中国科学院大学

University of Chinese Academy of Sciences

硕士学位论文

面向大视场时域巡天观测的大数据检索与融合方法研究

作者姓名：张琦乾

指导教师：樊东卫 副研究员 中国科学院国家天文台

学位类别：理学硕士

学科专业：天文技术与方法

培养单位：中国科学院国家天文台

2023 年 6 月

Large-scale Data Query and Fusion Method Research for Wide-field Transient Sky Survey

A thesis submitted to

University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Natural Science

in Astronomical Technology and Method

By

ZHANG Qiqian

Supervisor: Associate Professor FAN Dongwei

National Astronomical Observatories, Chinese Academy of Sciences

June, 2023

中国科学院大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文是本人在导师的指导下独立进行研究工作所取得的成果。承诺除文中已经注明引用的内容外，本论文不包含任何其他个人或集体享有著作权的研究成果，未在以往任何学位申请中全部或部分提交。对本论文所涉及的研究工作做出贡献的其他个人或集体，均已在文中以明确方式标明或致谢。本人完全意识到本声明的法律结果由本人承担。

作者签名：张琦乾

日　　期：2023年6月2日

中国科学院大学
学位论文授权使用声明

本人完全了解并同意遵守中国科学院大学有关收集、保存和使用学位论文的规定，即中国科学院大学有权按照学术研究公开原则和保护知识产权的原则，保留并向国家指定或中国科学院指定机构送交学位论文的电子版和印刷版文件，且电子版与印刷版内容应完全相同，允许该论文被检索、查阅和借阅，公布本学位论文的全部或部分内容，可以采用扫描、影印、缩印等复制手段以及其他法律许可的方式保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名：张琦乾　　导师签名：樊东卫

日　　期：2023年6月2日　　日　　期：2023、6、2

摘 要

国内外多个大视场时域巡天观测计划预期将会获得海量的天文观测数据，如何更高效地从这些数据中检索、挖掘出有价值的天文信息是一个具有重要意义的问题。尤其是对超新星、伽马射线暴和引力波事件的光学对应体等暂现源的搜寻，需要大视场望远镜对大面积天区连续观测，并在一个曝光周期内完成数据的处理以及与历史数据的检索和交叉证认等工作，以便能够在尽可能短的时间内发现正在变化的天体，并及时发出预警。因此，如何在尽可能短的时间里完成对大规模星表的快速检索是后续交叉证认、多波段星表融合和暂现源搜寻工作的基础。本文对如何提高数十亿天体大规模星表的检索与融合效率做了研究，提出了一系列的解决与优化方案，主要工作如下：

基于文件的星表动态层级划分和检索算法。当前的计算机体系结构决定了数据只有先读入到内存中才能检索与处理。现有的大规模星表的数据量往往高达数个 TB，普通计算机难以将星表完整地读取到内存中，在检索时需要频繁读写硬盘，严重影响了检索的速度。而星表检索的过程存在空间局部性，即一般只需要检索一个坐标邻近天区内的天体，无需检索整个星表。针对这一特点，本文提出了基于 HEALPix 的多层级星表划分算法，该算法能够根据不同天区的天体密度，动态选择最合适的切分层级，将星表切分成大小合适且均匀的文件。在检索时，根据检索需求，只读取并检索对应区域的星表文件，从而提升了检索速度。

星表的存取和缓存优化研究。为了提升星表在存取时的速度，本文使用开源的序列化库制定了一套针对星表的序列化存取协议，该协议使用二进制格式存储天体的坐标，以提升读写效率。检索前若能将多个星表以一定的方式加以融合，在检索时只需一次检索就可以获得某个天体在多个星表中的信息。因此本文对星表融合时的缓存优化问题做了研究，使用对缓存更友好的遍历顺序和缓存替换算法。经过测试，该方法能够有效提升缓存的利用率，减少缓存失效次数，提升了融合时的速度。

分布式的星表检索与调度系统。单机检索受到硬件条件等诸多因素的限制，速度难以继续提升。本文尝试在上述优化方案的基础上，构建一套分布式星表检索系统，并研究了星表在各个节点中的分发策略和检索时的智能调度策略。该系统在检索时能够将检索请求发送到合适的计算节点，实现多节点并行检索。经过实际测试，在节点数量分别为 2、3、4 个的情况下，该系统的检索速度可以达到单个节点速度的 1.84、2.75、3.31 倍。

本文工作能够有效提升大规模星表的检索速度，有助于后续大规模巡天计划和暂现源搜寻任务更好地开展。

关键词：天文大数据，分布式系统，天文数据检索，数据融合，虚拟天文台

Abstract

A number of domestic and foreign wide-field time-domain surveys are expected to obtain a large amount of astronomical observation data. How to retrieve and mine valuable objects from these data more efficiently is a significant issue. In particular, searching for transient sources such as supernovae, optical counterparts of gamma-ray bursts, gravitational waves, and other events requires uninterrupted observations with wide-field telescopes and processing of data and cross-match with historical data within one exposure cycle. Only in this way, transient objects can be detected in the shortest time, and early warnings can be issued in time. Therefore, quickly completing the retrieval of large-scale catalogs within the shortest possible time is vital for subsequent cross-match, multi-wavelength catalog fusion, and transient source search tasks. In this thesis, we research how to improve the retrieval and fusion efficiency of large-scale catalogs with billions of celestial bodies and propose a series of solutions and optimizations. The main work is as follows:

File-based dynamic hierarchical split and retrieval algorithms for catalogs. The computer architecture determines that data must be read into memory before processing. The size of existing large-scale catalogs is often as large as several terabytes, making it difficult for personal computers to read the catalogs into memory fully, so they can only read and write to the hard disk frequently during processing, which slows down the speed of retrieval seriously. However, the process of catalog retrieval has spatial locality, i.e., most tasks only need to retrieve the objects within a neighboring sky area, not the whole catalog. Based on this feature, this thesis proposes a multi-level catalog split algorithm based on HEALPix, which can dynamically choose the most appropriate splitting level according to the density of objects in different regions, splitting the catalog into uniformly sized files. During retrieval, only the catalog files in the corresponding and retrieved regions need to be read according to the retrieval demand, which improves the retrieval speed.

Research on storage access and cache optimization of catalogs. In order to improve the speed of catalogs during access, this thesis uses an open-source serialization library to develop a serialization access protocol for catalogs, which uses binary storage of floating celestial coordinates to improve read-write efficiency. If multiple catalogs can be merged before retrieval, the information of a particular object in multiple catalogs can be obtained in retrieval once. In this thesis, we research the cache optimization issues during catalog merging. By using a more cache-friendly traversal order curve and a better cache replacement algorithm, we can effectively improve the cache utilization, reduce the number of cache failure, and improve the speed when merging after testing.

Distributed catalog retrieval system. The speed of a single-machine retrieval is limited by hardware conditions and is difficult to improve further. In this thesis, we try to build a distributed catalog retrieval system based on the above optimization scheme. And we proposed the distribution strategies of catalogs among nodes and intelligent scheduling strategies during retrieval. This system enables retrieval requests to be sent to suitable computing nodes for multi-node parallel retrieval during searches. Through actual testing, the system was able to achieve retrieval speeds that are 1.84, 2.75, and 3.31 times the speed of a single node when there are respectively 2, 3, 4 nodes.

The work presented in this thesis can effectively improve the retrieval speed of large-scale star catalogs, thereby assisting subsequent large-scale sky survey projects and transient source search tasks.

Key Words: Astronomical Big Data; Distributed System; Astronomical Data Query; Data Fusion; Virtual Observatory

目 录

图目录

表目录

第 1 章　绪　论

1.1　研究背景及意义

天文学是一门探索宇宙起源、结构和演化的古老科学。1609 年伽利略将望远镜对准木星及其卫星的那一刻，开创了使用望远镜观测天体的新纪元。从此人类的视野便不断扩展，到如今迈入多波段、多信使观测的时代，天文学的发展随着观测能力的提高而不断进步。近代以来人类的科学技术水平飞速进步，特别是计算机信息技术广泛应用，天文观测所获取到的数据量也呈现出“爆炸式”的增长趋势。从 20 世纪 90 年代开始，国内外提出并开始运行许多大规模巡天计划，这些巡天计划利用大视场望远镜，加上大靶面探测器，以前所未有的方式获得了远远超出了人类手工处理极限的大量观测数据。这些观测数据包含了丰富而深刻的天文信息，为揭示宇宙中各种奇异现象和物理过程提供了有力的证据和线索。然而，如何高效地管理、分析和挖掘这些浩如烟海的数据，从中取得有价值的发现，也成为了当前天文学面临的一个重要机遇和挑战。

1.1.1　国内外大规模巡天计划简介

国内外大规模巡天计划中，最著名的莫过于从 1998 年开始的斯隆数字化巡天项目（Sloan Digital Sky Survey，SDSS），SDSS 是一项旨在对宇宙进行三维测量的大型国际合作项目。该项目最初使用位于美国新墨西哥州阿帕奇天文台（Apache Point Observatory）的 2.5 米望远镜拍摄图像和光谱，该望远镜有着 3 平方度的大视场 (York et al., 2000)。到如今正在进行的第五期巡天（SDSS-V）包括位于智利的杜邦望远镜（Irénée du Pont Telescope）在内，已有南北半球的多台望远镜参与其中。SDSS 现已发布 18 批科学数据，其中仅 DR8 到 DR17 的总数据量就已经超过 623 太字节（Terabyte，TB）(Abdurro’uf et al., 2022)。SDSS 的数据为星系形成和演化、暗物质和暗能量的性质、恒星结构和化学组成、地外生命等方面带来了革命性的研究成果。

盖亚（Gaia）卫星是由欧洲空间局（European Space Agency，ESA）发射的一颗天文观测卫星，于 2013 年 12 月 19 日从法属圭亚那航天中心（The Guiana Space Centre）发射升空，进入距离地球约 150 万公里的拉格朗日点 L2 轨道。它的主要任务是以前所未有的精度测量银河系中的恒星进行位置、距离、运动、光度和光谱，绘制出银河系最大、最精确的三维地图，以揭示银河系的结构、形成和演化历史 (Gaia Collaboration et al., 2016)。它由两个夹角为 106.5 度的望远镜和一个有 106 块 CCD 芯片的相机组成。2022 年 4 月 29 日，盖亚卫星发布了第三批科学数据（Gaia DR3），共有超过 18 亿颗天体的信息，包括位置、视差、自行、光度、色指数等参数，以及部分恒星的径向速度和光谱分类。这是目前最大最精确的天体目录，压缩后的数据大小接近 10TB (Gaia Collaboration et al., 2022)。

宇宙中的暂现源往往蕴含着剧烈的能量活动，有着极高的研究价值。例如对

研究引力透镜、寻找超新星、确定伽马射线暴的物理性质、搜寻引力波对应体、探测活动星系核的结构、研究变星、发现系外行星等许多前沿天体物理课题有着重要的作用。

对暂现源的搜索，需要能快速曝光的大视场望远镜或望远镜阵对全天进行快速监测，并对拍摄的数据快速处理。例如帕洛马瞬变工厂（Palomar Transient Factory，PTF）是使用位于加利福尼亚州圣地亚哥的帕洛玛天文台 48 英寸的望远镜的暂现源巡天项目，有着 7.3 平方度的超大视场，每次曝光 60 秒，该项目在 2009 年 3 月至 2012 年 12 月间运行。并在 2013 年升级到 iPTF(The intermediate Palomar Transient Factory)，2016 年 9 月发布的 DR3 数据中包括了超过 65 万张巡天图像，在运行期间发现了超过 2500 颗超新星。之后，做了包括将原有的 CCD 升级到 47 平方度等一系列升级，项目更名为 ZTF（Zwicky Transient Facility）并于 2017 年开始巡天观测，ZTF 配备有 16 块 6000*6000 像素的 CCD，每小时可以曝光 700 次，巡天 3760 平方度的天区，每晚会产生大约 1TB 的图像数据，一个三年的巡天周期预期会产生大约 3.2 PB 的总数据量 (Masci et al., 2019)。

由夏威夷大学天文学研究所开展的泛星计划（Panoramic Survey Telescope and Rapid Response System，Pan-STARRS）使用四个口径 1.8 米的望远镜组成阵列，运行期间预期将产生超过 100TB 的数据 (Kaiser, 2004)。Pan-STARRS 基于第一台望远镜 Pan-STARRS1（PS1），现已发布了 DR1、DR2 两批数据。2019 年 1 月 28 日发布的第二批数据（PS1 DR2）大小超过了 1.6PB ，使其在当时成为了有史以来单次发布最大数量的天文数据[1]，数据目录包括五个滤光片（grizy）的测量结果，覆盖赤纬-30 度以北的 30,000 平方度的天空，其中的 ObjectThin 星表包含了 105 亿行数据（约 5.4TB），StackObjectThin 表包含 34.7 亿行数据（约 1.2TB）。

建设在智利薇拉 · 鲁宾天文台的 8.4 米口径望远镜（Vera C. Rubin observatory legacy survey of space and time，LSST），拥有 9.6 平方度的大视场，使用 189 块 4K*4K 的 CCD，总像素高达 32 亿，甚至创造了“世界最高分辨率相机”的世界记录[2]。LSST 计划开展为期 10 年的南半球 18,000 平方度巡天。在十年中，每一个天区将会被巡访 1000 次以上，获得 400 亿天体的观测数据以及一个空前规模的动态时域巡天数据库。每晚将产生超过 15TB 的数据，这些数据将会被实时处理并将释放超过千万量级的暂现源信息 (Ivezić et al., 2019)。

地基广角相机阵（Ground Wide Angle Cameras，GWAC）是我国与法国合作的多波段空间天文变源检测器（Space-based multiband astronomical Variable Objects Monitor，SVOM）的地基相关观测设备 (Cordier et al., 2015)，一共由 40 台有效口径为 18 厘米的大视场望远镜组成，每台望远镜配备有一台 4K ×4K 的高性能相机，覆盖约 5000 平方度的天区，单幅图像拍摄时间 15 秒，极限星等约 16 等。按照设计，GWAC 单台相机所拍摄的每幅图像内将包含大约 12000 个天体，整个系统每晚将产生大约 2.7TB 的数据，10 年运行期间将产生超过 9PB 的数据。为

[1]https://outerspace.stsci.edu/display/PANSTARRS/PS1+News#PS1News-2019.01.28:PS1DataRelease2

[2]https://www.guinnessworldrecords.com/world-records/78317-highest-resolution-digital-camera

了发现微引力透镜等短时标的天文现象，GWAC 需要在 15 秒内完成对整个系统拍摄数据的处理工作，对处理的速度有着严格的要求 (Wan et al., 2016)。

“司天工程”是我国正在建设的地基光学观测网络，计划使用分布在全球各地的 72 台 1 米级望远镜和 3 台 4 米级望远镜以 30 秒的频率对超过 10000 平方度的天区高频率巡天，以探测和跟踪各种光学暂现源和高能天文事件的光学对应体（如引力波事件、快速射电暴、超新星等）。据估计，该计划每分钟将产生超过 40GB 的原始数据，每晚预计超过 24TB 的原始数据或 140TB 的处理后的数据，每年产生的数据量将高达 40PB (Liu et al., 2021)。

