# Indexing and Fast Near-Matching of Billions of Astronomical Objects

Bin Fu, Eugene Fink, Garth Gibson, and Jaime Carbonell

Computer Science
Carnegie Mellon University
Pittsburgh, United States

{binf, e.fink, garth, jgc}@cs.cmu.edu

***Abstract*—When astronomers analyze sky images, they need to identify the newly observed celestial objects in the catalog of known objects. We have developed a technique for indexing catalogs, which supports fast retrieval of closely matching catalog objects for every object in new images. It allows processing of a sky image in less than a second, and it scales to catalogs with billions of objects.**

***Keywords-Large-scale data; eScience; indexing; astrophysics application***

## I. Introduction

When astronomers analyze telescope images, they check whether the newly observed objects are listed in catalogs of known objects. Due to atmospheric and optical distortions, the positions of celestial objects in a telescope image may change slightly from observation to observation. Thus, retrieval of exact catalog matches would be inadequate. Astronomers need to retrieve not only exact catalog matches but also close approximate matches.

The modern astronomical catalogs contain hundreds of millions of objects. For example, the Sloan Digital Sky Survey includes more than 300 million objects [1], and the Guide Star Catalog II (GSC-II) contains almost one billion objects [7]. Straightforward matching algorithms, such as a linear search through a catalog, are too slow for analyzing a stream of newly incoming imaging data.

We have developed a new technique for indexing massive catalogs and matching newly observed objects. On a standard desktop computer, it takes less than a second to match all objects in an image to a catalog with two billions objects.

## II. Problem

We assume that we have a catalog of known celestial objects and new images. Typically, each image covers a square region of the sky, whose area is several square degrees. For example, the area of each image in the Sloan Digital Sky Survey is 1.5 square degrees. An image may contain from a few hundred to a few hundred thousand objects, depending on the image size and the telescope resolution.

The position of each object is represented by two values, called right ascension and declination, which define its equatorial coordinates (Fig. 1). The right ascension, which is the celestial equivalent of longitude, ranges from 0.0 to 360.0 degrees; the declination, the celestial equivalent of latitude, ranges from −90.0 degrees (which represents the South Pole) to 90.0 degrees (the North Pole). We ignore the third spatial coordinate, that is, the distance from Earth to the object, since it is not directly observable and usually unknown during the initial stages of the image processing.

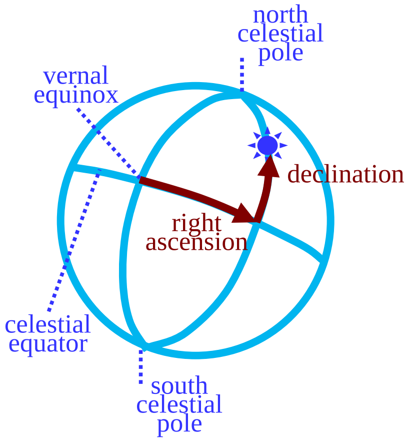

Figure 1. The representation of a celestial object in spherical coordinates.

Furthermore, astronomers also record the apparent magnitude of each object, which is the logarithm of its brightness. The apparent magnitude value serves as the “third coordinate”, which is used to identify the object along with its two spherical coordinates.

We assume that the edges of the image are parallel to the directions of right ascension and declination (Fig. 2). We need to find the matches for all image objects. Specifically, for each object $p$ in the image, we are looking for an object $q$ in the catalog such that:

- Among all objects in the image, $p$ is the nearest to $q$.
- Among all objects in the catalog, $q$ is the nearest to $p$.
- The distance between $p$ and $q$ in the two-dimensional spherical coordinates is at most 1 arc second, which is 1/3600 of a degree. The value of 1 arc second reflects the maximal possible observation error due to atmospheric and optical distortions.
- The difference between the apparent magnitudes of $p$ and $q$ is smaller than a given constant $C$.

