File-based Efficient Indexing and Fusion of Large-scale Star Catalogs (Postprint)

Authors: Zhang Qiqiang1,2, Fan Dongwei1,2,3, Cui Chenzhou1,2,3

Date: 2023-12-13T00:00:00+00:00

Abstract

Fast retrieval of large-scale astronomical catalogs is fundamental to implementing tasks such as cross-identification, multi-band data fusion, and transient source detection. In particular, wide-field transient source detection requires completing the retrieval and cross-identification of observational results with large-scale catalogs within a single exposure period to discover variable celestial objects. Existing large-scale catalogs typically contain billions of objects. To enable fast retrieval of these catalogs under limited memory constraints, a comprehensive solution is proposed. By employing a HEALPix-based multi-resolution dynamic partitioning algorithm, the catalog can be divided into appropriately sized and uniform files according to the object density in different sky regions. Furthermore, a catalog-specific serialization scheme is designed built upon the open-source serialization component Protocol Buffers, serving as an intermediate storage medium during catalog partitioning and retrieval to maximize retrieval speed. The application of Peano-Hilbert numbering is also explored to replace the original Z-order numbering of HEALPix for traversing catalogs, which improves cache hit rates and enables efficient fusion of large-scale catalogs, thereby facilitating subsequent data utilization and research.

Full Text

Preamble

Vol. 41, No. 3
September 2023

Progress in Astronomy

The Efficient Indexing and Fusion Algorithms for Large-Scale Catalogs Based on Files

ZHANG Qi-qian[12], FAN Dong-wei[123], CUI Chen-zhou[123]
(1. National Astronomical Observatories, Chinese Academy of Sciences, Beijing 100101, China;
2. University of Chinese Academy of Sciences, Beijing 100049, China;
3. National Astronomical Data Center, Beijing 100101, China)

Abstract

Rapid retrieval of large-scale astronomical catalogs forms the foundation for tasks such as cross-matching, multi-band data fusion, and transient source detection. In particular, wide-field transient searches require completing the retrieval and cross-matching of observed results with large-scale catalogs within a single exposure cycle to identify variable objects. Existing large-scale catalogs typically contain billions of celestial objects. To enable fast retrieval under limited memory constraints, this paper proposes a comprehensive solution. By employing a multi-resolution dynamic partitioning algorithm based on HEALPix, catalogs can be divided into appropriately sized and uniform files according to object density across different sky regions. Furthermore, building upon the open-source serialization component Protocol Buffers, we design a dedicated serialization scheme for catalogs as an intermediate storage medium during splitting and retrieval to maximize retrieval speed. We also explore using Peano-Hilbert ordering instead of HEALPix' s native Z-order for catalog traversal, which improves cache hit rates and enables efficient fusion of large-scale catalogs, facilitating subsequent data utilization.

Keywords: catalogs; HEALPix; cross-matching; virtual observatory; astronomical software
Classification Code: P129
Document Code: A

1 Introduction

With the continuous development of astronomical observation facilities, humanity' s ability to explore the universe has steadily improved. Particularly in recent decades, the widespread adoption of computers and charge-coupled devices (CCDs) in astronomy has transitioned human observation from naked-eye and photographic plate methods to the digital information era. Multiple large-scale digital sky survey projects are expected to generate data volumes far exceeding manual processing limits. For example, the 8.4-meter Vera C. Rubin Observatory Legacy Survey of Space and Time (LSST) in Chile plans to conduct a 10-year survey of 18,000 square degrees in the southern hemisphere starting from 2022. Over this decade, each sky region will be visited over 1,000 times, yielding observational data for 40 billion objects and creating an unprecedentedly large time-domain survey database. The Panoramic Survey Telescope and Rapid Response System (Pan-STARRS) employs an array of four 1.8-meter telescopes.

On one hand, these large-scale surveys will produce data volumes reaching ter-

abyte (TB) or even petabyte (PB) scales. For instance, based on its first telescope Pan-STARRS1 (PS1), the Pan-STARRS project has released two data batches (DR1 and DR2). The PS1 DR2 catalog released on January 28, 2019, contains the ObjectThin table with 10.5 billion rows (approximately 5.4 TB) and the StackObjectThin table with 3.47 billion rows (approximately 1.2 TB). Such data requires timely processing and analysis. On the other hand, transient source searches within these surveys require immediate cross-matching with previous observational data to identify variable objects.