1.1.2　海量天文数据带来的挑战

这些大规模巡天计划每天将产生大量的观测数据，尤其是如 GWAC 等大视场暂现源搜寻任务，更需要实现对拍摄的数据的实时处理，每次曝光的图像将在天文台当地或者通过网络实时传送到数据处理中心进行存储与处理。原始图像在经过一系列的处理和流水线校准过后，从中提取天体的位置与亮度等信息。这些数据将与之前观测的数据匹配并生成光变曲线，或者与其它巡天观测数据做交叉证认，以判断是否有正在发生变化的天体，并发出预警，这些工作都要在一个曝光周期内完成。这对计算机的检索与数据处理能力提出了很高的要求。

一个高效率的检索算法可以在尽可能短的时间内从已有的大规模星表中检索得到需要的天体信息，大幅提高数据处理的速度和准确度。传统的单机检索受限于硬件配置等原因，已经很难满足于当前检索速度的要求，亟需一个高效的多节点分布式检索方法。

这些巨量数据所带来的另一个问题是，不同巡天观测产生的天文数据都是分散在各个国家的不同天文机构中。当天文学家们需要使用这些数据做研究时，传统的方法是先去这些机构的网站，找到对应数据的发布地址，并花费大量的时间从机构所在国家的服务器下载数据到本地的计算机中，才能进行后续的研究。

例如 Gaia DR3 的数据存储于欧洲航天局天文中心（the European Space Astronomy Centre，ESAC）在马德里的数据中心，中国的国家天文科学数据中心（National Astronomical Data Center，NADC）花费了将近 3 周的时间才完成对该数据的镜像工作[3]。

如何利用计算机网络将分散在世界各地的天文观测资源整合，是虚拟天文台（Virtual Observatory，VO）研究的核心课题之一。世界上现在已经建成了多个虚拟天文台，提供对天文数据的归档、检索和计算服务。其中一项核心组成部分就是研究多波段异构星表的交叉证认 (Malkov et al., 2012)，即将不同望远镜的不同波段巡天观测获得的数据进行整合，对数据做统一的整理、归档，并提供功能强大的计算、数据检索与分析工具。用户只需要一次检索就可以非常方便地获取到各个波段、各个巡天计划的数据并加以研究。

可以预见，随着未来的大规模巡天数据量越来越大，使用自己的计算机在本

[3]https://nadc.china-vo.org/article/20220705162222

地做数据的处理和分析将会变得越来越困难，天文学家需要转变传统数据访问方式，将越来越多的工作转向“云端”完成 (崔辰州 等, 2015)，这是国家天文科学数据中心和中国虚拟天文台未来将会提供的服务之一。使用在云端的海量天文数据的高性能索引服务，天文学家们只需要将自己的检索需求在浏览器中提交，即可自动检索多个巡天计划的星表，找到符合要求的天体信息，并返回给天文学家。更进一步，未来将会推出天文大数据科研系统，可以将科研的整个工作流程全部搬到云端进行，将会实现计算与数据的协同调度，并支持在异构环境下的科研应用环境的自动部署[4]。

1.2 国内外研究现状

由于大规模星表的数据量往往过于巨大，直接检索较为困难。常见做法是将星表在检索前按照一定的规则做预处理或切分，以减轻检索时的负担。现在已经发展出了多种天球切分算法，并在此基础上产生了许多星表检索与交叉证认技术。

1.2.1 天球划分算法

天球划分算法是指根据一定的规则将天球划分成多个区域，并赋予对应的编号。最常见的划分算法是使用经纬度网格进行区域划分。类似于地球的经纬度，天球中的赤经赤纬将天球划分成了不同大小的网格。但是问题在于，赤纬中只有天赤道是经过球心的大圆，越靠近两极，赤纬圈的弧长越短。因此若仅仅按照固定角度间隔的赤经赤纬对天球切分，得到区域的面积并不相等。

已有的天球切分算法有很多，例如较为常用的条带算法（Zones Algorithm; Gray et al. 2007），分层三角网格（hierarchical triangular mesh，HTM; O'Mullane et al. 2001），四叉树立方体（quad tree cube，Q3C; Koposov & Bartunov 2006），多级等面积同纬度划分法（Hierarchical Equal Area isoLatitude Pixelation，HEALPix[5]; Górski et al. 2005）[759] 等。

条带算法是由 Jim Gray[6]等人提出，主要思想是将天球按照相等赤纬间隔划分为多个条带（Zones），若划分间隔为 h，则每个条带按照其中心赤纬 δ 使用公式1-1可以得到一个编号。任何一个天体根据其所在的赤纬可以很快计算出该天体所在条带的编号。在检索或交叉证认等场合下，可以先根据待检索坐标计算并只读取该坐标附近条带，减少了计算的数据量与计算时间。

$$ZoneID = \lfloor \frac{\delta + 90}{h} \rfloor \tag{1-1}$$

HTM 是将天球以一个八面体作为初始层级开始划分，八面体的每个面都是一个三角形，每个三角形都可以继续被切分成四个更小的三角形，该过程不断递

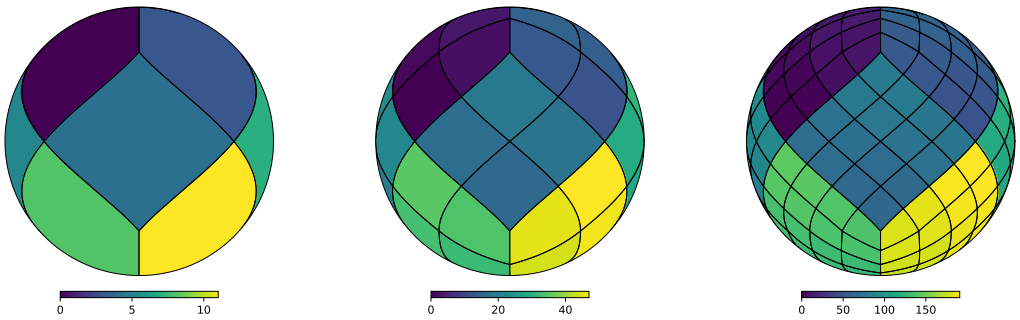
[4]https://nadc.china-vo.org/article/20230328165504

[5]https://healpix.sourceforge.io

[6]https://jimgray.azurewebsites.net/

归迭代直到达到预定的分辨率。

HEALPix 类似于 HTM，也是将天球按照层级划分，层级越大天球被划分的区域越多。该方法由 Górski 等人提出，HEALPix 最开始应用于宇宙微波背景辐射的研究与分析，如今已经在天文学研究中广泛使用。该算法与 HTM 的不同之处在于 HEALPix 首先将球面划分成 12 个等面积的四边形，此为层级 NSIDE=1 的划分。将每个大四边形横竖切成 4 个小四边形后，整个球面被划分成 48 个等面积的四边形，此为层级 NSIDE=2 的划分，如图1-1所示。同理，每个小四边形又能继续划分成四个更小的四边形，此过程可以一直递归进行。一个预先设定好的层级 NSIDE=2^k 可以将球面递归划分成 12×4^k 个四边形区域，每个四边形区域称之为“像素（pix）”，每个像素都有与之对应的编号。



(a) $k=0$　　**(b)** $k=1$　　**(c)** $k=2$

图 1-1　从左至右分别为 HEALPix 在 $k=0,1,2$ 时对球面的划分情况

Figure 1-1　From left to right, the partitioning of the sphere by HEALPix at $k=0,1$ and 2.

1.2.2　多层级 HEALPix 索引技术

原始 HEALPix 只能以固定分辨率划分球面，之后有各种多分辨率的 HEALPix 解决方案被提出，例如多层级覆盖天区（multi-order coverage map，MOC; Fernique et al.2014; Fernique et al. 2015）、多分辨率 HEALPix（multi-resolution healpix，MRH; Youngren & Petty 2017）、mhealpy (Martinez-Castellanos et al., 2022) 等。MOC 是一种用于对指定天球区域做覆盖的方法，可以快速比较不同数据集之间的覆盖范围。MOC 方法使用预定义的单元格层次结构来存储地图覆盖范围，使其易于生成和使用。在最新的 MOC2.0 版本中，还加入了对时域数据的支持。MOC 是国际虚拟天文台联盟（IVOA）标准之一，已经被广泛应用。许允飞（2020）[28–30] 等人通过 MOC 建立的多层级覆盖天区四叉树（multi-order coverage tree，MOC-Tree）实现了对不规则区域星表的索引。Martinez-Castellanos 等人发布的 mhealpy 是一套多分辨率 HEALPix 及可视化的 Python 包，支持两种多分辨率模式，其中一种允许用户通过自定义函数选择当前区域的划分层级，但是受到可视化的限制，某个区域只能被一个层级所覆盖。而星表的切分不需要考虑可视化相关的限制，本文尝试允许多个层级的像素覆盖同一区域，实现对星表的均匀切分。

1.2.3 星表的交叉证认技术

为了从海量的观测数据中快速找到新出现的天体，国内外的研究人员都对此做了很多研究，其中一个关键步骤就是判断一个天体在之前的观测数据中是否存在，通常使用交叉证认来实现。

交叉证认可用于找出同一个源在多个不同观测时间或不同波段星表中的观测数据。常用的方法是基于位置的天体匹配。当两个来自不同星表的天体之间的球面距离小于一定阈值时，就可以认为是同一个天体。由于不同望远镜的观测精度不同，并且天体存在自行等原因，导致同一个天体在不同观测时期被不同的望远镜或者以不同的波段观测时所得到的坐标略有差异。一般来说，对于误差半径分别为 r_1、r_2 的两个星表，当两个天体的角距离 $d < 3\sqrt{r_1^2 + r_2^2}$ 的时候，就可以认为是同一天体 (许允飞, 2020)[50]。该方法仅考虑坐标和误差半径，但在实际情况中，不同波段的望远镜观测精度往往差异巨大。例如，地面小口径望远镜与高分辨率的空间望远镜对一片密集星场区域的观测结果做证认时，难免会在误差半径范围内有多个对应的天体候选，出现一对多或者多对多的情况。另外同一天体在不同波段的形态也不一定相同，例如类星体等天体在光学波段只能观测到天体中心等部分，而在射电波段则可以观测到中心以及两侧的喷流 (Fan et al., 2020)。因此在实际的交叉证认过程中，还需要考虑到天体本身的物理特征等参数。

还有一批专家提出了基于置信度的交叉证认算法，即将之前的交叉证认从“是”或“不是”二元结果转为计算一个该天体是同一天体的概率值，或置信度。置信度越高，则意味着两个天体是同一个天体的概率越大。例如，Sutherland 等人就提出了基于似然函数的交叉证认公式 (Sutherland & Saunders, 1992)，原理是比较两个假设的似然函数，似然函数可以用于计算给定数据下每个假设的概率，其中一个假设候选源是正确的，另一个假设候选源不是正确的。通过比较这两个假设的概率，可以确定哪个假设更有可能成立。这个公式能够在给定数据和先验分布的情况下，计算出一个天体在另一个星表中对应天体的概率，确定哪个候选项最有可能是正确的源。Budavári 等人则提出了基于贝叶斯方法的交叉证认算法 (Budavári & Szalay, 2008)，该方法使用贝叶斯统计学原理，将不同目录中的源位置误差、亮度等信息结合起来，计算每个源被正确识别的概率。

2007 年，图灵奖获得者吉姆格雷（Jim Gray）提出了能够将天球切分的条带算法，并在此基础上使用微软的 SQL Server 建立了一套交叉证认服务 OpenSkyQuery (Budavári et al., 2004; O'Mullane et al., 2005)，它允许用户在多个天文数据库中进行复杂的查询。OpenSkyQuery 支持多种数据格式和查询语言，并提供了一些高级功能，如交叉匹配、可视化和数据下载。该工具旨在为天文学家们提供一个方便、快速和易于使用的方式来访问和分析天文数据。

catsHTM (Soumagnac & Ofek, 2018) 是一个由 Soumagnac 等人提出的利用 HDF5 文件格式和 HTM 对天文星表做检索和交叉证认的工具，可以在几毫秒内以从几个角秒到度的分辨率执行高效的圆锥搜索。高丹（2008）等人使用 HTM 和数据

库实现了对数十万量级星表的检索与交叉证认，但没能解决划分边界附近天体的漏源问题。赵青（2011）针对 HEALPix 提出了基于位运算的快速邻域编码计算算法，解决了边界漏源问题，并使用 MapReduce 分布式并行计算模型应用于大规模天文交叉证认计算中。Du（2014）等人通过同时使用 HEALPix 和 HTM 以及数据库实现了对星表的交叉证认。Boch（2012）等人通过使用 KD-Tree 和 HEALPix 实现了在线交叉证认平台 CDS-Xmatch，该平台允许用户在网页上传自定义星表与平台自带或之前上传的星表进行交叉证认。但是 CDS-Xmatch 对用户的上传星表的数据量和交叉证认的运行时间都有限制，且国内用户的访问较为缓慢，较为不便。

1.3　主要研究内容

面对当前以及未来大视场时域巡天观测而产生的海量天体检索需求，本文尝试提出更加高效的星表检索、交叉证认与融合技术，并在此基础上构建一套多个计算节点并行计算的系统，探索分布式的星表检索与融合技术。本文包括以下几部分的研究内容：

1.3.1　多层级星表切分算法

为了提高检索的速度，本文尝试将大规模星表按照不同的区域切分成多个文件。基本原理是将天球按照规则切分成多个区域，并给每个区域一个编号。在建立索引时，根据星表中天体的坐标计算该天体所属的区域及区域编号。在检索时计算出待检索范围所覆盖的区域编号列表，根据这些编号找到该区域内的所有天体并使用球面距离公式精确计算角距离。

通过这种方式，能够将二维的天体坐标映射到一维的区域编号，实现了坐标的降维。同时，区域编号检索在数据库等场景下也有较高的索引效率。但是，该方法的主要问题在于检索的效率与划分的区域面积息息相关。若区域面积过大，则有大量天体拥有相同的编号，给后续精确计算带来困难。若单个区域面积过小，则一次检索覆盖到的区域数量太多，同样也会影响检索的效率。

传统的天球划分算法每个区域的面积都是相同的，由于天体在天球上的分布不均匀，如使用 HEALPix 直接切分得到的星表内天体的数量差异较大，影响检索时的效率。本文在 HEALPix 的基础上加以改进，提出了动态层级的切分算法，根据区域内的天体数量选择合适的切分层级，以实现对星表的均匀切分，以文件的形式进行存储。同时通过在每个分片内建立 KD-Tree 等树形数据结构索引的方式，实现对星表的高效检索。

1.3.2　星表的存取和融合优化

数据在内存中通常是以对象、结构体、数组、列表等形式连续或离散地存储于不同的区域供 CPU 访问和操作。当某些情况下需要将内存中的数据保存到文件或通过网络传输时，则需要将数据编码导出（也称之为序列化 serialization、编

组 marshalling）为计算机或人类可读的序列，反过程则称之为解码（或反序列化 deserialization、反编组 unmarshalling）(Kleppmann, 2017)。例如将内存中的天体以 CSV 格式保存到硬盘即是对星表序列化的过程，对 CSV 星表的读取则是反序列化过程。

不同观测计划产生的星表有着不同的格式，文本格式的星表占用空间大、读取速度慢。为了尽可能地提升性能，本文以开源的高性能序列化组件协议缓冲区（Protocol buffers，Protobuf）(Varda, 2008) 为基础，并针对星表这一类型的数据定义了对应的序列化格式，实现对星表数据的高效存取。以统一的二进制格式存取，避免了不同星表格式不统一带来的各种问题，以提高检索的速度与效率。