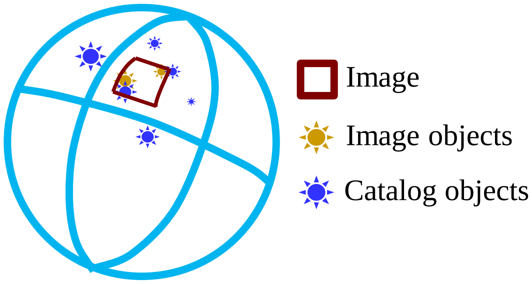

Figure 2. An example of the matching problem. There are two objects in the image, both of which have catalog matches.

If the catalog contains an object $q$ that satisfies all these constrains, we call it the *match* for $p$.

## III. INDEXING

The sizes of modern astronomical catalogs exceed the memory of desktop computers. For example, suppose that each celestial object is stored as a 14−byte record: 4 bytes for its right ascension, 4 bytes for its declination, 2 bytes for its apparent magnitude, and 4 bytes with a pointer to the respective record with more information about the object in an external database. Then a catalog with one billion objects takes 14 Gigabytes, which would not fit the memory of a regular desktop. We therefore store the catalog on disk and load only parts relevant to processing a given image.

We first describe the organization of the catalog on disk. We then present the retrieval procedure that identifies the relevant part of the catalog and loads it into memory. Finally, we explain the in-memory matching.

### A. *Indexing*

The indexing procedure arranges the catalog objects on disk, with the purpose to minimize the number of disk accesses during the retrieval of objects relevant to processing a given image.

We split the celestial sphere into multiple longitudinal strips, which are parallel to the direction of right ascension; each strip is exactly one degree wide (Fig. 3). In Section 5, we will further discuss the choice of the specific strip width and the reason for setting it to one degree in the current system.

The objects within a strip are stored as a separate file, where the elements in the file are in sorted order by their right ascension.

Since the whole catalog usually does not fit in memory, we cannot process all data in one pass. The described procedure is implemented indirectly in two passes. During the first pass, we read all catalog objects, and put them to the corresponding files without sorting. In the second pass, we load each file into memory, sort its objects, and store the file in sorted order.

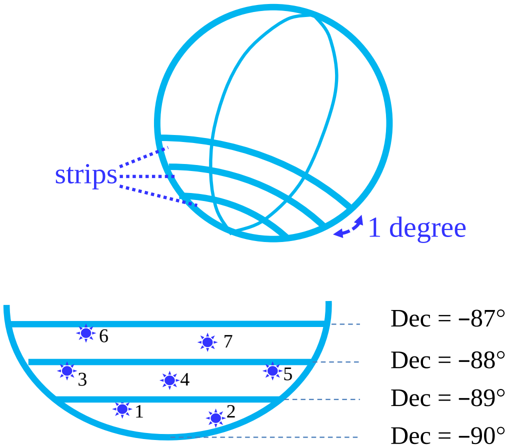

Figure 3. Indexing procedure. Top: The celestial sphere is divided into one-degree-wide strips. Bottom: We assume that there are seven objects in the catalog in this example, which are distributed among three strips. For each strip, the objects within the strip are sorted by their right ascension, and stored as a separate file.

### B. *Retrieval*

If we split the celestial sphere into $S$ strips, and $N$ catalog objects are about uniformly distributed among those strips, then the time complexity of this procedure is $O(N \cdot \lg(N/S))$, and the number of disk accesses during its execution is $O(N)$.

Given an image, we need to retrieve the catalog objects that may potentially match the image objects. Since a matching catalog object must be within 1 arc second from a newly observed object, possible matches must be located at most 1 arc second away from the image area. We determine the *axis-aligned minimum bounding box* of the image, which is the smallest rectangle that covers all image objects, with sides parallel to the directions of right ascension and declination. We then extend it by one arc second on all sides.

We retrieve all catalog objects inside the extended bounding box from the catalog files, which is done in three steps. The first step is to locate the strips that overlap the extended bounding box; the second is binary search within each respective file, which identifies all catalog objects whose right ascension value falls inside the extended bounding box; the third is to load all the related objects into memory.