Much previous work on catalog retrieval has been implemented using databases. For celestial coordinates as two-dimensional spatial data, direct storage in databases requires complex trigonometric calculations to compute angular distances between coordinates when performing cone searches using Structured Query Language (SQL). A few databases like PostgreSQL have spatial extensions such as PostGIS for spatial data retrieval, but these components are designed for Geographic Information Systems (GIS) and primarily handle complex three-dimensional geometries (points, lines, polygons) on Earth, with limited application in astronomy due to the need for coordinate transformations between astronomical and geographic systems.

A more common approach employs pseudo-two-dimensional spherical indexing algorithms to partition the celestial sphere into sub-regions, assigning each region an ID number. Objects in catalogs are then stored in databases using their region ID as the primary key. During retrieval, the region ID is calculated from coordinates to read the corresponding region' s objects before computing distances. Current spherical indexing algorithms include the Zones Algorithm, Hierarchical Triangular Mesh (HTM), Quad Tree Cube (Q3C), and Hierarchical Equal Area isoLatitude Pixelisation (HEALPix). Berriman et al. tested retrieval speeds of HTM and HEALPix at different levels using the 2MASS All-Sky Point Source Catalog (approximately 470 million objects) and the non-merged Hubble Source Catalog (approximately 383 million objects) across PostgreSQL and SQL Server databases on Solaris, Windows, and Red Hat Linux platforms. Szalay et al. developed SkyServer using Microsoft' s SQL Server database with the HTM algorithm, enabling online retrieval and access to Sloan Digital Sky Survey (SDSS) data—marking the first implementation of large-scale astronomical data management using commercial databases.

An important application of catalog retrieval is cross-matching and fusion of multiple catalogs. Gao Dan implemented retrieval and cross-matching for hundreds of thousands of catalogs using HTM and databases but failed to address source omission near partition boundaries. Du et al. alleviated this issue by using both HEALPix and HTM with databases. Boch et al. developed the online cross-matching platform CDS-Xmatch using KD-Tree and HEALPix, allowing users to upload custom catalogs for cross-matching with platform-provided or previously uploaded catalogs.

Previous work has demonstrated that databases can achieve efficient catalog retrieval. As shown in Figure 1 [Figure 1: see original paper], when using pseudo-

two-dimensional spherical indexing for coordinate-based retrieval, databases store object information and corresponding IDs, retrieving all objects matching the target ID during queries for upper-layer software to compute angular distances and filter results. Depending on the chosen level, thousands of objects may share the same region ID. In this specific scenario, instead of using databases, storing all objects with the same ID in a single file enables retrieval by simply reading the corresponding file, achieving indexing and retrieval functionality.

For databases, region IDs are merely index numbers, but algorithms like HEALPix exhibit relationships between IDs across different levels and regions. When using files as storage media, these characteristics can be exploited for catalog-specific optimizations to improve indexing and retrieval efficiency. Additionally, unlike commercial database software, files offer advantages including no licensing requirements, better universality, no need for pre-installation or configuration, and easier maintenance. These benefits are particularly valuable in remote observatories with limited maintenance capabilities or in low-performance environments like microcontrollers and small detectors where databases cannot run. Therefore, this paper explores using files as the storage medium for catalog retrieval with optimizations to improve efficiency during cross-matching or fusion.

We select HEALPix as the partitioning foundation. Proposed by Górski et al., HEALPix divides the sphere into multiple equal-area sub-regions called "pixels" at predefined levels, each with a unique ID. Higher levels yield more pixels with smaller areas, providing higher resolution. Initially developed for cosmic microwave background analysis, HEALPix is now widely used in astronomical research. Pineau et al. used HEALPix for cross-matching tests. While original HEALPix only supports fixed-resolution partitioning, various multi-resolution solutions have been proposed, including Multi-Order Coverage maps (MOC), Multi-Resolution HEALPix (MRH), and mhealpy. Xu Yunfei used MOC to build MOC-Trees for indexing irregular region catalogs. Martinez-Castellanos et al. released mhealpy, a Python package for multi-resolution HEALPix and visualization supporting two modes—one allowing users to select partitioning levels via custom functions but limited by visualization constraints where each region can only be covered by one level. Catalog splitting, however, faces no such visualization limitations, so we allow multiple levels to cover the same region for uniform catalog partitioning.

