. When a query arrives, the servers load the corresponding index files separately, and perform the cross-matching in parallel.

All these algorithms, and the main source code of mcatCS, can be found at https://github.com/libingyao/mcatCS.

## 4. Experimental Evaluation

According to the two parts of mcatCS, we first evaluate the overhead of the build time and the various sizes of grouped spatial index files, and compare both with respect to Q3C and H3C. Next, the performance of MCDL is evaluated, including average conflict value and computation time, and compared with the simulated annealing algorithm in terms of these two aspects. Finally, we evaluate the mcatCS query performance by performing cross-matching operations under an increasing number of sources and compare with MongoDB, PostgreSQL with the HEALPix B-tree index, Q3C, and H3C.

All tests in our paper are performed on Ubuntu servers equipped with an Intel i7-4970 CPU (4 cores@3.6 GHz), 16 GB of memory, and two HDDs, one (1 TB) for the Ubuntu operating system, and the other (3 TB) for storing the data set. The data sets used in our experiments are from actual

**Table 1**
Size of the Data Set

| Name of Catalogs | 2MASS | PPMXL | SDSS | USNOB2 |
|---|---|---|---|---|
| Num of records | 470,992,971 | 910,469,430 | 134,269,975 | 305,391,856 |
| Original file size (GB) | 200.2 | 198.2 | 253.4 | 39.1 |

observations archived in the National Astronomical Observatories of China. The details of thedata sets we used are given in Table 1.

### 4.1. Evaluation of the Grouped Spatial Index File

First, we evaluate the build time and the final size of the grouped spatial index files. In the choice of the HEALPix partition level, considering that a high partition level would increase the proportion of redundancy data and a low level would result in a long matching time, we partition the whole sky on level 13 using the HEALPix C facility. This series of experiments is conducted on a server, and the number of sources increases from 1 million to 15 million extracted from 2MASS. The results are shown in Figure 4. We find from the results that, with the increase in original file size, the index file grows at a fixed rate, and only takes a tiny fraction of the original data, which means that the index file will consume only a very small amount of storage space. The index file build time increases approximately linearly as the original file grows. This is due to the process of index file build having a time complexity of O($N$). Considering that it happens only once, so that the time consumption is not too demanding, the results are acceptable for real-life usage.

Then, we compare mcatCS with two sky-indexing extensions for PostgreSQL— Q3C and H3C—in terms of index file size and index file build time. Figure 5 shows a comparison of index file size for mcatCS, Q3C, and H3C. The original number of rows (sources) spans 20 million to 200 million. In this diagram, irrespective of the size of the original file, the index file size is always the same for Q3C and H3C, and grows by 428 MB for each additional 20 million of original data. The reason is that both have the same amount of data and similar index-building algorithm. The index file size of mcatCS also grows by a fixed value, and for every 20 million data elements, the index file size is 6 MB smaller than the other two. In many cases, index performance is improved by trading space for time, that is to say, the index takes up more space to reduce retrieval time. Compared to the other two methods, our method achieves an improvement in performance without increasing the index space, which is an advantage.

As shown in Figure 6, the growth slope of index build time of Q3C and H3C increase more and more obviously as the number of sources grows, while that of mcatCS increases approximately linearly and remains smaller than the other two. In terms of the index build time, mcatCS achieves a time