To analyze the time complexity of the retrieval procedure, we again assume that the celestial sphere is split into $S$ strips, and $N$ catalog objects are about uniformly distributed among those strips. We further assume that the cost of each disk access is $c_1$ in average, and the cost of loading each object from disk to memory is $c_2$. If the extended bounding box of image covers $s$ strips and contains $n$ catalog objects, then it takes $O(s \cdot \lg(N/S) \cdot c_1)$ to use binary search to locate the extended bounding box of image on the catalog, and $O(n \cdot c_2)$ to load related catalog objects to memory.

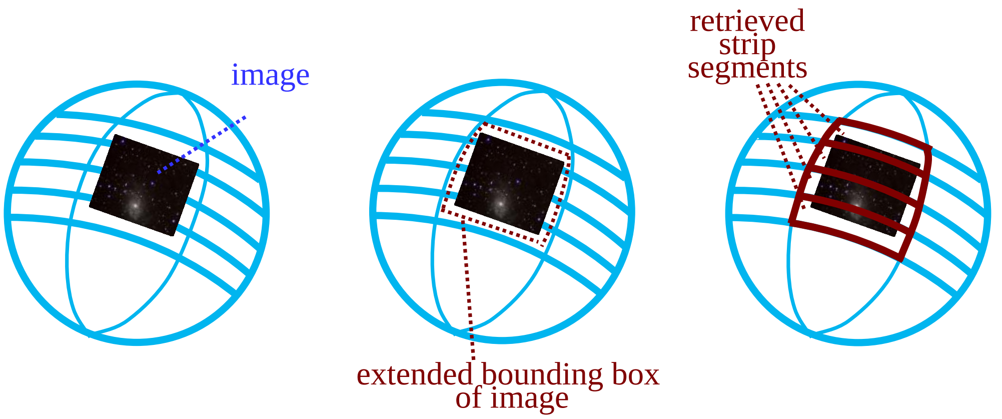

Figure 4. Retrieval procedure. Given an image, its axis-aligned minimum bounding box is calculated and extended by 1 arc second on all sides. Then, the part of the catalog that covers the extended bounding box of the image (that is, the retrieved strip segments) is loaded into memory.

### C. Matching

The last step is to identify matches among the objects loaded into memory. If the image contains $M$ objects, and $L$ objects are extracted from the catalog in the retrieval procedure, a naive matching algorithm would take $O(M \cdot L)$ time, which is impractically slow for real-time matching of images against large-scale catalogs. We next provide a more efficient technique.

The developed approach is similar to the "strip" idea used in the indexing procedure. Specifically, we further subdivide the retrieved strips into thinner *substrips*. These substrips are also parallel to the direction of right ascension, and each substrip is exactly one arc second wide. The objects within each substrip are sorted by their right ascension, which allows the use of binary search for identifying close catalog objects for each image object.

For each image object, since its match can be at most one arc second away, we consider only the catalog objects in its three nearby substrips, that is, its own substrip and the two adjacent substrips. We illustrate the matching procedure in Fig. 5 and give pseudocode in Fig. 6.

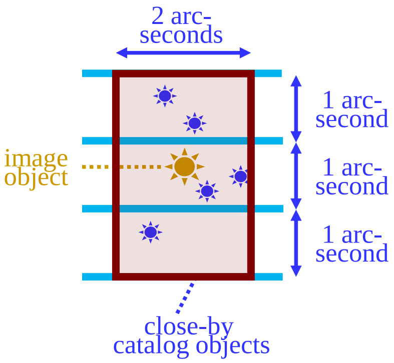

Figure 5. Illustration of the matching procedure. For each image object, we extract all catalog objects that are at most one arc second away, and then calculate their exact distances to the image object.

**Input:**

$q_1, q_2, \ldots, q_M$: Image objects.

$c_1, c_2, \ldots, c_L$: Extracted catalog objects that are possible to match the image objects. This is the output of the retrieval procedure.

**Output:** Possible match for each image object.

BestI[1, 2, …, $M$] and DistanceI[1, 2, …, $M$]: BestI[$i$] stores the index of the closest catalog objects to $q_i$. The distance between $q_i$ and $c_{\mathrm{BestI}[i]}$ is stored in DistanceI[$i$].