Astronomical catalogs are primarily stored and exchanged in Comma-Separated Values (CSV) and Flexible Image Transport System (FITS) formats, plus VOTable and institution-specific text or binary formats. CSV catalogs store data as plain text with comma delimiters, readable and editable with text editors. However, program access requires string-to-floating-point conversions, impacting performance for large catalogs. FITS, defined by the International Astronomical Union, was originally designed for image transmission and later extended to complex data formats like catalogs, storing data in binary

column-wise format requiring third-party libraries for reading/writing.

During retrieval, only coordinates and limited data are needed, with no readability requirements. Since these files are for internal program use only, custom serialization protocols can reduce format conversion overhead for better access performance.

Survey observations typically scan continuous sky regions where data from previous retrievals can benefit subsequent ones. Appropriate caching strategies can reduce file reads, improve data reuse, and enable faster sequential retrieval. Furthermore, when fusing multiple large catalogs requiring cross-matching of numerous objects, selecting optimal traversal order can improve cache hit rates and cross-matching speed.

Addressing efficient file-based catalog retrieval, this paper proposes a comprehensive solution. Section 2 investigates splitting large-scale catalogs into appropriate and uniform subsets under memory constraints. Section 3 studies efficient retrieval of split catalogs. Section 4 explores using open-source serialization libraries for efficient catalog access. Section 5 discusses methods for cross-matching and fusing multiple large catalogs. Section 6 presents experimental tests of the proposed methods. Section 7 concludes.

2 Catalog Indexing and Splitting Strategy

Searching for Y specific objects in a catalog containing X objects without preprocessing requires $X \times Y$ retrievals in the worst case. An improved approach uses multi-dimensional data structures like KD-Trees to accelerate retrieval. However, constrained by computer architecture, these structures must reside in memory. For multi-terabyte catalogs, memory cannot hold entire datasets, and reading catalogs from disk consumes substantial time. Targeted optimization is needed to accelerate searches.

Catalogs support two fundamental coordinate-based operations:
1) Nearest neighbor search: given coordinates, find the closest object in the catalog
2) Cone search: given coordinates and a distance threshold, retrieve all objects within that radius

Both operations only require searching nearby regions, not reading the entire catalog. Therefore, before retrieval, catalogs can be split by coordinates, placing neighboring objects in the same file with unique IDs. During retrieval, the target region's ID is calculated from query coordinates to read and search only relevant files, avoiding full catalog reads and dramatically improving efficiency.

Split subsets should be appropriately sized and uniform. If single files contain too many objects, read and indexing times during retrieval become excessive. If too few, each retrieval loads numerous small files, and random I/O for small files is far slower than sequential I/O. Retrieval across many files may degrade to sequential scanning, reducing efficiency.

Thus, upper and lower limits must be imposed on object counts in split files to maximize retrieval efficiency.

2.1 Multi-Resolution HEALPix Catalog Splitting Algorithm

Celestial position coordinates, being essential information, serve as the primary splitting criterion. However, linear partitioning by right ascension and declination yields unequal areas, preventing uniform division. We select HEALPix for its equal-area sphere partitioning capability.

Although HEALPix partitions the sphere equally, celestial object distribution is often non-uniform. For example, in catalogs dominated by Galactic stars, density toward the Galactic center exceeds that toward the anti-center or perpendicular to the Galactic plane. Original HEALPix uses a fixed level, causing large object count variations across regions and resulting file size imbalances that affect processing efficiency. To ensure uniform splitting, we propose a multi-level HEALPix coexistence algorithm that dynamically selects appropriate splitting levels based on regional object density.