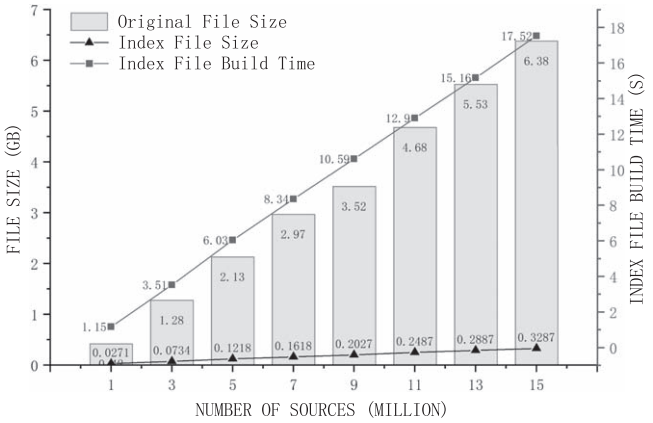


**Figure 4.** Build time and size of the grouped spatial index file.

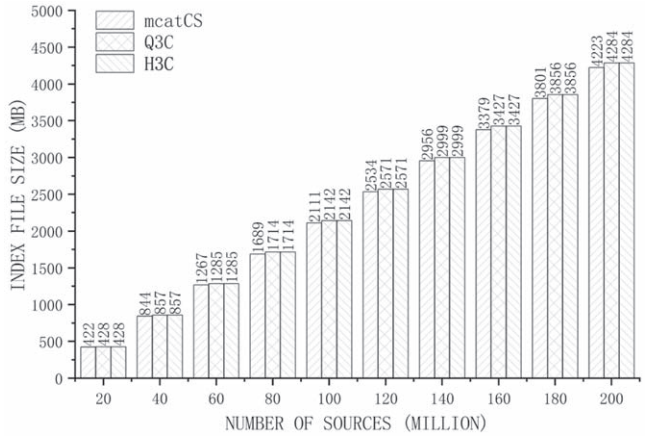


**Figure 5.** Index file size of mcatCS, Q3C, and H3C for different source numbers.

reduction of 27.9% and 21.5% compared to those of Q3C and H3C with 200 million sources.

### 4.2. Min-conflicts Data Layout Performance

We generated 1000 to 4000 index files using the data sets above, and conducted the experiment and its comparison experiments on four servers of equal capacity. To evaluate the performance of the MCDL, the average conflict values of each server and the total computation time of the file placement are measured in the following series of experiments.

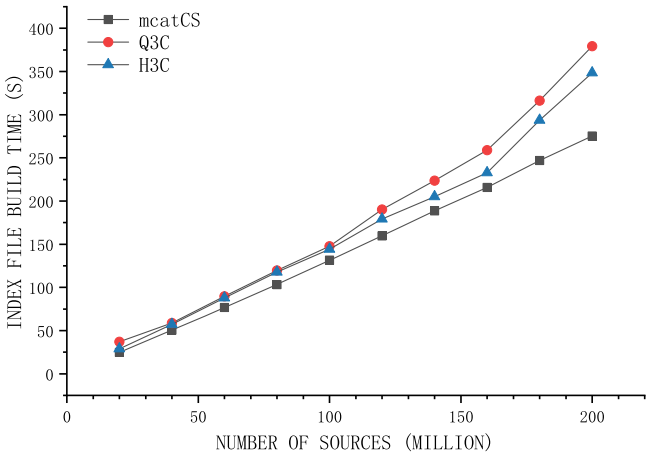


**Figure 6.** Index file build time of mcatCS, Q3C, and H3C for different source numbers.

(A color version of this figure is available in the online journal.)