在暂现源搜寻时往往需要同时对多个不同波段的星表，或者多个不同观测时间星表进行检索。为了加快检索速度，可以考虑提前将多个星表合并，即对一个星表中的所有天体与其他星表交叉证认，找到对应天体并对各项参数加以融合，实现一次检索就可以获得多个星表中的数据。融合的过程中很大一部分的时间都消耗在硬盘的存取过程中，本文尝试通过增加缓存的形式提高星表融合时的速度。同时对星表访问的顺序做优化，以更高缓存效率的 Peano-Hilbert 曲线取代了原有的遍历曲线，以提高缓存的利用率。

1.3.3 分布式星表检索

分布式技术通过将多个计算节点利用网络等形式连接起来，形成一个整体，各个计算节点共同参与任务的计算，加快数据的处理速度。因此，分布式计算的前提是将一个复杂任务划分成多个独立的子任务，这些子任务可以并行地执行，并且相互之间没有依赖或影响。这样就可以将计算负载分散到多个计算节点上，从而提高计算效率和吞吐量。

在天文星表检索时，往往只需要在一个局部区域中检索。检索的过程不会修改已有的星表数据，同时运行多个检索任务时，不同的检索任务之间不会相互影响。这些特性使得星表适合使用多台节点分布式检索。同时，相较于单机检索，分布式检索还具有较高的可靠性和容错能力，通过设置多个副本备份节点，可以在某个节点出现故障时自动更换备用节点，检索过程不会受到影响。

为了提高检索的速度，本文尝试使用多个节点的分布式检索替代单机检索。本文研究了多节点之间的通信、星表在多节点中的分发方式、检索的调度方式等内容，并进行了性能测试。

1.4 文章结构

本文分为五大章节：

第一章　引言。主要介绍了本课题的研究背景，当前国内外主要的大规模巡天计划以及能够产生的数据量，并引出本文研究工作的重要性，还介绍了国内外星表的检索、交叉证认等研究的进展等内容。

第二章　星表存取及检索技术原理简介。介绍了天文星表的产生、常见格式及现有的划分和检索检索算法，以及本文工作中采用方法的相关技术原理。

第二章　星表高效检索的研究。介绍本文提出的动态层级的切分算法的具体过程以及应用于实际星表的测试结果。

第三章　星表数据的高效存取与融合。介绍针对大规模星表的高效存取以及星表融合时的效率问题的优化方法。

第四章　分布式星表数据检索。提出了一套分布式星表检索的方案，包括对星表的分发、检索调度问题做了详细的研究与讨论，并实际搭建了一套分布式检索系统做测试。

第五章　总结与展望。对本文的工作内容做了总结，并阐述了以后可以改进的内容。

第 2 章　星表存取及检索技术原理

星表的检索可以分为前期的预处理部分和检索部分。预处理即在检索前就将星表按照一定的规则建立索引，以节省检索的时间。预处理过程的好坏将直接决定检索过程是否高效。预处理过程通常包括将不同巡天计划星表格式的统一、星表按照区域切分、索引的建立，以及若使用分布式检索还需要将星表在各个分布式节点做分发等内容。检索部分则是根据待检索的坐标计算并读取预处理后的星表索引，并根据检索要求得到检索的结果。本章主要介绍本文的研究所用到的技术背景及其原理。

2.1　天文星表的产生与存储格式

光学观测所产生的数据一般可分为图像数据、星表数据、光谱数据等类型。以测光任务为例，望远镜拍摄得到的原始图像往往包含有传感器本底、背景天光、杂散光等各类干扰信息，不能够直接使用，需要经过多道流水线处理。原始数据一般需要经过仪器效应改正、位置定标与流量定标等步骤，得到单幅图像中的天体的位置与光度等信息。将同一天体在多次曝光或多个波段的图像数据合并经过反复迭代多次校准后，才能获得一个较为精确的天体信息。根据不同的望远镜设备与巡天计划，一般可以获得天体的坐标、坐标误差、自行、距离等天体测量学参数，以及天体在不同波段的亮度、形态等物理参数。这些数据按照统一的格式进行打包、导出，才能得到最终的星表数据。根据巡天计划的不同，一批星表数据内包含的天体数量从数十万至数十亿不等。在之后的巡天与暂现源搜寻的任务中，需要对这些已有的星表进行快速检索与交叉证认，从而快速发现变化的天体。

天文星表在计算机中存储和交换的格式按照数据的存储方式不同，主要可以分为文本格式和二进制格式两大类。文本格式中最常用的是逗号分隔值（comma-separated values，CSV）格式，还有 VOTable、XML 等格式。文本格式的优点是方便人类可直接阅读和修改，使用任意一款文本编辑器即可读取与手动编辑。以 CSV 格式为例，最常用的标准由 RFC 4180 定义 (Shafranovich, 2005)，格式的规范为：

(1) 每条记录位于单独的一行，使用换行符（CRLF）分隔。

(2) 通常令首行为表头，记录了每个字段相对应的名称。

(3) 表头或每条记录中，可能有一个或者多个字段，使用英文逗号 “,” 来分隔。

(4) 每个字段可以用双引号括起来，也可以不用双引号括起来。

(5) 包含换行符（CRLF）、双引号和逗号的字段应该用双引号引起来。

只要编辑后的格式符合规范，便可由程序自动解析。

文本格式缺点在于文本格式的存储效率低，空间浪费严重。例如，对于真空光速数值 299,792,458，使用二进制格式存储，只需要 32 位整数即可存下，占用 4 个字节。用 CSV 等文本格式则需要将该数字每一位都转换成 ASCII 码，得到二进制序列 “00110010 00111001 00111001 00110111 00111001 00110010 00110100 00110101 00111000” 。每个 ASCII 码占用 1 个字节，此时需要 9 个字节用于存储，严重浪费存储空间与传输带宽。另一个缺点是程序在读写文本格式的坐标等数字时需要做文本字符与整数或浮点数的相互转换，特别是对浮点数的转换时，不仅转换效率较低，且容易出现丢失精度的情况。

二进制格式中，天文中较为常用的是普适图像传输系统（flexible image transport system，FITS; Hanisch et al. 2001）。FITS 是由国际天文学联合会定义的信息交换格式，最初是为了传输图像而设计，之后扩展到了星表等更复杂的数据格式。与文本格式优缺点正好相反，二进制格式需要特定的软件或者软件包才能读写，由于数字以二进制存储，能够节省存储空间且计算机不需要转换即可直接读取到内存，存取的速度较快。对于内含数十亿天体的星表来说，选择二进制格式将获得较高的存取效率。

2.2　天体距离计算方法

当使用赤道坐标系来描述天体位置时，每一个天体都可以用赤经（RA）和赤纬（Dec）两个角度来确定其在天球上的位置。天球中两个点的距离则被定义为经过天球上两个点，且圆心在天球球心的大圆被该两点划分得到的劣弧所对应的角距离。

在计算距离时不能仅仅计算两个坐标之间的算术差或使用平面距离公式计算欧式距离，而是应该使用球面距离公式计算两个坐标之间的角距离。对于两个赤经和赤纬坐标为 (α_1, δ_1) 和 (α_2, δ_2) 的点来说，两点在球面上的角距离有多种公式可以计算 (樊东卫 等, 2019)，在计算机编程时常用的有 Haversine 公式：

$$d = 2arcsin\sqrt{sin^2(\frac{\delta_1 - \delta_2}{2}) + cos\delta_1 cos\delta_2 sin^2(\frac{\alpha_1 - \alpha_2}{2})} \tag{2-1}$$

或者 Great-Circle 公式：

$$d = arccos(sin\delta_1 sin\delta_2 + cos\delta_1 cos\delta_2 cos(\alpha_1 - \alpha_2)) \tag{2-2}$$

可以看出，无论哪种公式，都需要经过复杂的三角计算后才能得出两个天体之间的角距离。在计算机中，三角函数的计算开销远远大于算术运算，需要想办法减少三角函数计算的次数，以提升检索的速度。

对于一个包含数十亿天体的大规模星表来说，若不经过任何处理，从中搜寻特定坐标的天体，在最坏的情况下需要遍历整个星表，所消耗的时间难以承受，通过使用合适的星表索引算法，可以避免不必要的检索，提高效率。

2.3 HEALPix 的编号模式

HEALPix 有两种区域编号模式，分别为“RING”模式和“NESTED”模式。这两种模式在 $k = 0$，即在第 1 个层级时，12 个初始区域的编号是相同的，如图2-1所示。

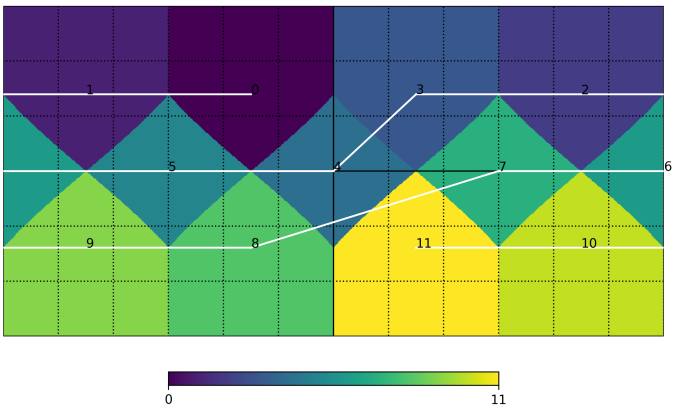


图 2-1　RING 模式和 NESTED 模式在第 1 层级的编号

Figure 2-1　The numbering of the first level in RING and NESTED modes.

但是到了更高的层级，两种编号模式就会产生区别。例如在 $k = 1$，即第 2 个层级时，第 1 层的一个初始区域会被切分为 4 个子区域。在 RING 模式下，天球被划分为一系列同心圆，每个圆上的像素数量相等。像素按照从北极到南极、从西到东的顺序依次编号，相连起来形成一个类似于螺旋线的结构，如图2-2所示。

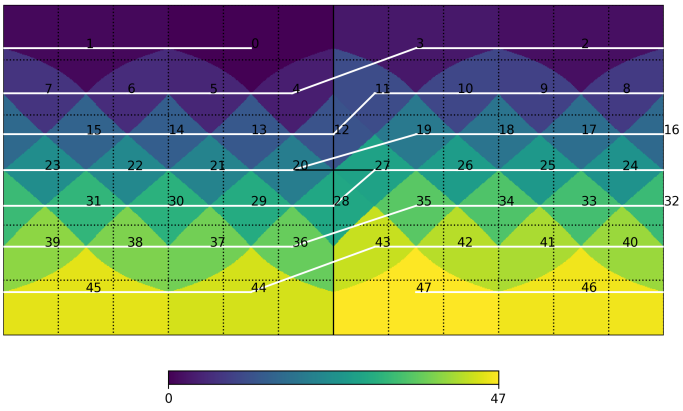


图 2-2　RING 模式在第 2 层级的编号

Figure 2-2　The numbering of the second level in RING mode.

NESTED 模式下，如图2-3所示，第 1 层的 0 号区域变成了第 2 层的 0、1、2、3 号区域，第 1 层的 1 号区域变成了第 2 层的 4、5、6、7 号区域，以此类推，将

第 i 层编号为 $npix_i$ 的区域继续划分到下一层，原来的区域划分四份，即可得到四个第 $i+1$ 层的区域，编号范围是 $[4\times npix_i, 4\times npix_i + 3]$。

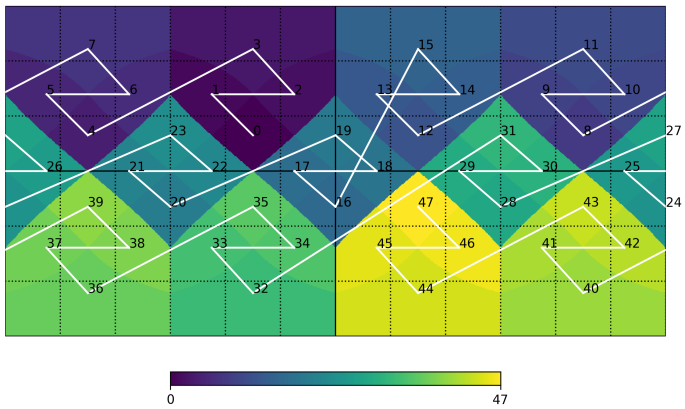


图 2-3　NESTED 模式在第 2 层级的编号

Figure 2-3　The numbering of the second level in NESTED mode.

若将 NESTED 模式的编号以二进制表示的话，任意一个像素在 NESTED 模式下的编号等于该区域在父层级下的编号左移 2 位加上该区域的位置编号（0 至 3）。以第一层的 5 号区域为例，如图2-4，5 的二进制为 101，若将该区域划分到第 2 层，则生成的 4 个新区域的编号方式是首先将 101 左移 2 位，得到 10100，再按照顺序分别加上 0 至 3，得到最终十进制的编号为 20 至 23。同理，第 3 层级的 16 个区域编号分别为 80 至 95。

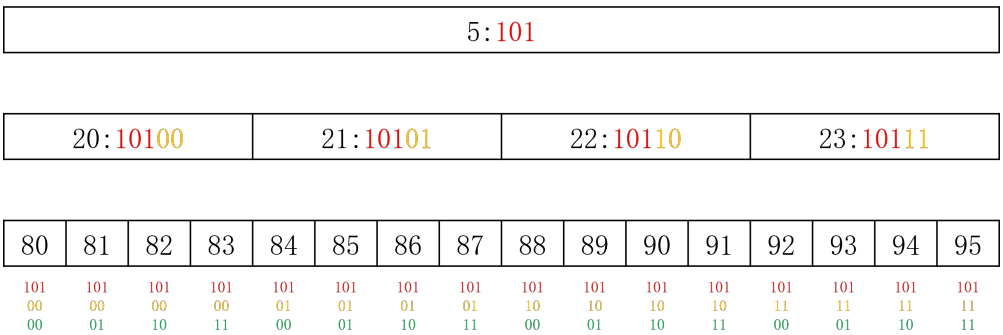


图 2-4　NESTED 模式的编号与二进制的关系

Figure 2-4　The relationship between the numbering in NESTED mode and binary code.