BestC[1, 2, …, $L$] and DistanceC[1, 2, …, $L$]: BestC[$j$] stores the index of the closest image objects to $c_j$. The distance between $c_j$ and $q_{\mathrm{BestC}[j]}$ is stored in DistanceC[$j$].

```
for $i$ = 1 to $M$ do DistanceI[$i$] = MAX; BestI[$i$] = 0
for $j$ = 1 to $L$ do DistanceC[$j$] = MAX, BestC[$j$] = 0
Split the extracted catalog area into substrips, and sort the catalog objects in each substrip
for $i$ = 1 to $M$ do
    // Find possible match for $q_i$
    Retrieve the nearby 3 substrips of $q_i$, and conduct binary searches in the three substrips to retrieve the catalog objects $r_1, r_2, \ldots, r_K$ that are at most 1 arc second away from $q_i$, and with apparent magnitudes at most $C$ from $q_i$ (Fig. 5).
    for $k$ = 1 to $K$ do
        //Assume $r_k = c_j$. Compute the distance between $q_i$ and $c_j$
        $d$ = distance($q_i$, $c_j$)
        if $d$ < DistanceI[$i$] then DistanceI[$i$] = $d$; BestI[$i$] = $j$
        if $d$ < DistanceC[$j$] then DistanceC[$j$] = $d$; BestC[$j$] = $i$
for $i$ = 1 to $M$ do
    // Output the possible match for $q_i$
    $match$ = BestI[$i$] // $c_{match}$ is the closest catalog object to $q_i$
    if $match$ > 0 && BestC[$match$] == $i$ then
        // $q_i$ is also the closest to $c_{match}$
        Output_Match($q_i$, $c_{match}$)
```

Figure 6. Matching algorithm.

# IV. Experiments

We have evaluated the running time of each procedure described in Section III: the indexing procedure, the retrieval procedure, and the matching procedure. We have used synthetic catalog data with a random uniform distribution of objects across the sky.

We have run the experiments on a desktop computer with Pentium Xeon 2.8 GHz dual quad core, 16GB memory, and 7200 RPM 160 GB disk. The described algorithms are implemented in Java 1.6. All data points in the summary graphs are the mean of three runs with system caches flushed between runs.

## A. *Indexing*

We show the running time of the indexing procedure in Fig. 7. As discussed in Section 3.1, its time complexity is $O(N \cdot \lg(N/S))$ where $N$ is the number of objects in the catalog and $S$ is the number of strips, which matches the observed empirical results. Specifically, the running time is about $1.85 \cdot 10^{-7} \cdot N \cdot \lg(N/S)$ seconds. It takes about 6,000 seconds (1.7 hours) to index a catalog of two billion objects. Note that this procedure has to be run only once for the given catalog, and occasionally rerun later after updates of the overall catalog.

## B. *Retrieval*

The two main factors affecting the retrieval time are the number of objects in the catalog, and the area of the image. On the other hand, the number of objects in the image does not affect the retrieval time.

We breakdown the retrieval time into three parts: the time to load image objects from a file (image loading), the aggregate time on binary search when we identify all the catalog objects falling in the bounding box of image (binary searches), and the time to load the catalog objects to memory (catalog loading). Specifically, using the notations in Section III B, the retrieval time is about $0.0028 \cdot \lg(N/S) \cdot s + 9 \cdot 10^{-7} \cdot n$ seconds.

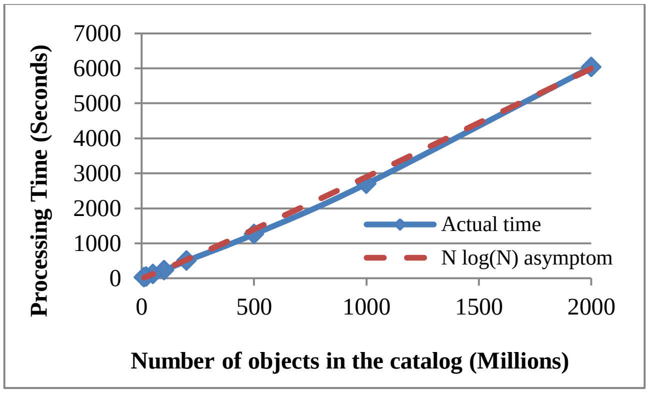

Figure 7. Running time of the indexing procedure.