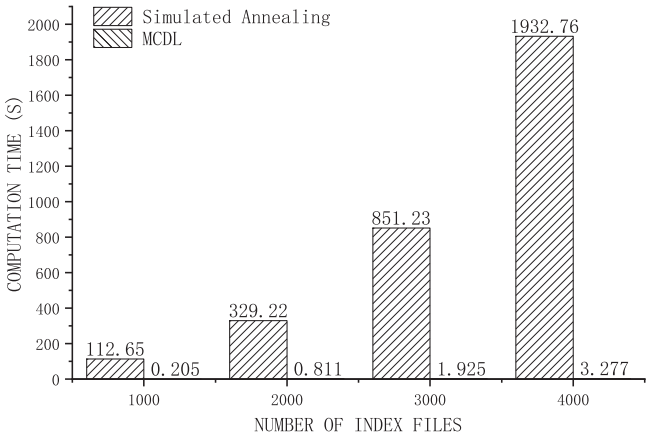


**Figure 7.** Data placement computation time of the two methods.

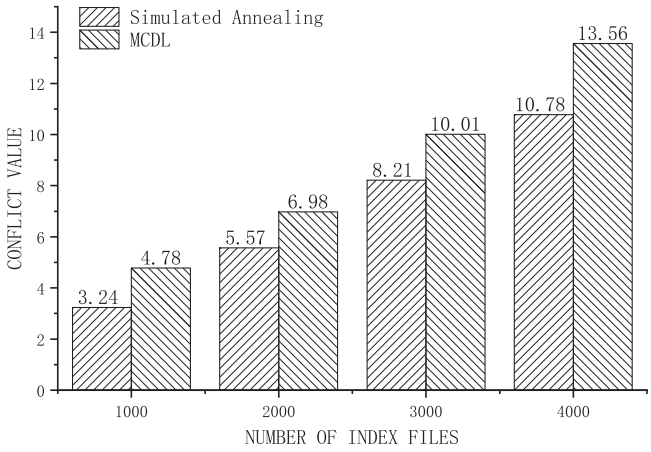


**Figure 8.** Average conflict value of the two methods for different numbers of index files.

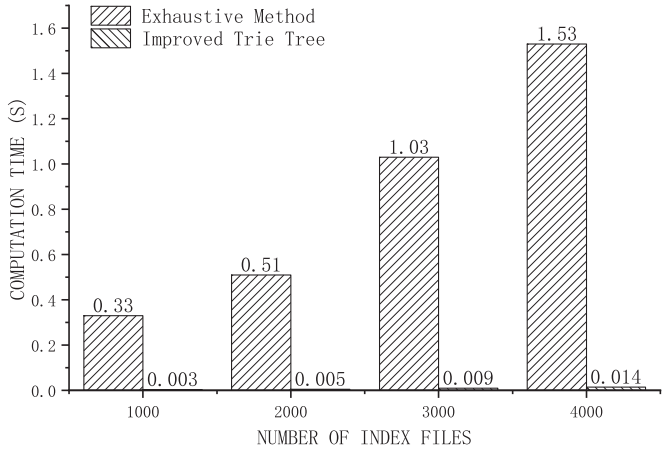


**Figure 9.** Conflict value computation time of the two methods.

Our innovative method is compared with one of the popular heuristic methods, the simulated annealing algorithm, which has strong local search ability compared with other heuristic methods, such as the genetic algorithm. Two pruning optimizations for the traditional iteration of simulated annealing are conducted: if the conflict value of the index file in the target server is larger than the original after the move, the operation is canceled; if the convergence result swings several times between the two results, it is recorded and skipped in the subsequent iterative process. The execution time and average conflict value are used to measure the performance of the above two methods.

The results are shown in Figures 7 and 8, which indicate that our method took only a fraction of the time needed by the simulated annealing method to complete the same task. Although the conflict value has increased, sacrificing some of the results' quality in handling tons of data is acceptable at the cost of 20% reduction in accuracy in exchange for a 60,000% increase in processing speed. In addition, we measured the computation speed of the conflict value of the improved Trie tree structure introduced in Section 3.3. The results in Figure 9 show that the use of our data structure is 100 times faster than the normal exhaustive method. This means that we can add new generated files to the server in a timely manner and flexibly.

### *4.3. mcatCS Query Performance*

In this set of experiments, four servers are used to build a distributed environment for the mcatCS. The bandwidth between each server is approximately 600 Mbps. All the comparison experiments in this series are also implemented in this environment, which demonstrates the performance of the methods fairly. We randomly generate the requests in order to better simulate a real request, which is without a specification at present. We input the same query commands and ensure that all the methods return the same results. The query results are

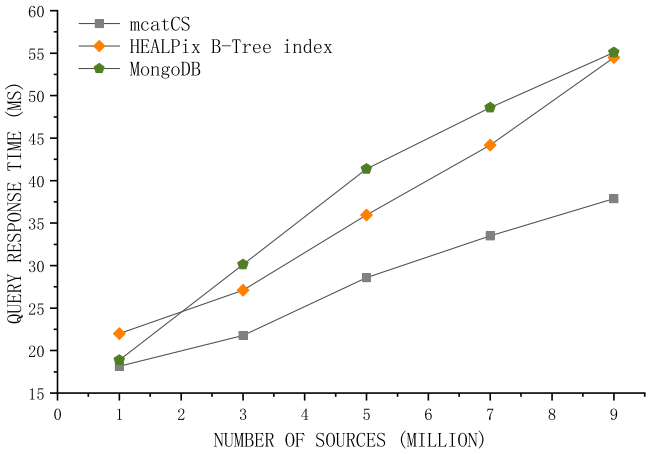


**Figure 10.** Query response time of mcatCS, the HEALPix B-tree index, and MongoDB (B-tree) for different source numbers.
(A color version of this figure is available in the online journal.)