HEALPix first divides the sphere into 12 equal base regions, then recursively partitions each into 4^k sub-regions at level k, yielding 12×4^k equal-area pixels. Larger k values provide finer resolution. Figure 2 [Figure 2: see original paper] shows partitions for $k = \{0, 1, 2\}$.

HEALPix supports RING and NESTED numbering modes. In RING mode, regions at the same latitude form rings numbered sequentially from north to south. In NESTED mode, a region at level i with ID $npix_i$ divides into four level $i+1$ regions with IDs $[4 \times npix_i, 4 \times npix_i + 3]$. This property suits quadtree implementation perfectly. Therefore, we use NESTED mode with quadtrees to index across different levels.

A quadtree is a hierarchical data structure based on recursive spatial decomposition. The root represents the entire region, dividing into four sub-regions represented by child nodes. This process recurses until reaching desired resolution. Figure 3 [Figure 3: see original paper] illustrates the mapping between quadtrees and HEALPix levels.

Since HEALPix regions at all levels derive from the initial 12 base regions, we construct 12 quadtrees, each root representing one of the 12 level-0 regions. Nodes at tree level i correspond one-to-one with HEALPix regions at level i. Each non-leaf node has four children pointing to four sub-regions, with child $npix$ IDs sharing the same binary prefix as their parent. This mapping enables using tree operations for ordered sky region access.

Because HEALPix numbering starts from 0 at each level, ID conflicts occur between levels. This two-dimensional numbering affects retrieval efficiency. We adopt the NUNIQ encoding from MOC to assign unique IDs across levels. For a region at level k with $npix$ ID, the UNIQ ID is:

$$\text{NUNIQ} = 4 \times 4^k + \text{npix} \quad (0 \leq \text{npix} \leq 3 \times 4^{k+1})$$

The inverse operations are:

$$k = \left\lfloor \frac{\log_2(\text{NUNIQ}/4)}{2} \right\rfloor$$

$$\text{npix} = \text{NUNIQ} - 4^{k+1}$$

Since large catalog size N often far exceeds available memory M, programs cannot read entire catalogs into memory. As shown in Figure 4 [Figure 4: see original paper], catalogs must be pre-split into $\lceil N/M \rceil$ local subsets, each smaller than M. Programs can then read one subset at a time for splitting, clear memory, and proceed to the next. However, directly merging locally split results does not equal splitting the original catalog—quadtree merging is required. Therefore, our splitting algorithm has two steps: local splitting and merging of multiple local splits.

2.2 Local Catalog Splitting

First, parameter k_{max} specifies the maximum HEALPix level and quadtree depth. Larger k_{max} increases memory consumption.

Step one constructs 12 quadtrees with k_{max} levels each. Level i ($i \in [0, k_{max}-1]$) contains 4^i nodes, totaling $4^{k_{max}+1} - 4$ nodes across all trees, each with a UNIQ ID corresponding to a sky region.

Each quadtree node contains variable C counting objects in its region, initialized to 0. For each catalog object read, its coordinates calculate HEALPix region IDs at all levels. Corresponding UNIQ IDs increment node C values, indexing objects in the quadtree.

After reading a local catalog subset into memory and indexing it with quadtrees, splitting executes. A threshold T specifies upper or lower bounds for object counts per file under different modes. When traversing a node, its total object count compares against T to determine next steps. During recursive splitting, three cases occur:

1) If both parent and all four child nodes' C values exceed T, recursively traverse the four children

2) If parent' s $C > T$ but some child' s $C < T$, handle as described below

3) If reaching k_{max} level, split objects in the current node and stop traversal

For non-uniform catalogs (e.g., ground-based northern telescopes limited in observing southern sky), root node C values in southern regions may be $< T$—a special case where all root objects are split.

For case (2), three handling methods exist, illustrated in Figure 5 [Figure 5: see original paper]:

1) **"Aggressive" mode**: Continue traversing children with $C > T$; split children with $C < T$ into separate files and stop recursion. This is one of mhealpy' s two splitting schemes, named "aggressive" here based on its characteristics.

2) **"Conservative" mode**: If any of the four children has $C < T$, split the entire parent region into one file and stop traversing children.