The top graph in Fig. 8 shows the dependency of the retrieval time on the number of objects in the catalog. The bottom graph in Fig. 8 shows the relationship between the retrieval time and the length of the side of the square image. It takes about 0.5 second for a large image (3.5 × 3.5 degrees) and a large catalog (2 billion objects).

## C. *Matching*

We have evaluated the dependency of the matching time on three parameters: the number of objects in the image (top of Fig. 9), the side length of the image (middle of Fig. 9), and the number of objects in the catalog (bottom of Fig. 9). The running time is under 0.6 second in all cases. Most of the running time is spent on the substrip division and sorting.

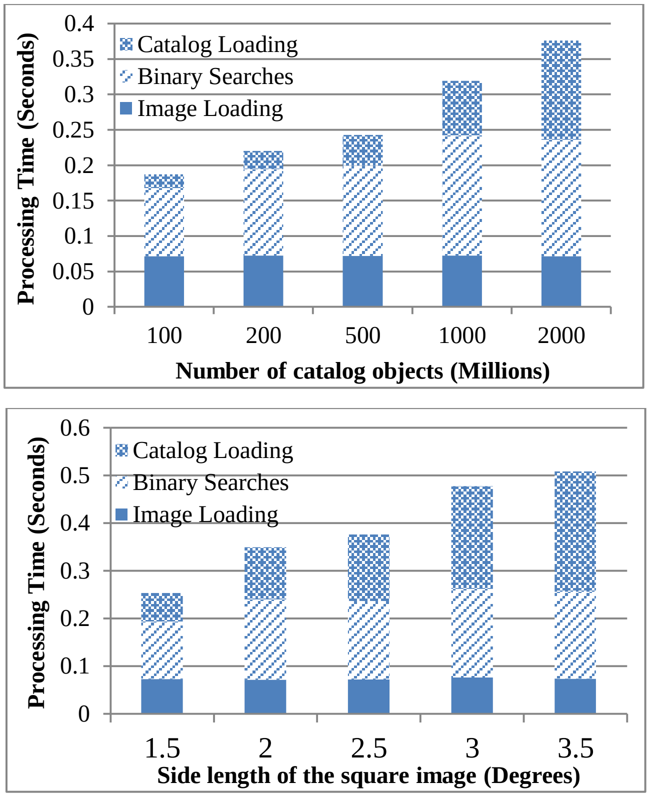

Figure 8. Retrieval time. We use 2.5 × 2.5 degree images and a catalog with 2 billion objects as the baseline. We show the dependency of the running time on the catalog size for 2.5 × 2.5 degree images (top), and the dependency of the time on the side length of the square image for a catalog with 2 billion objects (bottom).