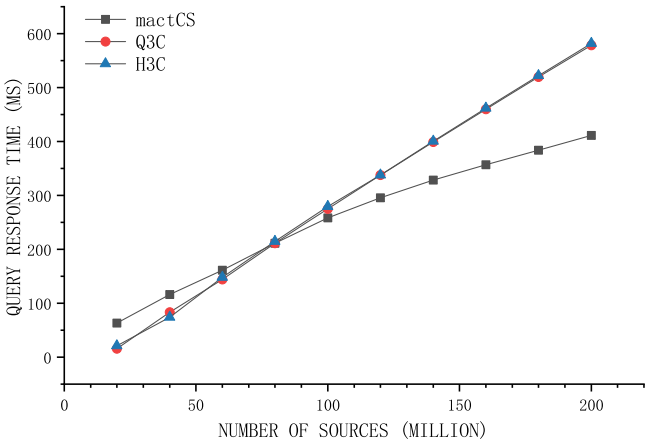


**Figure 11.** Query response time of mcatCS, Q3C, and H3C for different source numbers
(A color version of this figure is available in the online journal.)

compared with the results from VizieR to ensure the credibility of our evaluation. The query response time is the metric of this series of experiments, which represents the time from the submission of the query to the return of the result to the user.

In comparison experiments of the HEALPix B-tree index and MongoDB, the R.A., decl., catalog name, and the number of rows where the data originally existed are extracted and imported into databases. We use HEALPix to map the R.A. and decl. (two-dimensional) to a single HEALPixID, and build a B-tree index on the HEALPixID column. Figure 10 demonstrates the cross-matching efficiency, where the same task is completed using 38%–45% less time than MongoDB and 21%–44% less than the HEALPix B-tree index. We also find that the advantage of mcatCS becomes more obvious with the increase in data volume.

Next, we compare the query response time of mcatCS with Q3C and H3C under larger data sets. The results are shown in Figure 11. As we can see from the line graph, Q3C and H3C outperform our scheme at the beginning while, with the increase in data volume, this advantageous tendency decreases gradually. Our scheme proves faster than the other two after the number of sources is scaled up to 85 million. Moreover, after the number of sources is scaled up to 100 million, the response time of our scheme is reduced by 10.0%–30.3% and 10.3%–30.7% compared to Q3C and H3C, and the time growth slope of our scheme is smaller than those of the other two. All of the above experiments show that mcatCS is suitable for cross-matching a large number of astronomical catalogs.

In addition, KD-tree, B-tree, Q3C, and H3C use tree data structures which theoretically enable data retrieval in O(log $n$) time. The results of mactCS are approximately consistent with the theoretical trend, but the other three methods are approximately linear (we find the same results as in Koposov & Bartunov 2006; Zhong et al. 2015). Thus, we specifically analyzed the likely reasons behind the difference. In the HEALPix B-tree method, the time required to select the sources with the same HEALPixID as the request is O(log $N$) ($N$ is the number of sources). Then, the selected sources are scanned sequentially to find the matched target. The complexity of this process is O($n$) ($n$ is the selected number), and it also takes up a lot of time. In Q3C, the selected sources are retrieved through the quad-tree. However, the quad-tree has difficulty in maintaining a balanced structure. If the spatial objects are distributed unevenly, with the insertion of the new points, the level of the quad-tree will be deepened continuously, and a severely unbalanced quad-tree will be formed. In this case, the depth of each query will be greatly increased, leading to a decline in query efficiency. Moreover, the number of result set increases with the number of sources, which will increase the traversal time of the results, and this may also be the reason for the approximately linear trend. H3C is inspired from Q3C, and follows the same idea. Theoretically, O(log $n$)+A and B*O(log $n$) are both O(log $n$), but A and B cannot be ignored in the actual computation time if these parts take up a certain amount of time.

## 5. Conclusion and Future Work

In this paper, a highly efficient and stand-alone multi-band astronomical catalog cross-match scheme is presented, which reduces the cross-matching time consumption and facilitates the fusion of billion-row multi-band astronomical catalogs. Astronomers can arbitrarily choose a celestial object's coordinates to obtain the final multi-band matching results.