3) **"Neutral" mode**: Visit all four children. If a child' s $C > T$, continue traversing and return value r representing split objects in that region. If $C < T$, stop traversal and return C as r. The parent aggregates child returns to compute remaining unsplit objects. If the total exceeds T, split them under the parent' s name and return the total; otherwise return the split count to the upper level. This continues until objects are split or reaching the root. Figure 5 assumes regions $4 \times p + 1$ and $4 \times p + 2$ are fully split.

Table 1 compares the advantages and disadvantages of these methods. The comparison shows our proposed "neutral" splitting theoretically produces the most balanced files, making it the preferred method.

After splitting completes, memory is cleared. If the catalog is not fully read, the process repeats until all data is processed, completing local splitting.

2.3 Merging Locally Split Catalogs

After $\lceil N/M \rceil$ local splits of an unsorted catalog, we obtain $\lceil N/M \rceil$ file sets, each containing an uncertain number of files, as shown in Figure 4 [Figure 4: see original paper]. While objects within each block are partitioned by region, blocks remain independent. Therefore, merging blocks is required to obtain the complete splitting result for the original catalog.

We reconstruct the 12 corresponding quadtrees. Traversing each tree depth-first from its root, if a locally split catalog contains a file for a given node, it is read into memory to update the node' s C value. We then recursively traverse the node' s four children, aggregating their return values (only in "neutral" mode).

After traversing all four children and returning to the current node, all its descendant nodes have been visited—all objects in that region have either been read into memory or split into final files during child traversal. Similar to local splitting, the splitting decision is made based on mode, threshold T, and remaining object count. "Neutral" mode also returns values to parent nodes. The resulting files represent the merged catalog. This process continues until all local catalogs are merged.

Figure 6 [Figure 6: see original paper] shows unsplit and dynamically split

catalogs in "aggressive" mode. Dense regions are split more finely than sparse regions, maintaining balanced object counts per region.

3 Retrieval of Split Catalogs

After splitting, each catalog file has a UNIQ ID representing its sky coverage. During retrieval, query regions convert to UNIQ ID lists, and corresponding files are read.

3.1 Region-Based Retrieval

Due to multi-coverage in "neutral" mode, objects in a highest-level region exist only in that node' s file or its nearest upper-level node with a catalog subset. Objects in non-highest-level regions may be distributed across multiple levels. Therefore, both parent and child nodes must be traversed.

To retrieve all objects covering a specific region (e.g., node at level k with HEALPix ID pix), start from node pix and recursively visit parent nodes until finding the first node with a catalog subset file. Read all objects within the target region ID range into memory. If $k < k_{max}$, traverse all child nodes of pix and read their existing subset files to obtain complete results.

For polygonal query regions, import the polygon into MOC to obtain the corresponding UNIQ list. Perform the above region query for each UNIQ and aggregate results.

3.2 Coordinate-Based Retrieval

Given a coordinate and radius threshold, first use HEALPix' s query_{disc} function to compute the list of pixels at the maximum level covered by the circular region. Execute the region query for each pixel, marking read catalogs to avoid duplication.

The angular distance d between two celestial coordinates (α_1, δ_1) and (α_2, δ_2) is given by the Haversine formula:

$$d = 2 \arcsin\left(\sqrt{\sin^2\left(\frac{\delta_1 - \delta_2}{2}\right) + \cos\delta_1 \cos\delta_2 \sin^2\left(\frac{\alpha_1 - \alpha_2}{2}\right)}\right)$$

Calculate distances for objects in memory and add those within the threshold to the result list.

For nearest neighbor searches, the target and its nearest neighbor may not share the same level-k_{max} region. Therefore, adjacent regions must also be read. After reading, compute distances using the Haversine formula to find the closest object.

4 Efficient Data Access

Data in memory is typically stored as objects, structures, arrays, or lists in various regions for CPU access. When writing to files or network transmission, data must be encoded (serialized/marshalled) into byte sequences, with the reverse process called decoding (deserialization/unmarshalling). Saving memory-resident objects as CSV format represents serialization; reading CSV catalogs is deserialization.

During splitting, each object is read twice and written twice. Billions of objects generate massive disk I/O and time consumption. Since coordinates are stored as floating-point numbers in memory, text formats like CSV require parsing each line and converting between floats and strings, creating non-negligible computational overhead. Moreover, splitting only requires coordinates, not other physical parameters.