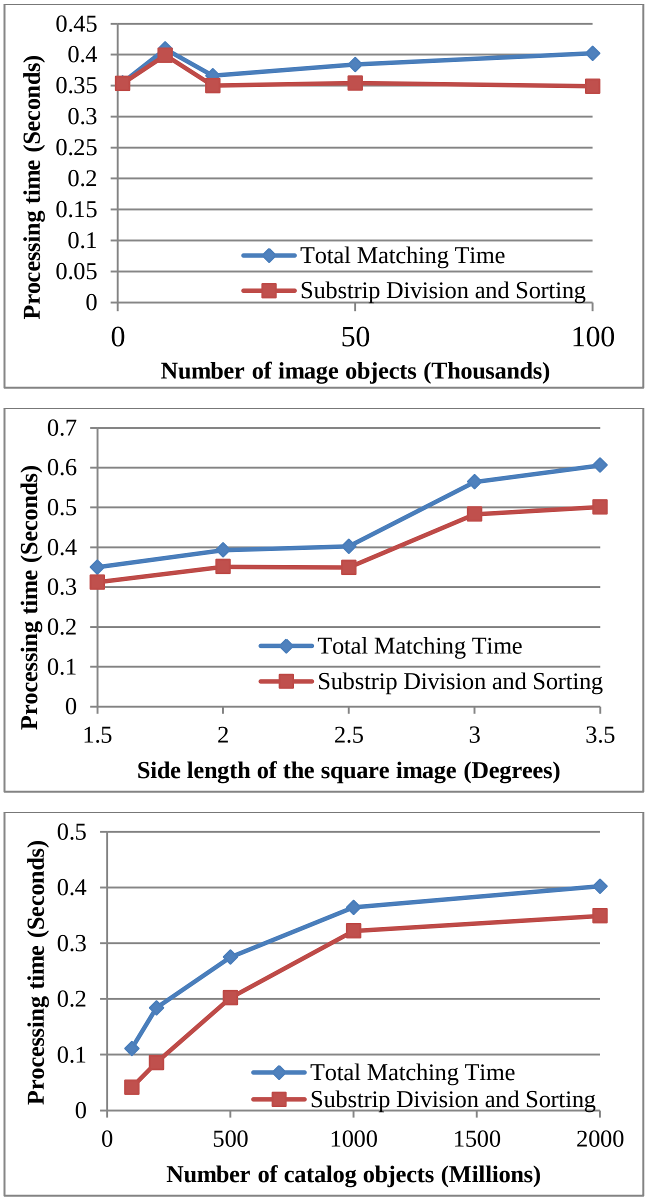

Figure 9. Matching time. The baseline experiment is with a catalog of 2 billion objects, and a 2.5 × 2.5 degree image with 100 thousands objects. We show the dependency of the matching time on the number of image objects (top), the side length of the image (middle), and the number of catalog objects (bottom).

## V. DISCUSSION

We next discuss some issues related to the problem and the proposed approach.

### A. *Indexing method*

We use a simple method of splitting the celestial sphere into strips. There are other standard ways to divide the sphere into multiple parts and index them, such as the Hierarchical Triangular Mesh [12], that some datasets may benefit from.

### B. *Updating catalog*

We may need to perform occasional updates of the catalog to add newly discovered objects or delete some of the old objects. The described technique provides an efficient solution for such updates. To insert and delete celestial objects, we load each related strip into memory, make insertions and deletions, re-sort each strip, and store the updated strips on disk.

### C. *Width of Strips*

We have set the strip width to one degree. We now explain the reason to this choice.

The main related tradeoff is that, if the strips are too wide, we retrieve many catalog objects that do not match objects in the image; on the other side, if the strips are too narrow, we need to open many files to conduct binary search, thus incurring high disk-access costs. In Fig. 10, we show the dependency of the retrieval time on the strip width, which confirms that the use of 1-degree strips leads to the fastest retrieval of 2.5 × 2.5 degree images.

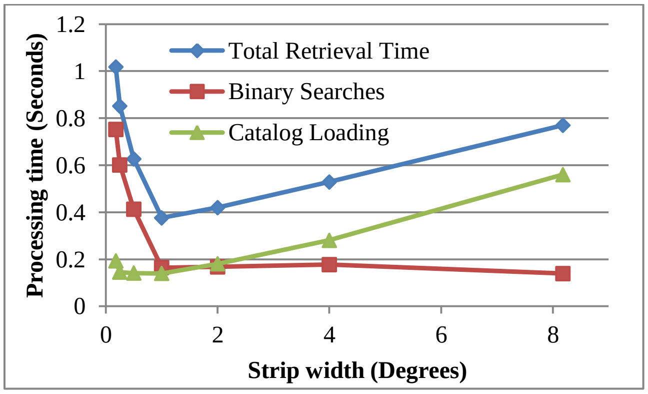

Figure 10. Impact of the strip width on the retrieval time for 2.5 × 2.5 degree images.