To ensure high-performance, a unified format for multi-band astronomical catalogs—the grouped spatial index file—has been designed, which can reduce the calculation complexity. The min-conflicts data layout strategy achieves splendid performance in terms of time-consumption of correlated data placement, and can add new index files to the server flexibly. All these novel optimizations boost the cross-matching speed,

and experimental results show that mcatCS is 38%–45% and 21%–32% faster than MongoDB and the HEALPix B-tree index, respectively. In addition, we compared the query performance of mcatCS with two extensions of PostgreSQL, namely, Q3C and H3C, for a larger number of sources. Although Q3C and H3C outperform our scheme for less than 85 million sources, the latter turns out to be more advantageous after the number of sources reaches 100 million, and achieves a time reduction of 30.3% and 30.7% compared to those of Q3C and H3C for 200 million sources, affirming that our method has obtained credible results while reducing the time consumption of the cross-matching process.

In addition, our methods can be applied to other applications aside from fusing astronomical catalogs. Generally, any application that requires the integration of massive data or placing correlated data could benefit from it, such as remote sensing and digital maps.

In future research, we will work on reducing the load time of grouped spatial index files and try to deploy the scheme in a geo-distributed environment. The current mcatCS provides a fundamental data fusion index function, which can be used directly by scientific researchers, or it can replace the underlying data management of existing tools. We will provide the graphic interface and more functionality in future work.

This work is supported by the Joint Research Fund in Astronomy (U1731243, U1731125, and U1531111) under cooperative agreement between the National Natural Science Foundation of China (NSFC) and Chinese Academy of Sciences (CAS), the National Natural Science Foundation of China (11573019, 61602336). The authors thank the National Astronomical Observatories of China and Chinese Virtual Observatory for providing the data set used in the experiments.