To maximize performance, we build upon the open-source high-performance serialization component Protocol Buffers (Protobuf) to define a custom serialization format for efficient catalog access.

Protobuf encodes data according to predefined formats for inter-program communication or network transmission, and can write to disk for persistence. Functionally similar to XML and JSON, Protobuf requires pre-defined Interface Description Language (IDL) protocols. Data is stored in binary format per IDL specifications using compression algorithms, avoiding number-string conversions, and supports major languages including C++, Java, Python, and Go.

We define the Protobuf transmission format for catalogs: a "Catalog" item stores catalog information including filename, UNIQ ID, header information, and total row count (Table 2). It contains repeated "CatalogLine" items storing object information including coordinates and region IDs as floats and integers, with other parameters as strings to reduce parsing time (Table 3).

Based on this format, Protobuf automatically generates serialization/deserialization code for different languages. This enables efficient catalog storage as Protobuf files during splitting, merging, and retrieval, and reading files back into memory, achieving high-performance data access.

5 Fusion of Multiple Large-Scale Catalogs

Transient detection often requires simultaneous retrieval across multiple band or epoch catalogs. To accelerate retrieval, multiple catalogs can be pre-merged through cross-matching objects across catalogs and fusing their parameters, enabling single-retrieval access to multi-catalog information.

Due to varying telescope precisions, the same source may have slightly different positions across catalogs. Generally, two objects with angular separation $d < 3\sqrt{r_1^2 + r_2^2}$ are considered the same source, where r_1 and r_2 are error radii. Objects near HEALPix region boundaries may have error radii covering adjacent

regions, requiring neighbor region reads during distance calculations. Appropriate traversal order and caching can improve efficiency.

5.1 In-Memory Catalog Indexing

When splitting catalogs into files, object order follows the read sequence during splitting. Sequential retrieval is inefficient; instead, files can be fully read into memory to build KD-Trees for 2D indexing, improving speed.

KD-Tree is an efficient multidimensional data indexing structure. During construction, it selects different dimensions at each level to compute medians as split nodes, forming a balanced binary tree. Retrieval compares different dimensions level-by-level against tree data for efficient high-dimensional queries.

Based on splitting results, we build KD-Trees per catalog file. If a query region covers multiple files, multiple KD-Trees are constructed in memory and cached for reuse, avoiding repeated reads (Figure 7 [Figure 7: see original paper]). Cache size cannot be infinite; when reaching capacity during multiple concurrent tasks, replacement algorithms like First-In-First-Out (FIFO) or Least Recently Used (LRU) can evict KD-Trees. These algorithms have been extensively studied, and we experimentally investigate their performance differences and optimal cache sizes for catalog cross-matching.

5.2 Catalog Traversal Using Peano-Hilbert Curves

Traversing HEALPix in NESTED order follows a Z-curve pattern in space. While efficient for building quadtrees or octrees from point sets, Z-curves exhibit large jumps at region boundaries, moving to non-adjacent regions. This causes cache invalidation during catalog traversal, reducing efficiency. To overcome this, we replace Z-curves with Peano-Hilbert curves, which renumber HEALPix regions so adjacent IDs correspond to adjacent sky regions. This enables continuous neighbor access during traversal, maximizing cache utilization. Figure 8 [Figure 8: see original paper] compares Z-curve and Peano-Hilbert traversals.

6 Experimental Analysis

Experiments use the LAMOST DR7 catalog on a cloud virtual machine from the National Astronomical Data Center with:
1) CPU: Intel(R) Xeon(R) E5-2690 4 Core @ 2294.686 MHz
2) Memory: 8 GB
3) Disk I/O: 82.5 MB/s
4) OS: Ubuntu 20.04.4 LTS
5) Languages: C++, Python

6.1 Comparison of Splitting Methods

We tested "aggressive," "neutral," and "conservative" dynamic splitting modes against fixed-level splitting with threshold $T = 1000$. Results are shown in

Figure 9 [Figure 9: see original paper] and Table 4 .