## VI. RELATED WORK

The described matching problem can be formulated as a range query: Find all catalog objects that are within 1 arc second from the image objects. Space-partitioning data structures, such as R-tree [6] and *kd*-tree [3] and others [13], can be used for such queries. However, the retrieval based on these structures is significantly slower than the described technique, especially when the catalog is too big for memory.

Scientists from the database community have also developed several tools for organizing astronomical data [2]. Although traditional database technologies are difficult to use for efficient handling large astronomical databases [9], the emergence of new database technologies, namely, Object Data Management Systems, and Object-Relational Database Management Systems, now provides spatial indices for astronomical data, and already used by researchers [4].

For example, two widely-used open source databases, MySQL [8] and PostgreSQL [5], support R-tree spatial indices [11, 10]. It is a promising direction since the database approach is straight-forward and easy to implement, and it provides more functions than a stand-alone implementation.

However, the scalability of current off-the-shelf solutions is limited. Our experiments show that it takes over two thousand seconds (33 minutes) to load 50 million objects into the database and create the spatial index. Furthermore, the database approach requires 5 GB to store those 50 million objects, while our solution takes only 600 MB on the same dataset.

## VII. Conclussion

If we have a catalog with billions of objects, how do we index it to support fast matching operations? We have tackled this problem on a standard desktop computer. We propose a way to organize the catalog on disk and dynamically loading relevant parts of the catalog into memory, thus achieving very good performance. Experiments on a catalog with 2 billion objects show that building a catalog takes less than 2 hours, and the retrieval and matching for an astronomical image takes less than a second.

## Acknowledgment

We thank Michael Wood-Vasey and Joel Welling for helping us understand the related astronomical problem and providing thorough feedback on our results. We also thank Julio Lopez, Lei Li, and Helen Mukomel for their comments and insights.

This work was sponsored in part by grants from Google, The Moore Foundation, National Science Foundation Open Cloud Consortium, The Petascale Data Storage Institute (PDSI), The McWilliams Center for Cosmology and the companies of the Parallel Data Laboratory Consortium (PDL).

## References

[1] K. Abazajian *et al*. The seventh data release of the Sloan Digital Sky Survey. The Astrophysical Journal Supplement Series, 182, 2009.

[2] Andrea Baruffolo. R-trees for astronomical data indexing. Astronomical Data Analysis Software and Systems VIII, ASP Conference Series, 172, 1999.

[3] J. L. Bentley. Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9), pages 509–517, 1975.

[4] I. Chilingarian, O. Bartunov, J. Richter, and T. Sigaev. PostgreSQL: The suitable DBMS solution for astronomy and astrophysics. In ASP Conference Series, 314, ADASS XIII, pages 225–228, 2004.

[5] Korry Douglas and Susan Douglas. PostgreSQL. SAMS Publishing, 2003.

[6] Antonin Guttman. R-Trees: A dynamic index structure for spatial searching. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 47–57, 1984.

[7] Barry Lasker *et al*. The second-generation Guide Star Catalog: Description and properties. The Astrophysical Journal, 136, pages 735–766, 2008.

[8] MySQL, MySQL 5.1 Reference Manual, Chapter 18, Spatial Extensions in MySQL.

http://dev.mysql.com/doc/refman/5.1/en/index.html

[9] B. Pirenne and F. Ochsenbein. The Guide Star Catalogue in STARCAT. Space Telescope European Coordinating Facility Newsletter, 15, pages 17–19, 1991.

[10] PostGIS: http://postgis.refractions.net/

[11] Philippe Rigaux, Michel Scholl and Agnes Voisard.Spatial Databases: With Application to GIS. Morgan Kaufman, Los Altos, 2002.

[12] Alex Szalay, Jim Gray, Gyorgy Fekete, Peter Kunszt, Peter Kukol, and Ani Thakar. Indexing the sphere with the hierarchical triangular mesh, 2005. Technical Report MSR-TR-2005-123.

[13] A. J. Wicenec and M. Albrecht. Methods for structuring and searching very large catalogs. Astronomical Data Analysis Software and System VII, ASP Conference Series, 145, 1998.