这个特性使得 NESTED 模式可以根据某一层级某个区域的编号很容易地推断出该区域在上一层级的编号，以及由该区域划分得到的所有子区域的编号。这个模式下两个不同层级的区域直接比较其编号的二进制前缀就可以计算得到两个区域的隶属关系，仍以图2-4为例，若有一个第 3 层编号为 82 的区域，和一个第 2 层编号为 20 的区域，则可以发现 82 的二进制编号为 1010011，20 的二进制

编号为 10100，两者有共同的前缀 10100，且位数相差 2 位。就可以推断出第 3 层的 82 区域是由第 2 层的 20 区域切分得到的。

2.4 KD-Tree 的索引原理

KD-Tree 是一种基于空间划分思想的数据结构，其中“K”代表了待索引数据的维度，对于拥有二维坐标的星表数据，则 K 的值为 2。在 KD-Tree 中，每个节点代表一个超矩形区域，该节点的左子树和右子树分别代表该超矩形区域内左半部分和右半部分的数据。KD-Tree 使用了类似于二叉搜索树的索引思想，对于 K 维的数据，在建立 KD-Tree 时，每次仅选择其中的一维作为划分的分割轴，计算数据中该维度的数值的中位数，将数据划分为左右两个部分。左边部分的数据中该维度的值全都小于中位数，右边部分的该维度数据全部大于中位数。之后对两部分数据选择另外的维度作为分割轴，递归划分，直到全部划分完毕。

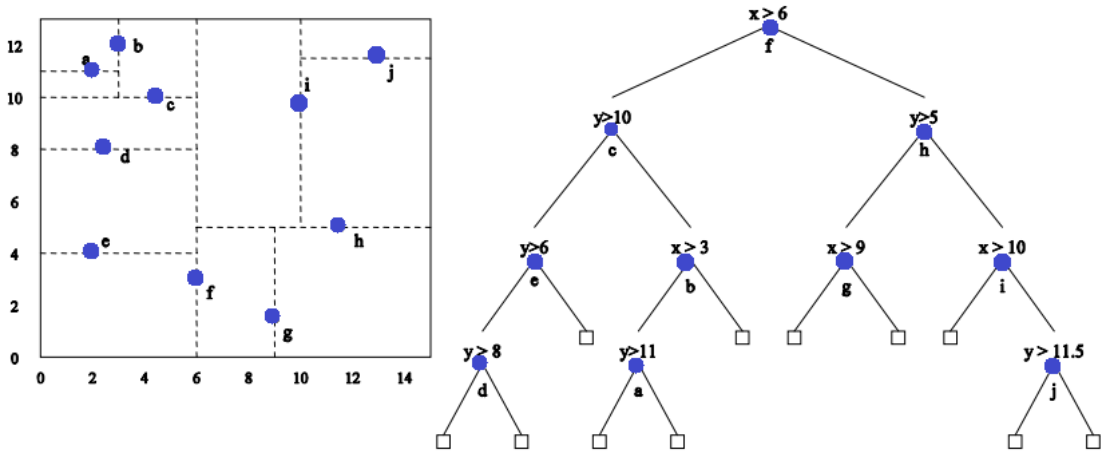


图 2-5 KD-Tree 的空间划分原理[1]

Figure 2-5 The space partitioning principle of KD-Tree.

如图2-5中的例子所示，若对 a、b、...、j 这 10 个点建立对应的 KD-Tree。首先以 x 轴为分割轴，寻找处于 x 轴中间位置的点，即 f 点。数据被分割为小于 f 的左半部分，包括 5 个点 a、b、c、d、e，和大于 f 的右半部分，包括 4 个点 g、h、i、j，以及 f 点本身。在 KD-Tree 中，将 f 点作为根节点，左右两个部分分别作为根节点的左子树和右子树。之后，再以 y 轴作为分割轴，用相同的过程分别对左子树和右子树的数据计算中位数并分隔成上下两部分。得到 c 点和 h 点分别作为左右子树的两个根节点。以此类推不断递归，最终建立起一个完整的 KD-Tree。

KD-Tree 的检索与建立过程类似，从根节点开始，根据待检索的坐标和节点的关系访问左右子树，直到叶子节点。再根据待检索坐标和叶子节点的距离或预先设定的距离为阈值，向上回溯。判断父节点或兄弟节点的距离是否小于阈值，并不断更新检索结果，最终完成检索。

[1]https://www.researchgate.net/figure/Basic-KD-Tree-each-Circle-Represents-a-Feature-and-Each-Area-is-Used-to-Limit-the _fig3_302594789

KD-Tree 的优点是检索速度较快，检索时每一步都可以让数据的规模减半，最坏时间复杂度只有 $O(N^{1-\frac{1}{K}})$。缺点是建立与检索需要在内存中进行，因此对于大规模数据集，KD-Tree 可能会占用大量内存。且建好后任何新增或删除数据都会破坏 KD-Tree 的结构。如果需要更新数据，则需要重新构建整个 KD-Tree。而大规模星表数据量动辄数 TB，一方面，现有计算机的内存往往不足够全部容纳这些数据。另一方面，即使少许的修改或增删天体，都需要重新建立 KD-Tree 索引，代价很高。

直接将完整的大规模星表用来建立 KD-Tree 并不可行，但是可以将 KD-Tree 作为切分后的分片内检索方式。这种方式既提升了每个星表分片内数据的检索速度，又能够在需要小部分更新星表数据时，只需要重新建立个别 KD-Tree 即可实现星表的更新。

2.5　Protocol Buffers 及其实现原理介绍

Protocol Buffers 是一种与语言无关、平台无关的可扩展机制，用于序列化结构的数据。Protobuf 能够将内存中数据按照事先约定的格式在程序和程序之间，或者通过网络传递，也可以写入到磁盘保存。功能上类似于可扩展标记语言（extensible markup language，XML）和 JavaScript 对象简谱（JavaScript Object Notation，JSON）。但是与之不同的是，Protobuf 需要事先约定好一套协议，称之为接口描述语言（interface description language，IDL），读写时需要持有相同的 IDL。数据按照 IDL 规定的格式以二进制的形式存储，并使用多种压缩算法，避免了数字与字符串之间的相互转换。Protobuf 当前支持 C++、JAVA、Python、Go 等主流的语言。

Protobuf 将定义好的 IDL 称之为消息（message），一个消息内可以包含多个成员。每个成员可以是多种类型，例如整型（int32、int64 等）、浮点型（float、double 等）、字符串型（string、bytes）或者是枚举类型（Enumerations），甚至一个消息类型也可以作为成员嵌套（Nested Types）在另一个消息内。

在定义好消息的格式之后，Protobuf 提供了一套支持多种语言的生成工具，可以根据消息的内容生成对应语言的序列化与反序列化代码。在编程时导入生成好的代码，即可实现对该消息的读写。

值得一提的是，Protobuf 为了尽可能地减少传输的数据量，采取了多种数据压缩与优化措施。除了上文中提到的以二进制保存数据之外，还引入了可变长度的 Varints 编码方式。对于整数类型的数据来说，在内存中的占用空间大小往往是固定的，与该数的数值大小无关，常见的有 32 位整数类型和 64 位整数类型。32 位整数类型可以保存 $(-2^{31}-1, 2^{31})$ 之间的所有整数。但是如果有大量的数据数值远远小于 2^{32} 的话，采用固定长度的整数保存将会浪费大量的空间。而使用 Varints 编码可以做到按照整数数值的大小占用不同的空间。数值的绝对值越小，所占用的存储空间也就越小。例如，数字 100 如果用 32 位整数类型表示，需要 4 个字节，如果用 Varint 编码表示，则只需要 1 个字节，减少了 75% 的空间消耗。

2.6 MapReduce 架构分布式检索原理

为了解决处理大规模数据时传统的计算方法遇到的性能瓶颈，谷歌提出了分布式计算领域中赫赫有名的计算架构“MapReduce”(Dean & Ghemawat, 2008)，该架构将计算过程分解为“Map”和“Reduce”两个过程，如图2-6所示。

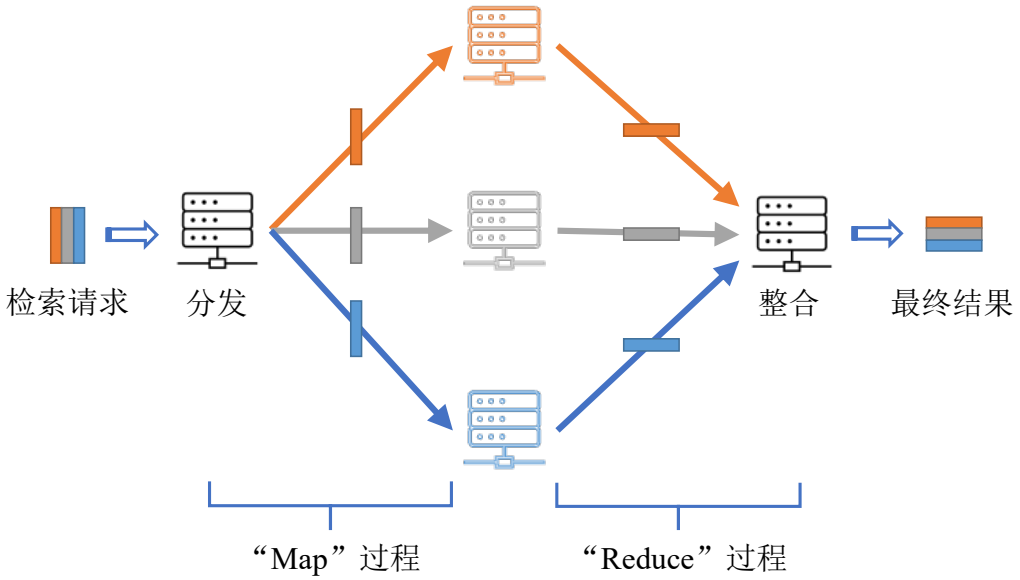


图 2-6 MapReduce 分布式检索过程示意图

Figure 2-6 Schematic diagram of MapReduce distributed retrieval process.

一个复杂的计算任务首先通过事先定义好的“Map（映射）”函数被分解成为多个子任务，并分发到不同的节点上进行并行计算。在各个节点计算完成之后，由“Reduce（归约）”函数对各个节点的数据做整合，得到该复杂任务的最终结果。由于 MapReduce 框架可以很好地利用集群中多台机器的计算能力，因此它已经成为了大规模数据处理领域中最流行和最常用的分布式方法之一。

2.7 小结

本章主要介绍了本文研究过程中所用到的一些算法和技术，包括星表格式、角距离计算、HEALPix 的编号模式、KD-Tree 的实现原理、开源序列库和 MapReduce 分布式架构的介绍。这些内容作为本文后序研究工作的技术背景铺垫，本文将通过对这些算法与技术加以改进和创新，并融入到星表的检索过程中，以实现星表的高效检索。

第 3 章　星表数据高效检索研究

大规模星表的数据量往往巨大，难以直接检索。对于星表来说，有以下两种常见的基于坐标的检索操作。

(1) 最近邻检索，即给定坐标，检索星表中与该坐标最为接近的天体。

(2) 锥形检索，即给定坐标和距离阈值，检索星表中所有距离该坐标小于阈值的天体。

这两种类型的检索操作都有较大的空间局部性，即仅需要检索一个坐标点附近区域内的天体，并不需要遍历整个星表。可以考虑将星表按照天球区域切分，同一区域的天体保存到同一文件中。检索时只需要读取局部区域的文件，而不用检索整个星表。本章主要讨论如何在原有天球划分算法的基础上，通过改进切分的方式实现更高的检索效率。

3.1　星表切分的策略

一个好的切分策略应该保证切分出的星表文件中的天体数量大致相当。若单个文件内天体数量太多，则检索一个很小的区域时也需要读取文件中的大量天体，消耗时间较长。若数量太少，则每次检索需要读取大量的星表文件。一方面，这些文件可能离散分布在硬盘的各个位置。若星表由机械硬盘存储，在读取前机械硬盘磁头需要移动到相应位置，大量随机分布的文件将会产生大量的寻道开销，严重拖慢读取速度。固态硬盘虽然无需寻道，但是对小文件的随机读写速度仍然远远慢于顺序读写速度。另一方面，考虑每个文件内只有 1 个天体的极端情况，此时甚至会退化成顺序检索，造成效率的下降。综上分析，单个星表文件数据量与星表的检索效率应该是一个倒“U”型曲线。让切分后的每个星表文件天体数量完全一致较为困难，但是只要对单个星表的天体数量上下限做出限制，使其能够保持在最优的数据量附近，就可以实现较高的星表检索效率。

原始的 HEALPix 算法虽然可以将天球均匀切分成多个等面积大小的区域，但是其层级是固定的且需要根据不同星表手动指定层级。而星表中的天体在天球中的分布并不是均匀的。例如，以 Gaia DR2 星表为例，如图3-1所示，其在银河系中心方向上的天体密度远远大于垂直与银道面方向与反银心方向。因此，若直接按照 HEALPix 的固定层级做星表切分，难免会造成部分区域内的星表数据量过大，部分区域星表数据量较小，难以高效检索。且不同星表的天体数量不同，切分时使用的层级也应该不同，选择层级时，过大过小都会影响检索的效率。

若要实现星表的高效检索，就需要在 HEALPix 的基础上加以改进，使之能够根据天球上天体的密度不同，动态选择合适的层级。

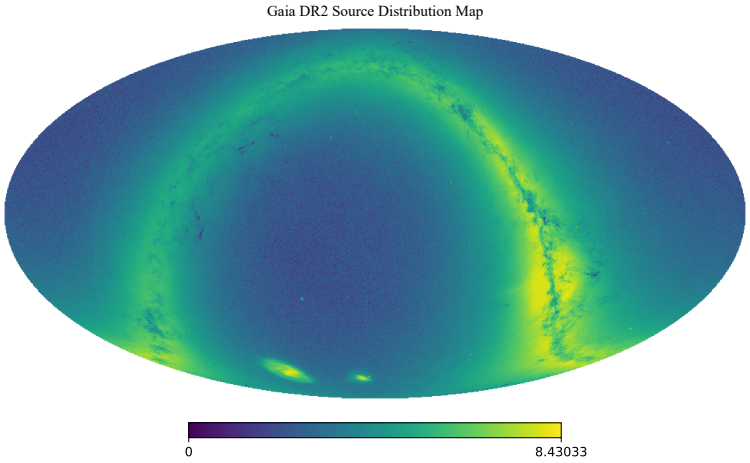


图 3-1　Gaia DR2 星表的天体分布图

Figure 3-1　Gaia DR2 source distribution map.