The "aggressive" mode produces files with object counts below threshold, mostly concentrated in $[200, 1000]$. The "neutral" mode yields counts primarily in $[1000, 4000]$, with minimal files below threshold due to sparse southern sky coverage, and shows the lowest coefficient of variation, indicating most uniform distribution. The "conservative" mode produces files all > 1000, with some excessively large ($\sim 10^6$ objects) and the highest coefficient of variation. Fixed-level results show similar variation, all higher than "neutral" and "aggressive" but lower than "conservative."

Results confirm that "neutral" splitting produces more uniform files than other modes and fixed-level splitting, establishing a foundation for efficient retrieval.

6.2 Protobuf vs. CSV Read Speed

LAMOST DR7 was split in "neutral" mode into both CSV and Protobuf formats at various thresholds. File volumes are shown in Tables 5 and 6 , with read times in Figure 10 [Figure 10: see original paper]a. As threshold increases, read times for both formats decrease until stabilizing. Despite slightly larger file sizes than CSV, Protobuf achieves $> 50\%$ faster reads for thresholds > 1000, approaching continuous disk I/O speed.

We also tested self-matching: reading the unsplit catalog into memory, sorting by HEALPix ID as the source catalog, then computing target file IDs in order. If files exist in memory, cross-match with cached KD-Trees; otherwise, evict old KD-Trees, read new files, and build fresh KD-Trees. Figure 10 [Figure 10: see original paper]b shows cross-matching times versus threshold. Both formats show initial decrease then increase. CSV peaks at threshold ~ 1000 (~ 155 s), while Protobuf peaks at threshold $10,000$ (~ 116 s). Protobuf thus outperforms CSV for storage and cross-matching.

6.3 Curve and Cache Algorithm Efficiency

For different splitting levels, we traversed the sky using both HEALPix NESTED Z-curves and Peano-Hilbert curves. At region i, we read i and its eight neighbors (up, down, left, right, and four diagonals). Cache hits occur when regions reside in memory; misses trigger disk reads and cache insertion. When full, FIFO or LRU replacement selects victims. Miss counts for four combinations across four levels are shown in Figure 11 [Figure 11: see original paper]:

1) Larger caches reduce misses, but with diminishing returns per additional cache unit

2) LRU outperforms FIFO; Peano-Hilbert outperforms Z-curve

3) Performance gaps are independent of k value

4) Small caches dramatically reduce misses: Peano-Hilbert + LRU with cache size 9 reduces reads by $\sim 66.7\%$ versus no cache + FIFO; size 20 reduces reads by 77.4%

5) The combined Peano-Hilbert + LRU advantage over Z-curve + FIFO peaks at $\sim 1\times$ improvement with cache size 9

Peano-Hilbert curves with LRU replacement effectively improve cross-matching efficiency during catalog fusion.

7 Conclusion

For efficient large-scale catalog retrieval, this paper proposes a file-based solution encompassing splitting, indexing, and fusion. As shown in Figure 12 [Figure 12: see original paper], an adaptive density-based splitting algorithm divides large catalogs into size-limited subsets under memory constraints. Tests show our "neutral" mode restricts split catalog sizes to $[T, 4 \times T]$, achieving better uniformity than other modes and laying the foundation for efficient retrieval.

To accelerate retrieval, we introduce the open-source serialization library Protocol Buffers as the disk storage format. Storing coordinates in binary avoids performance costs from float-string conversions during splitting and retrieval, improving read efficiency.

During catalog fusion, adjacent files must be read to prevent source omission. We improve HEALPix traversal by replacing Z-curves with Peano-Hilbert curves and introducing caching with replacement algorithms, reducing file reads and accelerating fusion.

Future work includes:
- Extending single-threaded methods to multi-threading or distributed clusters
- Incorporating brightness and other parameters beyond coordinates to improve cross-matching accuracy in dense fields
- Porting file-based innovations to databases and other non-file systems for further optimization

This work was supported by data resources and technical assistance from the National Astronomical Data Center, Chinese Academy of Sciences Astronomical Data Center, China Virtual Observatory, and the National Astronomical Observatories-Alibaba Cloud Astronomy Big Data Joint Research Center. The National Astronomical Data Center is a national science and technology resource sharing service platform recognized by the Ministry of Science and Technology and the Ministry of Finance, affiliated with the Chinese Academy of Sciences National Astronomical Observatories.