## References

Abbott, B. P., Abbott, R., Abbott, T. D., et al. 2017, PhRvL, 119, 161101
Altiparmak, N., & Tosun, A. S. 2014, IEEE Transactions on Parallel and Distributed Systems, 23, 538
Atallah, M. J. 2003, Inf. Sci., 157, 21
Becla, J., et al. 2008, DatSJ, 7, 1
Bhadkamkar, M., Guerra, J., Useche, L., et al. 2009, in Proc. 7th Conf. on File and Storage Technologies (San Francisco, CA), 183, https://dl.acm.org/citation.cfm?id=1525908.1525922
Boch, T., Pineau, F. X., & Derriere, S. 2014, CDS xMatch Service Documentation, http://cdsxmatch.u-strasbg.fr/xmatch/doc/
Budavari, T., & Lee, M. A. 2013, Astrophysics Source Code Library, ascl:1303.021
Cui, C., Xue, Y., Li, J., et al. 2013, Bulletin of Chinese Academy of Sciences, 28, 511
Cui, C., He, B., Yu, C., et al. 2017, arXiv:1701.05641
Fan, D., Budav, S. T. R., Norris, P. R., & Hopkins, M. A. 2015, MNRAS, 451, 1299
Fan, W., Peng, J., Zhang, X., & Huang, Z. 2016, LNCS, 10065, 253
Google 2008, Google protocol buffers homepage, https://developers.google.com/protocol-buffers
Gray, J., Nieto-Santisteban, M. A., & Szalay, A. S. 2007, arXiv:cs/0701171
Górski, K. M., Hivon, E., Banday, A. J., et al. 2005, ApJ, 622, 759
Helou, G., Madore, B. F., Schmitz, M., et al. 1990, ASSL, 171, 89
Hsu, W. W., Smith, A. J., & Young, H. C. 2005, ACM Trans. Comput. Syst., 23, 424
Jia, X., & Luo, Q. 2016, in Proceedings of the 28th International Conference on Scientific and Statistical Database Management, 1
Jia, X., Luo, Q., & Fan, D. 2015, in IEEE 21st Int. Conf. on Parallel and Distributed Systems (Washington, DC: IEEE), 617
Koposov, S., & Bartunov, O. 2006, in ASP Conf. Ser. 351 Astronomical Data Analysis Software and Systems XV (San Francisco, CA: ASP), 735
Kunszt, P. Z., Szalay, A. S., & Thakar, A. R. 2001, in Mining the Sky: Proc. of the MPA/ESO/MPE Workshop, ed. J. Banday, S. Zaroubi, & M. Bartelmann (Berlin: Springer), 631
Landais, G., Ochsenbein, F., & Simon, A. 2013, in ASP Conf. Ser. 475 Astronomical Data Analysis Software and Systems XXII (San Francisco, CA: ASP), 227
Lee, M. A. 2013, in Proc. ASP Conf. Ser. 475, Astronomical Data Analysis Software and Systems XXII (San Francisco, CA: ASP), 235
Li, C., Yu, C., Xiao, J., et al. 2017, in Int. Conf. on Algorithms and Architectures for Parallel Processing (Cham: Springer), 239
Li, Z., Chen, Z., Srinivasan, S. M., & Zhou, Y. 2004, in Proc. of the 3rd USENIX Conference on File and Storage Technologies (San Francisco, CA), 173
Liu, Y., Huang, X., Huang, Y., et al. 2017, Concurrency, Pract. Exp., 29, e4039
Maselli, A., Massaro, F., D'Abrusco, R., et al. 2015, Ap&SS, 357, 141
Motch, C., Carrera, F., Genova, F., et al. 2016, in ASP Conf. Ser. 512, Astronomical Data Analysis Software and Systems XXV, ed. N. P. F. Lorente, K. Shortridge, & R. Wayth (San Francisco, CA: ASP), 165
Nietosantisteban, M. A., Thakar, A. R., & Szalay, A. S. 2006a, Astronomy (Baltimore, MD: Johns Hopkins Univ.)
Nietosantisteban, M. A., Thakar, A. R., Szalay, A. S., & Gray, J. 2006b, in ASP Conf. Ser. 351, Astronomical Data Analysis Software and Systems XV, ed. C. Gabriel et al. (San Francisco, CA: ASP), 493
Ochsenbein, F., Bauer, P., & Marcout, J. 2000, A&AS, 143, 23
Papini, R., Franco, L., Marchini, A., & Salvaggio, F. 2015, JAVSO, 43, 207
Patterson, D. A., Gibson, G., & Katz, R. H. 2002, ACM Sigmod Record, 17, 109
Peng, D. U., Ren, J. J., Pan, J. C., & Luo, A. 2014, SCPMA, 57, 577
Pineau, F., Boch, T., & Derriere, S. 2011, in ASP Conf. Ser. 442, Astronomical Data Analysis Software and Systems XX, ed. I. N. Evans et al. (San Francisco, CA: ASP), 85
Riccio, G., Brescia, M., Cavuoti, S., et al. 2017, PASP, 129, 024005
Rush, E. N., Harris, B., Altiparmak, N., & Tosun, A. A. 2017, in IEEE 23rd Int. Conf. on High Performance Computing (Washington, DC: IEEE), 132
SDSS, 2015, DR13, http://www.sdss.org/dr13/data_access/volume
Song, H., Yin, Y., Sun, X. H., et al. 2011, in IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (Washington, DC: IEEE), 414
Soumagnac, M. T., & Ofek, E. O. 2018, PASP, 130, 075002
Taylor, M. 2011, Starlink User Note 253 (http://adsabs.harvard.edu/abs/2013StaUN.253.....T)
Wang, S., Liu, Z., Zheng, Z., et al. 2014, in IEEE 21st Int. Conf. on Parallel and Distributed Systems (Washington, DC: IEEE), 102
Wang, S., Zhao, Y., Luo, Q., Wu, C., & Yang, X. 2013, in IEEE 9th Int. Conf. on E-Science (Washington, DC: IEEE), 326
Wenger, M., Ochsenbein, F., Egret, D., et al. 2000, A&A, 143, 9
Williams, R., Hanisch, R., Szalay, A., & Plante, R. 2011, submitted arXiv:1110.0498
XHWebsite, 2016, Xinhua Insight: Installation Complete on World's Largest Radio Telescope, http://news.xinhuanet.com/english/2016-07/03/c135485389.htm
Yasar, A., Gedik, B., & Ferhatosmanoglu, H. 2017, Distrib. Parallel Databases, 35, 23
Zhang, Y., Zheng, H., & Zhao, Y. 2008, Proc. SPIE, 7019, 701938
Zhao, E. D., Qi, Q., Xiang, X. X., & Chen, Y. 2013, in IEEE 8th Int. Conf. on Computational Intelligence and Security (Washington, DC: IEEE), 146
Zhao, Q., Sun, J., Yu, C., et al. 2009, in IEEE 9th Int. Conf. on Algorithms and Architectures for Parallel Processing (Berlin: Springer-Verlag), 604
Zhong, S., Han, B., Zhang, Y., et al. 2015, AR&T, 12, 510, http://caod.oriprobe.com/articles/46868802/Design_and_Implementation_of_a_Software_Tool_Packa.htm