3.2　多层级共存下的 HEALPix 编号

由于 HEALPix 不同层级区域的编号均是从 0 开始，多个层级共存时编号会产生冲突，因此一个区域在编号时必须要注明所在的层级，在多层级共存环境下使用较为不便。本文采用了 MOC 中的 UNIQ 编码方式 (Fernique et al., 2014)，赋予不同层级的区域一个唯一的编号。在 UNIQ 的编码方式下，层次为 k，编号为 $npix$ 的区域的 UNIQ 编号可通过如下公式计算：

$$UNIQ = 4 \times 4^k + npix \quad (0 <= npix <= 12 \times 4^k) \tag{3-1}$$

逆运算为：

$$k = \lfloor log_2(UNIQ/4)/2 \rfloor \tag{3-2}$$

$$npix = UNIQ - 4^{k+1} \tag{3-3}$$

采用此种编号，HEALPix 在 $k = 0$ 的编号范围变成了 [4，15]，$k = 1$ 编号范围为 [16，63]，第 k 层的区域编号范围为 $[4^{k+1}, 4^{k+2} - 1]$，不同层级之间的编号不会产生冲突，且编号范围是连续的。

3.3　基于四叉树的天体索引

为了能够根据天体的区域密度动态选择合适的层级，本文使用四叉树为不同层级之间的区域建立联系。四叉树是一种基于空间递归分解思想的分层数据结构 (de Berg et al., 1997)，与二叉树类似。参考二叉树的递归定义，四叉树的递归定义如定义3.1所示。

定义 3.1. 四叉树是一棵空树，或者是一棵由一个根节点和四棵互不相交的子树组成的非空树；四棵子树同样都是四叉树。

满四叉树可以定义为：

定义 3.2. 若一棵高度为 k 的四叉树，其有 $(4^k - 1)/3$ 个节点，则称此树为高度为 k 的满四叉树。

四叉树的父子节点是一分为四的关系，这与 HEALPix 对相邻层级之间区域的划分是一致的。HEALPix 初始时将天球切分为 12 块等面积的区域，此时若建立 12 棵满四叉树，将每棵树的根节点分别编号为 $[0, 11]$，该编号与 HEALPix 在 $k = 0$ 时的 NESTED 编号相对应。如图3-2所示，将 HEALPix 的一个 k 层级区域继续划分，形成四个 $k + 1$ 层级的子区域后，每个子区域都与四叉树 $k + 1$ 层的一个节点相对应。每个子区域还可以继续细分为四个更小的区域，直到最高的层级。这个过程可以一直递归进行，直到达到所需的层级。由此，HEALPix 的不同层级区域之间可以实现与满四叉树的节点一一对应。

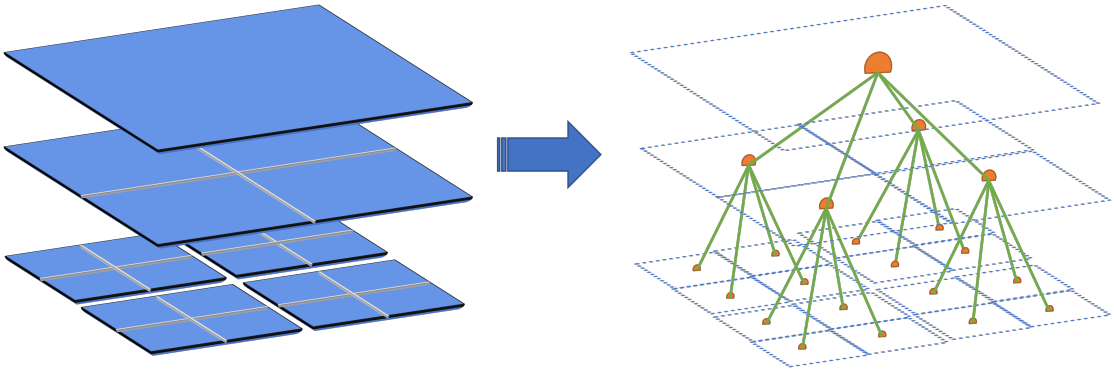


图 3-2　不同层级的相同区域可以用四叉树的不同节点来表示

Figure 3-2　The same region at different levels can be represented by different nodes of a quadtree.

若对四叉树使用和 HEALPix 一样的 NESTED 模式下的编号方式，则可以由四叉树某个节点的编号计算得到其父节点的编号，以及所有子节点编号（若存在子节点），并根据公式3-1得出与之对应的 UNIQ 编号。利用这个特性，将四叉树存储在一个连续的数组中，四叉树节点的 UNIQ 编号即为其所在的数组下标。

四叉树的节点内可以保存对应区域的一些信息。例如，将某一层级下某个区域内的天体数量，保存到对应的四叉树节点中。当若添加或删除天体信息时，只需自底向上，从叶子节点开始，依次将父节点的天体数量更新为四个子节点的天体数量之和。这样就可以很方便地获取到某一层级某一区域内的天体数量信息，并根据该信息做下一步的切分决策。

3.4　星表的动态层级切分算法

星表切分前首先需要确定一个切分阈值 T，表示切分出来的单个文件内天体数量为 T 时，检索的效率最高，将该值作为切分出的单个子星表的天体数量的预期值。根据前面的分析，这个阈值的取值与计算机的硬件性能有关。具体来

说，星表从硬盘读入到内存，再由 CPU 对内存中的星表做计算的过程中将会受到硬盘的读写速度、内存的访存速度与延迟、CPU 的计算速度等多种硬件的性能的限制。在阈值较小时，由于单个文件数据量较小，每次需要检索的文件总量较多，检索速度主要受到硬盘随机读写能力的影响。而在阈值较大时，单个文件数据量较大，检索速度将会受到硬盘的连续读写性能和内存访存性能等影响。这些影响因素导致不同配置的计算机所适合的最优切分阈值也是不一样的。

由于实际星表检索时，只需要检索某个坐标附近区域的星表文件，因此为了获得最优的阈值，可以在对大规模星表切分之前取该星表一个局部区域作为测试样例。即按照不同的阈值对这一小部分星表做切分，得到切分后的文件。并根据实际检索的需求，对这些测试样例做读取与检索的性能测试。根据不同阈值下的读取和检索的性能，得到当前硬件环境下最优的阈值大小。以此作为完整的大规模星表切分时的阈值。

另外，还需要确定一个层级上限 k_{max}，因为 HEALPix 层级越大，划分的区域数量也越多，若不对最大层级加以限制，则容易出现某些天区天体密度过高，导致程序选择的层级过大，计算过程中出现数据溢出的现象。k_{max} 同时也作为四叉树的最大高度。

根据 HEALPix 的编号原理，32 位整数能够支持的层级最大为 13 级，即 NSIDE=8192，此时 HEALPix 将天球划分成了 805,306,368 区域，每个区域的张角为 25.8 角秒。64 位整数能够支持的最大层级为 29 级，即 NSIDE=2^{29}=536,870,912，此时 HEALPix 将天球划分成了 3.46×10^{18} 个区域，每个区域的张角为 $3.93\times10^{(-4)}$ 角秒 (Górski et al., 2005)[765]。

对于现阶段绝大部分星表来说，13 级的区域分辨率已经足够高，故将 k_{max} 设置为 13 并使用 32 位整数保存天体数量是一个较为合理的选择。

3.4.1 四叉树的初始化与星表的读取

在确定好阈值与 k_{max} 之后，建立起 12 棵四叉树，每棵树共有 k_{max} 层，每棵树的第 i 层有 4^i 个节点，12 棵树共 $4^{k_{max}+2}-4$ 个节点。每个节点内保存的信息有该节点所对应的 HEALPix 层级、NSIDE 编号和 UNIQ 编号，以及该节点所对应的区域内的天体数量 tot，并将 tot 值初始化为 0。

之后，将待切分的大规模星表读入内存，并按照如下步骤建立起四叉树的索引。

(1) 按照格式解析星表，得到每个天体的坐标（ra，dec）。

(2) 根据天体的坐标计算在 k_{max} 层级下天体所在的 HEALPix 区域编号

(3) 根据编号找到对应四叉树中的节点，将该节点中 tot 值加 1。

(4) 根据 NESTED 编号规则，将当前编号右移 2 位得到父节点编号，将父节点内的 tot 值加 1，并重复此步骤，直到根节点。

读取过程如算法1所示。

由于大规模星表大小 N 往往会远远大于可用内存容量 M，而程序运行时单

算法 1 星表的读取

Require: 星表文件 C, 12 棵 4 叉树 $TREE$,k_max

function TreeElementAdd($Npix$)　▷ Npix 所对应四叉树节点 tot 值加一
　$Current\_K \Leftarrow k\_max$　▷ 当前节点层次为最大层次
　while $Npix \neq 0$ **do**　▷ 从叶节点依次向上遍历到根节点
　　$Tree[Current\_K][Npix].tot+ = 1$
　　$Npix >>= 2$　▷ 利用 HEALPix 特性计算父节点
　　$Current\_K+ = 1$
　end while
end function
function ReadCatalog($filename$)　▷ 星表的读取
　while True **do**
　　while $(C \neq EMPTY)$ & $(Memory \neq FULL)$ **do**　▷ 星表未读完且内存未达到上限
　　$Line \Leftarrow$ ReadLine(C)　▷ 从星表中读取一行目标
　　$Ra, Dec \Leftarrow$ ParseCoordinatesFromLine($Line$)　▷ 从目标中获得坐标 Ra，Dec
　　$Npix \Leftarrow$PixCalculate(Ra, Dec, k_max)　▷ 计算最大层次时的的像素编号 npix
　　SaveCatalogToMemory($Line, npix$)　▷ 将目标保存到 npix 对应的列表中
　　TreeElementAdd($Npix$)　▷ 更新四叉树中的信息
　　end while
　　for $i = 0 \rightarrow 11$ **do**▷ 当无法继续读取时，遍历 12 棵四叉树并开始切分
　　SetSplitDir($TempDir$)　▷ 设置切分到临时星表目录
　　SplitCatalog($0, i$)　▷ 从层级为 0，编号为 i 的节点开始切分
　　end for
　　if $(C == EMPTY)$ **then**
　　Return　▷ 如果星表读完完毕则退出
　　end if
　　if $(Memory == FULL)$ **then**
　　ClearMemory　▷ 如果内存达到上限则清空内存
　　end if
　end while
end function

次所能读取的星表又不能大于 M。因此程序在读取 M 数据量的星表后，就需要将内存中的局部星表做切分，并清空内存，继续读取后续的星表。这样就需要将星表分成 $\lceil N/M \rceil$ 块局部星表，每块大小都小于等于 M。

3.4.2 局部星表的切分

经过上述的星表读取过程后，四叉树中的每个节点中的 *tot* 变量都保存了对应区域的天体数量。从每棵树的树根开始，按照深度优先的顺序遍历四叉树的每个节点。通过节点中父节点与子节点的 *tot* 数量与阈值 T 的关系，确定合适的切分层级。

此时，节点之间有如下关系：

若某个层级为 k 的非叶节点 P 的子节点为 $4 \times P + i, i \in [0, 4)$，则

$$tot_P = \sum tot_{4 \times P + i} \tag{3-1}$$

可以考虑如下情况：在某个层级 K 下，某个区域内的天体数量为 N。若 N 远大于 T，则该区域的最优切分层级不应是 K，而应该以比 K 更大的层级切分。此时应将该区域继续向下一层级遍历，即将该区域一分为四，四个子区域与四叉树中 $K+1$ 层级的四个子节点相对应。但是由于天体分布的不均匀，四个子节点中天体数量 *tot* 也不尽相同，可能出现如下几种情况：

(1) 父节点的天体数量和 4 个子节点天体数量均大于 T，则继续递归遍历四个子节点的子节点。

(2) 父节点的天体数量大于 T，存在天体数量小于 T 的子节点，这种情况的处理方式见下文。

(3) 已到达 k_{max} 层，则将当前节点所对应区域内的所有天体切分到以该区域 UNIQ 编号的文件中，并停止向下遍历。

对于情况（2），以图3-3中的情况为例，假设图3-3中 $k+1$ 层编号为 $4 \times p$，$4 \times p + 3$ 的子区域内的天体数量少于阈值，其余区域大于阈值。此时，又可以分为以下 3 种处理方式，本文分别按照其特性，命名为“激进”、“保守”、“中立”方式，如下文所示。

- “激进”方式：对 total 值大于 T 的子节点继续递归遍历其子节点。对小于 T 的子节点，则将该子节点所在区域的天体切分到对应文件中，并停止该子节点的向下递归，即在图3-3的例子中将 $4 \times p$ 和 $4 \times p + 3$ 的节点各自切分到两个文件中。

该方式已经存在于 mhealpy 中，在本文中按照其特点将其命名为“激进”方式。

- “保守”方式：只要 4 个子节点中存在 1 个 total 值小于 T 的节点，就将父节点对应区域的天体切分到对应文件中，并停止遍历其子节点，即将图3-3的例子中编号为 p 的父节点切分。

● “中立”方式：“中立”方式是本文的原创算法和主要工作之一，具体过程如算法2所示。分别访问父节点的 4 个子节点，若某个子节点的 *tot* 值大于 T，则先不切分，而是对该子节点继续向下递归遍历，并得到一个返回值 r，代表了相应区域有 r 个天体被切分，将该值返回至父节点中。若某个子节点 *tot* 值小于 T，则不再遍历其子节点，也不切分，而是将 *tot* 的值作为 r 返回至父节点。

父节点收到各个子节点的返回值后计算剩余未被切分的天体数目，若总数大于 T，则以父节点的名义将这些天体切分到对应文件中，并返回总数。若小于 T，则将已切分的数量返回至更上一层节点。依次类推，直到节点内天体总数大于 T，或到达根节点时将天体切分到文件中。图3-3的例子中假设 $4\times p+1$ 和 $4\times p+2$ 区域已全被切分。

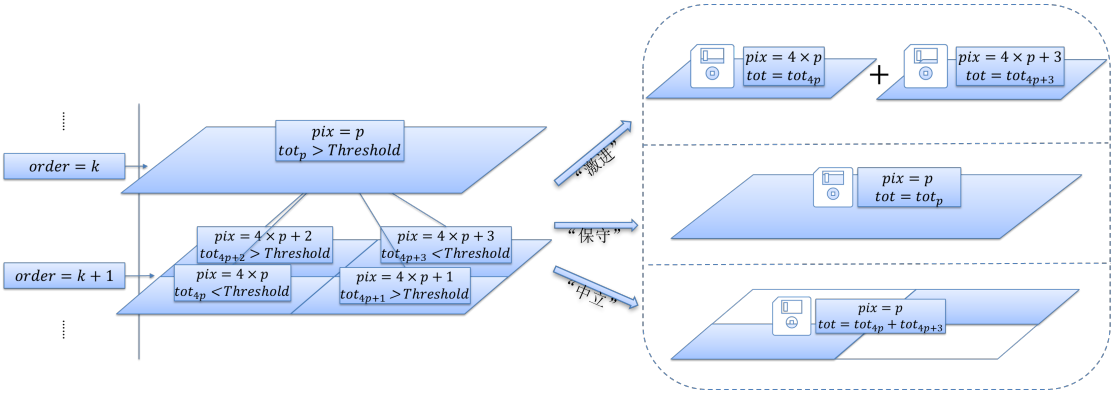


图 3-3　星表切分情况 2 时的 3 种处理方式及分别得到的星表编号和天体数量

Figure 3-3　Three processing methods for case 2 of catalog splitting are shown.

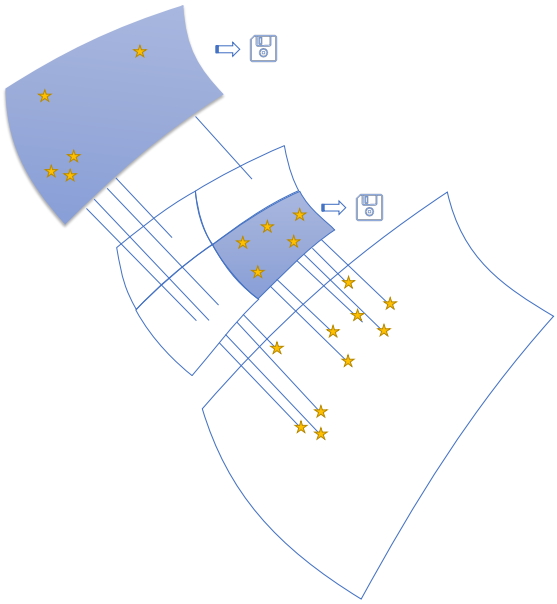


图 3-4　“中立”模式下多个层级共存时的星表切分情况

Figure 3-4　Catalog splitting in the presence of multiple levels in neutral mode.