References

[1] Ivezić Ž, Kahn S M, Tyson J A, et al. ApJ, 2019, 873(2): 111

[2] Chambers K C, Magnier E A, Metcalfe N, et al. https://arxiv.org/abs/1612.05560, 2016
[3] Xu Y F. PhD Thesis. Beijing: National Astronomical Observatories, Chinese Academy of Sciences, 2020: 21
[4] Gray J, Nieto-Santisteban M A, Szalay A S. https://arxiv.org/abs/cs/0701171, 2007
[5] O' Mullane W, Banday A J, Górski K M, et al. Mining the Sky. Banday A J, Zaroubi S, Bartelmann M, eds. Heidelberg: Springer, 2001: 638
[6] Koposov S, Bartunov O. Astronomical Data Analysis Software and Systems XV. Gabriel C, Arviset C, Ponz D, et al, eds. San Francisco: ASP, 2006, 351: 735
[7] Górski K M, Hivon E, Banday A J, et al. ApJ, 2005, 622(2): 759
[8] Berriman G B, Good J C, Shiao B, et al. Astronomical Data Analysis Software and Systems XXVII. Ballester P, Ibsen J, Solar M, et al, eds. San Francisco: ASP, 2020, 522: 191
[9] Szalay A S, Gray J, Thakar A R, et al. https://arxiv.org/abs/cs/0202013, 2002
[10] Thakar A R, Szalay A, Fekete G, et al. Computing in Science & Engineering, 2008, 10(1): 30
[11] Gao D. PhD Thesis. Beijing: National Astronomical Observatories, Chinese Academy of Sciences, 2008: 32
[12] Du P, Ren J, Pan J, et al. Science China Physics, Mechanics, and Astronomy, 2014, 57(3): 577
[13] Boch T, Pineau F, Derriere S. Astronomical Data Analysis Software and Systems XXI. Ballester P, Egret D, Lorente N P F, eds. San Francisco: ASP, 2012, 461: 291
[14] Pineau F X, Boch T, Derriere S. Astronomical Data Analysis Software and Systems XX. Evans I N, Accomazzi A, Mink D J, et al, eds. San Francisco: ASP, 2011, 442: 85
[15] Fernique P, Boch T, Donaldson T, et al. MOC-HEALPix Multi-Order Coverage map Version 1.0. https://www.ivoa.net/documents/MOC/20140602/index.html, 2014
[16] Fernique P, Allen M G, Boch T, et al. A&A, 2015, 578: A114
[17] Youngren R W, Petty M D. Heliyon, 2017, 3(6): 332
[18] Martinez-Castellanos I, Singer L P, Burns E, et al. AJ, 2022, 163(6): 259
[19] Hanisch R J, Farris A, Greisen E W, et al. A&A, 2001, 376: 359
[20] Wells D C, Greisen E W, Harten R H. A&AS, 1981, 44: 363
[21] de Berg M, van Kreveld M, Overmars M, et al. Computational Geometry: Algorithms and Applications. Heidelberg: Springer, 1997: 289
[22] Fan D W, He B L, Li C H, et al. Astronomical Research & Technology, 2019, 16(1): 69
[23] Kleppmann M. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. Sebastopol: O' Reilly Media Inc., 2017: 112
[24] Varda K. Protocol buffers: Google' s data interchange format. https://opensource.googleblog.com/2008/07/protocol-buffers-googles-data.html,

2008
[25] Reinecke M, Hivon E. A&A, 2015, 580: A132
[26] Bern M, Eppstein D, Teng S H. International Journal of Computational Geometry & Applications, 1999, 9(06): 517
[27] Warren M S, Salmon J K. Proceedings of the 1993 ACM/IEEE conference on Supercomputing. New York: Association for Computing Machinery, 1993: 12
[28] Schäfer B M. Dissertation. München: LMU: Faculty of Physics, 2005: 99

Note: Figure translations are in progress. See original paper for figures.