不同处理方式之间的优缺点比较如表3-1所示。经过对比不难看出，本文所

算法 2 “中立”模式下星表的切分

Require: 12 棵 4 叉树 $Tree$，k_max，阈值 T;

1: **function** SPLITCATALOG($Current\_k, Current\_npix$) ▷ 判断某个节点是否需要切分，返回该节点已被切分的天体数量
2: $totalWrite \Leftarrow 0$ ▷ 该节点下被切分的天体数目，初始为 0
3: $CurrentNum \Leftarrow Tree[Current\_k][Current\_npix].total$ ▷ 当前节点天体数量
4: **if** $(CurrentNum > T)\&\&(Current\_k < k\_max)$ **then**
5: **for** $i = 0 \rightarrow 3$ **do**▷ 父节点大于阈值且非最大层次，对子节点递归判断
6: totalWrite+= SPLITCATALOG($Current\_k+1, (Current\_npix << 2)+i$)
7: **end for**
8: **end if**
9: **if** $(CurrentNum - totalWrite > T)||(Current\_k == 0)$ **then** ▷ 当前节点剩余未切分天体数量大于阈值，或为根节点，则切分该节点
10: $UNIQ_id \Leftarrow$CALUNIQ($Current\_k, Current\_npix$)
11: SAVETOFILE($Current\_k, Current\_npix, UNIQ\_id$)
12: RETURN (CurrentNum)
13: **else**
14: RETURN (totalWrite)
15: **end if**
16: **end function**

提出的“中立”切分方式在理论上可以切分出最为均衡的星表文件，适合作为实际应用时的首选切分方式。

表 3-1　不同切分方式的优缺点对比

Table 3-1　Comparison of the advantages and disadvantages of different splitting methods.

类型	优点	缺点
保守	切分后单个星表文件的数目皆大于阈值 T（特例情况时除外）。	当 4 个子节点中有一个小于阈值 T 时，则将整个父节点一并切分输出，极端情况下切分出的文件可能会很大。
中立	通过选择仅切分大于阈值的区域，并通过增加返回值的方式将被切分的目标数量传递给父节点汇总。这种方式使得切分后星表内天体的数目会在 $[T, 4\times T]$ 之间（特例情况时除外），大小较为均衡。	某个天区可能同时会被多个层级所覆盖，如图3-4所示，但该天区的目标只存在于层级最高的那个文件中。即虽然重叠，但不会重复。这一特点对星表的检索无影响，但在可视化时无法将所有区域绘制在一副天球图像上。
激进	切分后单个星表文件的天体数量严格小于阈值 T，此时阈值 T 的大小即为切分出的星表文件的数目上限。	可能会切分出大量小文件，相同阈值 T 下切出的文件总量也是三种方式最多的。

在切分完成之后，清空内存中的星表，若星表未读取完毕，则继续读取星表，并重复上述切分步骤，直至所有星表读取完毕，完成星表的局部切分。

3.4.3　多个局部切分星表的归并

“中立”模式对星表切分得到 $\lceil N/M \rceil$ 个分块之后，需要将各个分块的星表做归并，得到对原始星表完整的切分结果。由于每个分块的切分过程都是互相独立的，直接将不同分块内相同 UNIQ 编号的文件合并在一起，会使得单个文件内天体的数量超出阈值的要求，违背了前文所述的星表切分规则。

此时每个分块内的星表都按照所属区域被切分到了不同的文件中，所以在归并时，可以只读取分块中某个特定区域的星表文件，避免了不必要的内存占用。

归并的流程同样需要建立 12 棵 k_{max} 层的四叉树，将树中每个节点的 tot 值置为 0。与局部切分不同的是，归并从每棵树的根节点开始，按照深度优先的顺序遍历树的各个子节点。检查每个局部切分的星表里是否有该节点对应的文件存在。若存在，则读取到内存，使用与局部切分时一样的过程更新四叉树。

对于局部切分星表与完整切分的星表内天体的关系，可以得到如下推论：

推论 3.1. *某个最终切分层级为 k 的的星表文件中的每个天体在局部切分星表中所属的层级为 $k_1, k_2, ...k_n$，则 $k \geq \max(k_1, k_2, ...k_n)$。*

即天体从局部星表归并到完整星表后，其所在归属的星表的层级只会越来越大。这个特性决定了，在自根节点向下遍历时，子节点中未被读取的星表不会对父节点产生影响。

算法 3 星表的合并

Require: 临时星表目录 $TempDir$

1: **function** CATALOGMERGE($Current\_k, Current\_npix$)▷ 深度优先遍历各个四叉树节点
2: $UNIQ_id \Leftarrow$CALUNIQ($Current\_k, Current\_npix$)
3: **if** FILEEXISTS($UNIQ\_id$,$TempDir$) **then** ▷ 若临时星表目录中存在对应 UNIQ 编号的星表则读取
4: READCATALOG($UNIQ\_id$)
5: **end if**
6: **if** $(Memory \neq FULL)||(Current\_k == k\_max)$ **then**
7: **for** $i = 0 \rightarrow 3$ **do**
8: CATALOGMERGE($Current\_k + 1, (Current\_npix << 2) + i$) ▷ 遍历四个子节点
9: **end for**
10: **end if**
11: $CurrentNum \Leftarrow Tree[Current\_k][Current\_npix].total$
12: **if** $(CurrentNum > T)||(Current\_k == 0)$ **then**
13: SAVETOFILE($Current\_k, Current\_npix, UNIQ\_id$)
14: **end if**
15: **end function**

之后依次递归遍历该节点的 4 个子节点，根据深度优先遍历的特点，当遍历到 k_{max} 层级的节点时，以该节点本身所属的星表文件，以及该节点所有父节点所属的星表文件，都已经被读取到内存中。此时内存中该 k_{max} 层级区域的天体数量，和四叉树中该节点的 tot 值，均与完整星表中该区域的天体数量一致。

此时需要判断该节点的数量是否大于阈值 T，若大于，则切分并返回到父节点，若小于则直接返回到父节点。此时切分得到的星表文件即为归并之后的星表文件。当父节点的 4 个子节点都遍历完并返回到当前节点时，此时每个子节点的天体数量不会大于 T，父节点区域内的天体数量不会大于 $4 \times T$。以相同的过程判断父节点判断是否需要切分，并返回至上一层，此过程不断递归进行，直到所有 12 棵四叉树均遍历一遍。保证了切分得到的星表天体数量在 $[T, 4 \times T]$ 之间，实现对局部星表的归并，得到的切分结果即是对完整星表的切分结果。

3.5　分片内星表的检索

在将星表按照上述切分算法切分之后，每个星表文件以 UNIQ 编号作为文件名，代表了其所覆盖区域。在检索时，可将待检索区域转换为对应的 UNIQ 编号列表并读取对应的文件。为了加快检索速度，可以先建立对应的 KD-Tree 再做检索。这种检索方式可以避免读取整个星表，能够提高检索效率。根据不同检索方式，有如下不同的星表文件读取方法。

3.5.1　检索特定区域的所有天体

由于“中立”模式下多重覆盖的特性，一个区域可能会被多个星表文件所覆盖，但区域中的某个天体只会保存在能覆盖到它的文件里面层级最大的那个文件内。此时，只需要从四叉树中该天体所在的 k_{max} 层级的节点开始，不断向父节点递归，直到第一个存在星表文件的节点，该天体就保存在这个文件中。

如果想要在切分后星表中检索到某个区域，例如第 k 层，HEALPix 编号为 $npix$ 的所覆盖区域内全部天体。首先从该节点出发，向根节点查找，找到第一个存在星表子集的节点，并读取。由于该节点还覆盖有其他相邻的区域，可以根据不同任务的需要选择将星表全部读取或者仅读取待检索区域内的天体。若 $k < k_{max}$，则还需要依次遍历该节点的所有子节点，并将子节点中存在对应的星表子集全部读取。

若待检索区域是一个多边形，则可以将该多边形区域导入到 MOC 中，得到对应区域的 UNIQ 列表。再对该 UNIQ 列表中的每个区域进行上述的区域查询，并汇总，即可得到最终的读取结果。

3.5.2　对坐标的检索

若给定一个坐标和阈值，要求检索以坐标为圆心，阈值为半径内的所有目标，首先使用 HEALPix 自带的 query_disc 函数计算得到在最大层次下被该区域覆盖得到的像素编号列表，将列表中的每一个像素执行上述区域查询，并对已经

读取的星表做好标记避免重复读取。

而在最近邻检索时，待检索坐标与最近邻天体可能不在 k_{max} 层级下同一个区域中。所以需要读取与该区域相邻接区域内的所有天体。读取后通过 Haversine 公式计算与待检索坐标的距离，找到与待检索目标距离最近的天体。

3.6 星表的更新

若需要对切分后的星表新增天体，首先计算该天体在 k_{max} 层级下的编号，并找到四叉树对应的叶子节点。从该叶子节点开始，递归向上访问其父节点，直到到达第一个存在星表文件的节点，将新增的天体插入到该节点对应的文件中。删除天体也是以同样方法找到该天体所在的文件，删除对应天体。

若某星表文件在新增或删除天体后，文件内的天体数量范围还在 $[T, 4\times T]$ 之间，则无需做其他改动。若超过这个阈值范围，则会使得星表文件不在符合“中立”模式下的数量规则，此时应该将该文件视作一个“局部星表”，与剩余的星表文件用前文所述的步骤做星表的归并，以保证单个星表内天体数量的均衡。

3.7 对 LAMOST 星表在不同切分模式下的测试

本实验使用 LAMOST DR7 星表作为测试数据，分别对测试星表做“激进”、“中立”、“保守”模式的动态切分和以固定层级的切分。测试动态切分使用的切分阈值为 1000，将切分得到的星表按照体积排序后生成分布曲线。得到的结果如图3-5和表3-2所示。

表 3-2　不同切分方式的得到的星表数据对比

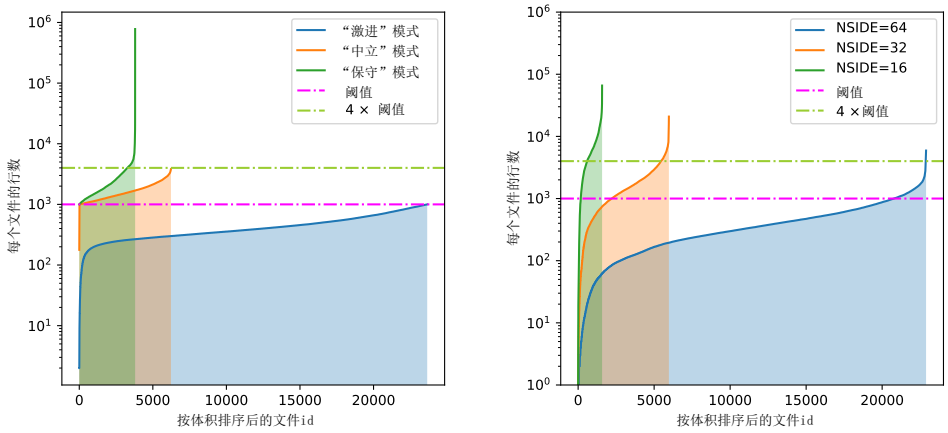
Table 3-2　Comparison of the catalog data obtained by different splitting methods.

类型	星表文件总数	平均行数	行数最大值	行数最小值	标准差	变异系数
动态层级，保守	3798	2746.49	778747	1002	12965.52	4.7207
动态层级，中立	6233	1673.54	3818	178	565.57	0.3379
动态层级，激进	23649	441.08	1000	2	199.05	0.4512
固定层级，k=4	1578	6610.39	65792	1	6016.63	0.9101
固定层级，k=5	5972	1746.68	20835	1	1608.43	0.9208
固定层级，k=6	22891	455.69	5917	1	421.35	0.9246

为了比较不同不同切分方式的均匀程度，本文采用了变异系数（coefficient of variation，C_v）来作为衡量星表文件均匀程度的依据。C_v 是一个无量纲量，定义为数据的标准差 σ 与平均值 $\bar{x}$ 之比，如公式3-1所示。

$$C_v = \frac{\sigma}{\bar{x}} \tag{3-1}$$

由于方差或标准差只有在两堆数据的平均值一样的情况下才可以相互比较，而变异系数则相当于将不同平均值数据的标准差归一化，使得不同平均值的数



(a) 相同阈值下动态切分的三种方式做切分　　**(b) 使用相近的三个层级对星表做静态切分**

图 3-5　将相同星表按照不同方式切分并排序后的大小曲线

Figure 3-5　Size curve of the same catalog after being splitted and sorted in different ways.

据也能比较数据的离散程度。变异系数越低，则意味着数据的离散程度越低，分布越均匀。

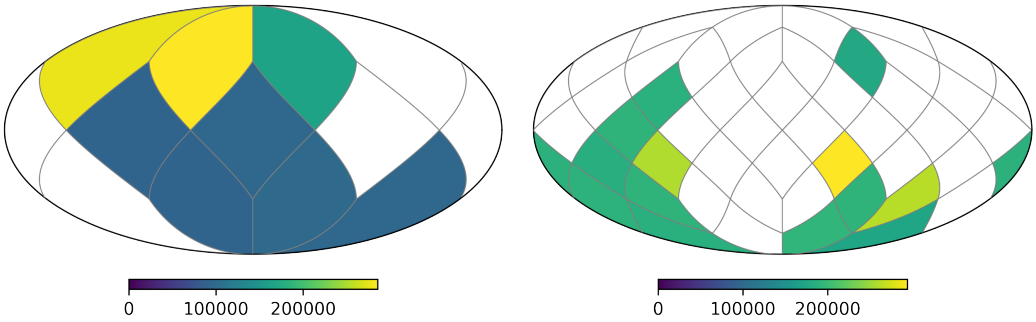
可以看出，“激进”方式切分出的星表的天体数量均小于阈值，大部分集中在 [200,1000] 之间。“中立”方式切分出的星表的天体数量绝大部分集中在 [1000,4000] 之间。由于南天区某些区域天体过少，导致极少量的文件小于阈值。同时，“中立”模式下切分出的星表的变异系数也是所有模式中最低的，说明了星表的离散程度小于其他模式。“保守”方式切分出的星表均大于 1000，特别是一部分星表的天体数量过大，最大约 10^6，其变异系数也是各个方式中最大的。三个固定层级切分的结果变异系数相差不大，皆大于“中立”和“激进”模式，小于“保守”模式。

切分结果与预期相符，证明了“中立”方式切分出的星表较其他两种切分方式和固定切分更为均匀。

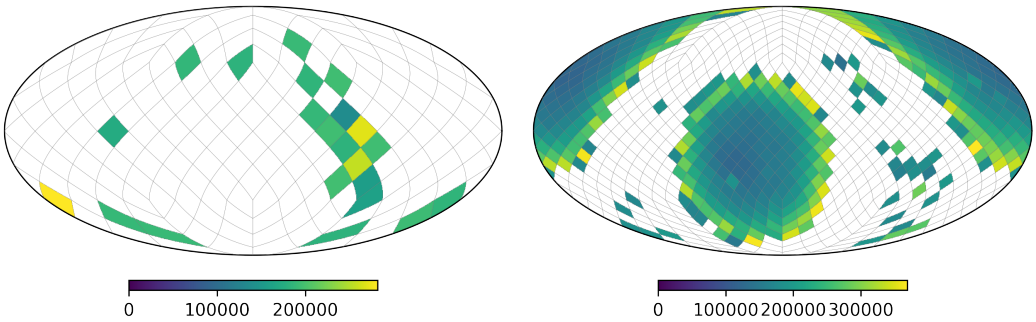
3.8　对 Gaia 星表的动态层级切分测试

Gaia 的 DR2 星表是由欧洲空间局（ESA）于 2018 年 4 月发布的一份星表，其中包含了 1,692,919,135 条记录的恒星和其他天体数据 (Gaia Collaboration et al., 2018)。Gaia DR2 星表的分布如图3-1所示，颜色代表天体的相对密度。可以从图中清晰地看出银河系以及大小麦哲伦星云的轮廓，均为天体分布较为密集的区域。使用本文介绍的动态切分算法的“中立”模式对该星表做切分，为了避免切分的层级过多难以展示，阈值设置为 100,000，得到的切分结果如3-6所示，颜色对应的坐标代表某个区域内的天体数量。其中一个局部区域的放大图如3-7所示。

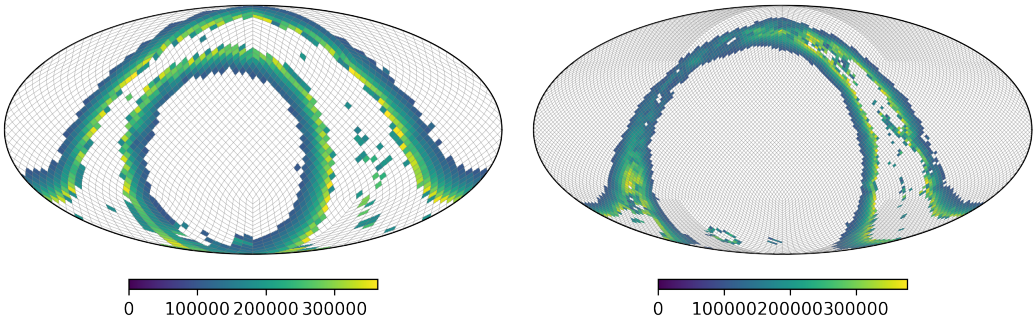
通过对 Gaia 星表的切分实验，证明了本文的切分算法确实可以完成对 TB 级别大规模星表的切分。从图中可以看出，对于天体分布较密集的银心以及银道



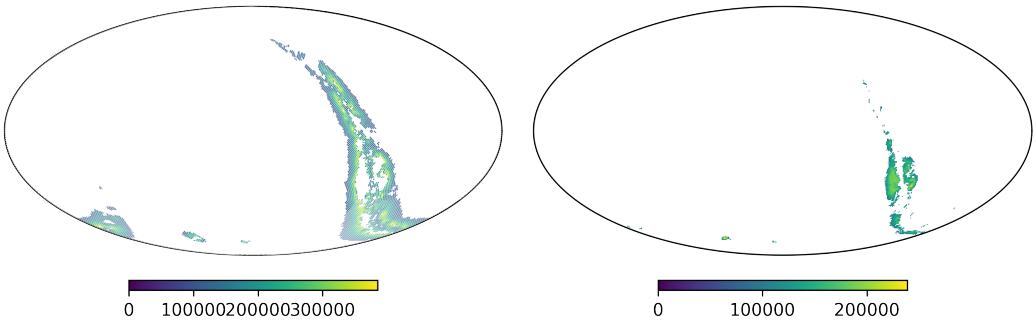
(a) $k = 0$ **(b)** $k = 1$



(c) $k = 2$ **(d)** $k = 3$



(e) $k = 4$ **(f)** $k = 5$



(g) $k = 6$ **(h)** $k = 7$

图 3-6　对 Gaia DR2 星表的动态层级切分结果

Figure 3-6　Dynamic hierarchical partitioning results for the Gaia DR2 catalog.

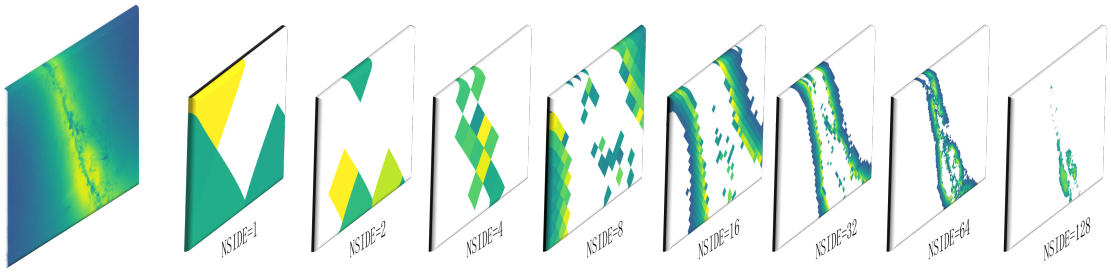


图 3-7　Gaia DR2 星表切分结果的局部放大

Figure 3-7　Local magnification of the partitioning results for the Gaia DR2 catalog.

面的部分，算法选择了以较高的层级做切分。对于垂直于银道面的区域，则以较低的层级切分。几乎每个区域内的天体数量都在 100,000 至 400,000 之间，符合切分的规则。

3.9　小结

本章主要介绍了本文提出的基于 HEALPix 的动态层级切分的原理和具体过程。在“中立”模式下能够实现将切分出的星表内天体数量限制在一倍阈值至四倍阈值之间。为了能够在有限内存的情况下完成切分，还将算法过程分为了局部切分与归并两个过程。并介绍了与之匹配的检索方法。

经过实际 LAMOST 星表和 Gaia 星表的测试，证明了该算法切分得到的星表均匀程度最高，且能够完成对 TB 规模的星表切分，为后续的星表的高性能检索打下了基础。

第 4 章　星表数据的高效存取与融合

暂现源搜寻以及多个大规模星表融合等任务需要在短时间内对大天区面积的星表做检索，此时星表存取性能的好坏将会在很大程序上影响检索的速度。本章将讨论如何对星表存取和融合时的过程做改进与优化，实现星表检索和融合性能的提升。

4.1　天文星表的序列化

一个好的序列化方法可以在以下几个方面起作用：

一方面，不同观测计划发布的星表格式往往不统一，存在 CSV、FITS 或机构自定义的格式，程序难以同时支持所有的星表格式。即使同为 CSV 格式，不同的星表在表头的命名方面也没有统一的规范，导致星表读取时往往需要手动指定坐标所在的列号。通过使用序列化工具，能够将所有星表以相同格式存储，消除不同星表格式之间的差异，降低星表检索时的复杂程度。

另一方面，在实际的切分和归并过程中，星表的每个天体都会被读取两次，写入两次，数十亿天体将会产生大量硬盘 I/O 并消耗大量的时间。一方面，由于星表的坐标等数据都是以浮点数的形式存储在内存中，如果简单地以 CSV 等文本格式存储，需要解析每行的各个表项以及对浮点数与字符串之间相互转换，由此产生的计算量和耗时也不容小觑。另一方面，在切分时，程序只关心天体的坐标，其他的物理参数则不需要考虑。通过使用序列化工具，直接以二进制保存天体坐标，可以避免上述转换与计算的过程。

最后，对于切分之后的星表先建立 KD-Tree 再检索，可以获得相较于顺序检索更高的检索效率。但是建立 KD-Tree 的过程需要消耗较多时间，若能将 KD-Tree 在开始检索前事先建立好，并使用序列化工具保存到硬盘中，在检索时便可直接读取 KD-Tree，能够提升检索的速度。

4.1.1　对星表数据的序列化

本文定义了星表在 Protobuf 中的传输格式，包括一个 “Catalog” 项用于储存星表信息，内容如表4-1所示，其中包括星表文件的文件名、UNIQ 编号、星表头信息、总行数信息等，同时内部有任意数量的 “CatalogLine” 项，用于存储天体信息，内容如表4-2所示，其中包括了天体的坐标、区域编号，分别以浮点数和整数进行存取。而天体的其他信息则保存为字符串类型，以减少对数据的解析时间。

表 4-1　使用 Protocol buffers 定义的 Catalog 项

Table 4-1　Catalog items defined with Protocol buffers.

字段名	类型	说明
name	string	星表的名字
uniq	int32	星表内天体所在区域的 UNIQ 编号
total_lines	int32	星表的天体数量
header	string	星表的表头信息
line	repeated CatalogLine	星表中的天体，repeated 表示数量可为 1 至多个

表 4-2　使用 Protocol buffers 定义的 CatalogLine 项

Table 4-2　CatalogLine items defined with Protocol buffers.

字段名	类型	说明
pix	int32	该天体所在区域的编号
ra	double	该天体的赤经
dec	double	该天体的赤纬
line	string	该天体的其他信息

4.1.2　对 KD-Tree 的序列化

KD-Tree 作为二叉树的一种形式，所有的非叶节点都有指向左右两个节点的指针。由于在切分之前已经设定好了切分阈值 T，且“中立”模式下切分的特点就决定了切分得到的星表大小不会大于 4×T。此时，KD-Tree 的节点数量也不会超过 4×T。因此，可以在内存中申请一个大小为 4×T 的数组，数组每个元素的大小为单个 KD-Tree 节点所占空间，用于存放 KD-Tree。将根节点存放于数组的 0 号区域，每个节点以所在的数组下标为访问地址。在检索时，先找到该数组的 0 号节点，根据 0 号节点内的信息决定下一步检索的节点数组下标，并根据该下标找到对应的数组区域，实现了 KD-Tree 的检索。

此时，可以将整个数组以 Protobuf 的格式做序列化，并以文件的形式保存到磁盘中。需要检索该区域时，则读取该文件，恢复原有的数组，即可获得一棵事先建立好的 KD-Tree，实现了对 KD-Tree 的保存与恢复。

4.2　星表融合时的缓存策略研究

位于 HEALPix 区域边缘的天体的误差半径可能会覆盖到相邻区域，所以在计算某一个区域的距离时还需要读取与该区域相邻的几个区域内的天体，此时可以通过选择合适的遍历顺序和增加缓存以提升效率。

根据之前的切分情况，以星表文件作为建立 KD-Tree 的单位，若某次检索的区域覆盖了多个星表文件，则需要在内存中建立多个 KD-Tree。此时可以将之前检索时建立的 KD-Tree 缓存到内存中，供之后的检索使用，避免重复读取硬盘，提高效率。如图4-1所示，当需要对 a 区域的天体做检索时，需要读取 a 以及相

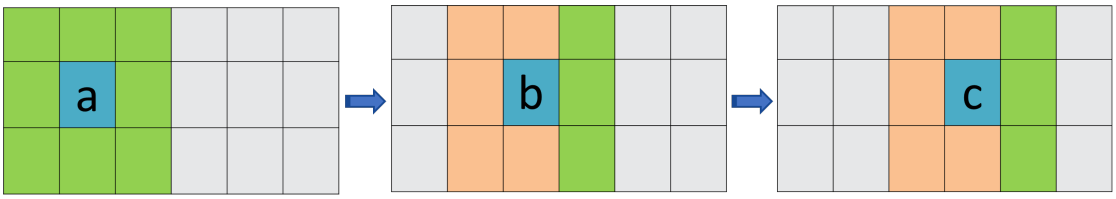


图 4-1　星表缓存示意图

Figure 4-1　Catalog cache diagram.

邻近的 8 个图中绿色的区域，并将这 9 个区域缓存在内存中。之后在对 b 区域做检索时，同样需要读取 b 及相邻近的区域，但只需要新读取 3 个图中绿色的区域即可，c 区域同理。

而受到内存容量的限制，缓存的 KD-Tree 数量不可能无限大，特别是在同时运行多个任务时。当内存中的 KD-Tree 数量达到上限，此时需要从已缓存的 KD-Tree 中选取一个用作替换。一个理想的缓存替换算法将会选择在未来最不可能被使用的那个 KD-Tree 作为替换，但是在实际中由于无法预先得知未来的检索请求，因此该算法是不存在的，所以只能依靠过去的检索请求来预测哪一个 KD-Tree 应该被替换。因此，缓存替换算法的优劣也会对检索的效率产生影响。

现在已经有多种缓存替换算法，例如先进先出（FIFO），或者最近最久未使用（LRU）等等。先进先出算法是指每次在缓存 KD-Tree 时都保存好该文件加入到缓存时的时间，当缓存容量到达上限时，选择最早被缓存的那个作为替换。最近最久未使用是指当使用到某个 KD-Tree 时，保存使用的时间。当需要替换时，选择最久未被使用过的那个作为替换。

上述替换算法早已被提出，并在计算机等领域已经被广泛使用。但是在天文星表检索领域的性能如何则不得而知，本文将通过实验探究不同缓存算法在星表证认时的性能差异以及最优缓存的大小。

4.3　星表高效遍历顺序

在对 HEALPix 按照 NESTED 模式编号顺序遍历时，路径在空间表现为“Z”型曲线的形式 (Reinecke & Hivon, 2015)，如图4-2所示。

该顺序可以高效地为平面或空间中的一组点集构建四叉树或者八叉树 (Bern et al., 1999; Warren & Salmon, 1993)。而“Z”型曲线特点是会在不同区域的边界处有比较大的转折，即会跳转到一个不相邻的区域中。在对星表遍历时往往会导致之前的缓存大量失效，降低了效率。以图4-3中的左图为例，图中按照编号顺序访问天球区域，存在多处较大的跳转，例如从图中编号为 63 的区域继续访问编号为 64 的区域时，两个区域相距较远。在程序实际检索时，之前缓存过的编号 63 区域附近的星表便会无法发挥作用。

为了克服这个缺点，本文尝试在对星表遍历时使用 Peano-Hilbert 曲线 (Reinecke & Hivon, 2015; Schäfer, 2005) 代替原有的“Z”型曲线。Peano-Hilbert 曲线

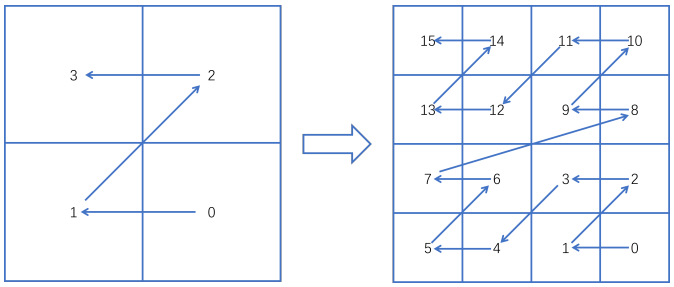
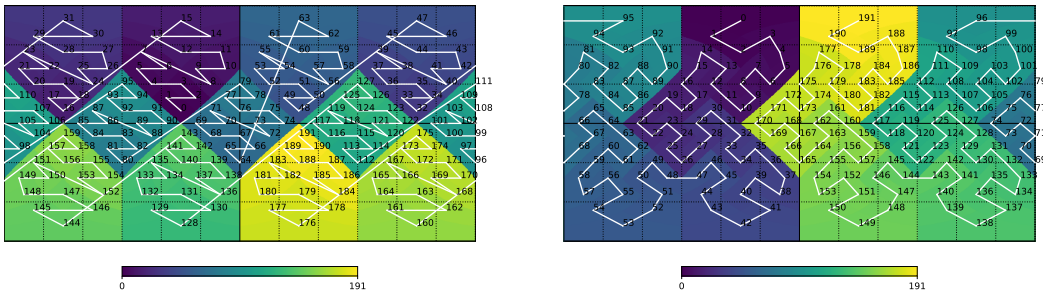


图 4-2　按照 NESTED 模式编号顺序遍历得到“Z”型曲线

Figure 4-2　The ”Z” curve is obtained by traversing the NESTED pattern numbering sequence.

是一种基于分形原理的空间填充曲线，该曲线将 HEALPix 区域重新编号，实现了对区域的连续访问。该曲线最大的特点是具有连续性，即该曲线上相邻编号的区域在空间上也是相邻的，按照该曲线的编号遍历星表区域时，能够充分利用已经读取过区域的缓存。在 HEALPix 层级为 $k = 2$ 时的 Peano-Hilbert 曲线与“Z”型曲线对比如图4-3所示。



(a) NESTED 模式编号顺序　　**(b) Peano-Hilbert 曲线编号顺序**

图 4-3　按 NESTED 模式编号顺序和 Peano-Hilbert 曲线编号顺序遍历星表示意图

Figure 4-3　Traverse catalog representation in order of NESTED mode number sequence and Peano-Hilbert curve number sequence.

与“Z”型曲线类似，Peano-Hilbert 曲线也是由低层级向高层级转化的过程中不断递归产生的。Peano-Hilbert 曲线在面对由 1 个共同的父层级切分得到的 4 个区域时，存在 8 种访问顺序。如图4-4所示，分别为（0，1，3，2），（1，3，2，0），（3，2，0，1），（2，0，1，3），（0，2，3，1），（1，0，2，3），（3，1，0，2），（2，3，1，0）。本文称之为 8 种基本的遍历顺序，完整的 Peano-Hilbert 曲线由这 8 种顺序递归组合而成。

若将基本遍历顺序中的每个区域作为父区域，切分成更小的 4 个子区域。此时，父区域与父区域、子区域与子区域之间仍然需要满足 Peano-Hilbert 曲线的连续性，8 个基本顺序的递归生成曲线如图4-5所示。

以 8 种顺序中的 0 号顺序为例，如图4-5a所示，划分前的访问顺序是 0 区域–>1 区域–>3 区域–>2 区域，将这 4 个区域继续划分到下一层级后得到 16 个

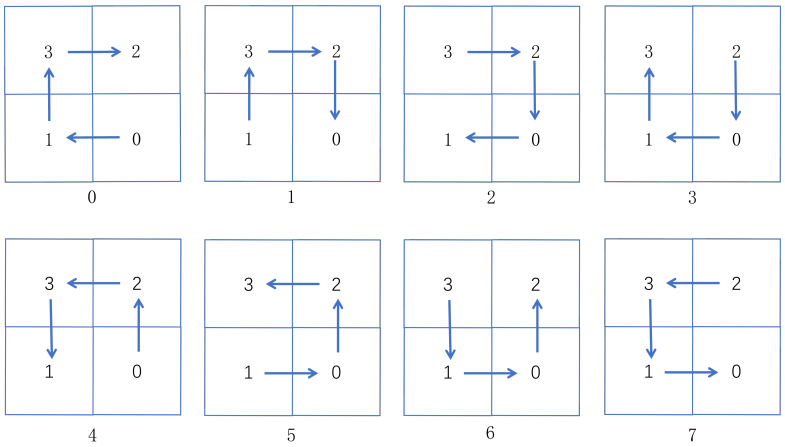
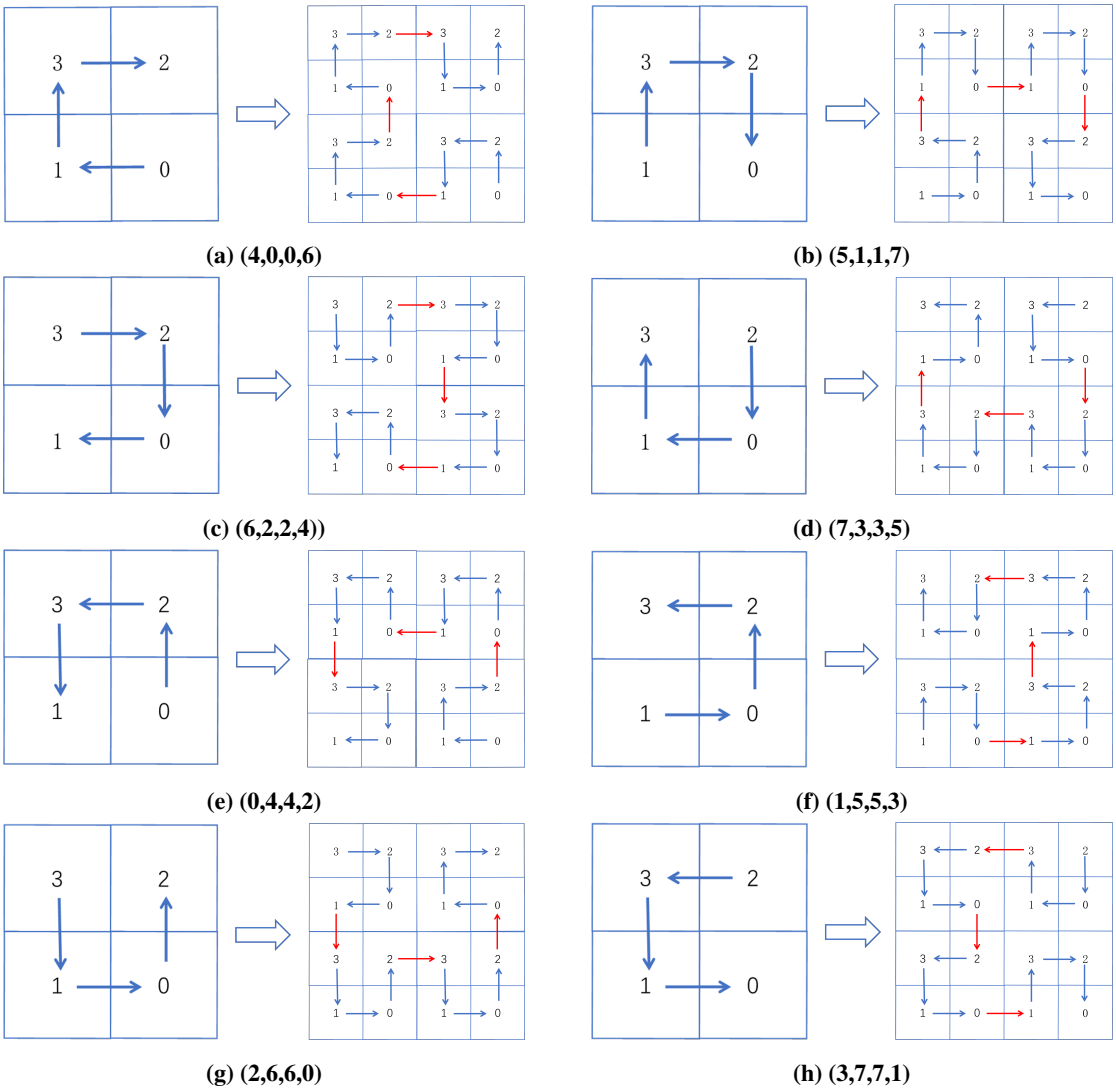


图 4-4　Peano-Hilbert 曲线的 8 种基本遍历顺序

Figure 4-4　8 basic traversal orders of Peano-Hilbert curve.

区域后，不能改变原有父区域的访问顺序，且不同父区域之间能够首尾相连。此时，原有的 0 号区域变为了基本顺序中的顺序 4，1 号区域变为了基本顺序中的顺序 0，3 号区域变为了基本顺序中的顺序 0，2 号区域变为了基本顺序中的顺序 6，并将各个父区域之间连接起来，得到4-5a的右图。该过程可以用矩阵（4, 0, 0, 6）表示，同理可以得到剩余 7 种顺序的变换矩阵分别为：（5，1，1，7），（6，2，2，4），（7，3，3，5），（0，4，4，2），（1，5，5，3），（2，6，6，0），（3，7，7，1）。这个过程可以一直递归进行，直到目标层级，最终得到完整的 Peano-Hilbert 曲线。

在 $k=0$ 层级时，也需要对区域重新编号以满足 Peano-Hilbert 曲线的连续特性，编号后的访问顺序如图4-6所示，Peano-Hilbert 编号和 HEALPix 原始编号之间的转换关系的方法如4-7所示。



(a) (4,0,0,6)　(b) (5,1,1,7)

(c) (6,2,2,4))　(d) (7,3,3,5)

(e) (0,4,4,2)　(f) (1,5,5,3)

(g) (2,6,6,0)　(h) (3,7,7,1)

图 4-5　Peano-Hilbert 曲线在层级变化后的顺序

Figure 4-5　Order of Peano-Hilbert curve after hierarchical changes.

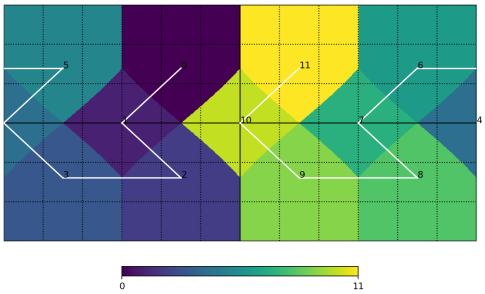


图 4-6　Peano-Hilbert 曲线在 $k=0$ 时的编号顺序

Figure 4-6　Numbering order of Peano-Hilbert curve when $k=0$.

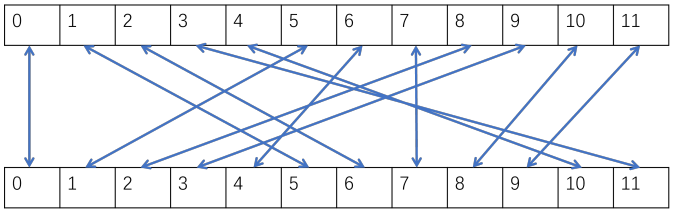


图 4-7　对第 $k=0$ 层级区域的重新编号，上面为原始 HEALPix 中的编号，下面是与之对应的 Peano-Hilbert 编号

Figure 4-7　Re-numbering of the $k=0$ hierarchical level region, the above is the original numbering of HEALPix curve, below is the numbering of Peano-Hilbert curve.

4.4　实验测试

为了测试本文提出的上述存取优化方法的实际效果，本文使用了实际的星表分别做了不同的对比测试。

4.4.1　不同格式不同阈值星表的切分测试

本文将 LAMOST DR7 星表分别在“中立”模式下按照不同的阈值切分为 CSV 格式和 Protobuf 格式，不同格式切分后的文件数据量如表4-3和表4-4所示。

由于星表在 Protobuf 格式内每个天体的存储策略是天体的二进制浮点坐标 + 字符串格式的原始星表，因此数据量相较于 CSV 会略有增加。在稍微牺牲数据量的情况下，换取读取的速度提升是完全值得的。

4.4.2　不同格式的星表检索速度测试

检索速度测试使用的运行平台为国家天文科学数据中心提供的云虚拟机，配置如下：

- CPU：Intel(R) Xeon(R) CPU E5-2690 4 Core @ 2294.686 MHz
- 内存：8 GB
- 硬盘：100GB ，I/O 速度：82.5 MB/s
- 操作系统：Ubuntu 20.04.4 LTS
- 编程语言：C++、Python

本文将 LAMOST DR7 星表分别在“中立”模式下按照不同的阈值切分为 CSV 格式和 Protobuf 格式，并分别对其全部读取，测试读取的时间如图4-8.a 所示。

可以看出，随着切分阈值的增加，两种格式的星表读取时间都逐渐减小直至稳定。Protobuf 格式在文件体积略大于 CSV 格式的情况下，各个阈值下的读取速度均快于 CSV 格式。尤其是当阈值大于 1000 之后的星表读取速度比 CSV 格式快了超过 50%，基本达到了硬盘连续读取 I/O 的速度。

之后，本文又对这两种格式的星表进行了自交叉证认（Self-Matching）的测试，即先将未切分星表全部读入到内存并按 HEALPix 编号排序作为源星表，按

表 4-3　CSV 格式在不同阈值下切分后文件和原始星表的数据量对比

Table 4-3　Comparison of the volume of the splitted file and the original catalog in CSV format under different thresholds.

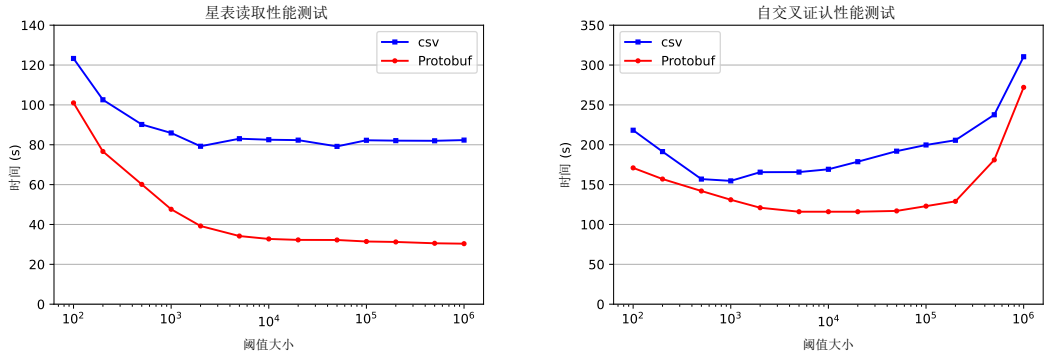
阈值	单个文件平均数据量（单位：字节）	文件总数	总数据量（单位：字节）
100	59,804	59,804	3,406,675,650
200	112,524	30,275	3,406,675,650
500	287,192	11,862	3,406,675,650
1000	546,555	6,233	3,406,675,650
2000	1,136,316	2,998	3,406,675,650
5000	2,695,155	1,264	3,406,675,650
10000	5,584,714	610	3,406,675,650
20000	10,679,234	319	3,406,675,650
50000	25,422,953	134	3,406,675,650
100000	50,098,171	68	3,406,675,650
200000	89,649,359	38	3,406,675,650
500000	200,392,685	17	3,406,675,650
1000000	262,051,973	13	3,406,675,650
原始星表	3,396,245,328	1	3,396,245,328

注：原始 LAMOST 星表未能严格遵守 RFC4180 标准，换行符是 \n，程序切分后星表换行符是 \r\n，导致文件总数据量略有增加。

表 4-4　Protobuf 格式在不同阈值下切分后文件的数据量对比

Table 4-4　Comparison of the volume of the splitted file and the original catalog in Protobuf format under different thresholds.

阈值	单个文件平均数据量（单位：字节）	文件总数	总数据量（单位：字节）
100	61,672	59,804	3,688,260,196
200	121,753	30,275	3,686,069,170
500	310,631	11,862	3,684,699,641
1000	591,093	6,233	3,684,282,465
2000	1,228,834	2,998	3,684,043,860
5000	2,914,491	1,264	3,683,916,770
10000	6,039,130	610	3,683,869,172
20000	11,548,113	319	3,683,847,953
50000	27,491,301	134	3,683,834,379
100000	54,173,964	68	3,683,829,559
200000	96,942,826	38	3,683,827,378
500000	216,695,639	17	3,683,825,864
1000000	283,371,198	13	3,683,825,578



(a) 不同阈值下星表读取速度曲线　　(b) 不同阈值下 Self-Matching 速度曲线

图 4-8　对切分后 Protobuf 格式与 CSV 格式的星表读取和证认速度对比

Figure 4-8　Comparison of reading and identification speed of star catalogs in Protobuf format and CSV format after splitting.

天体在源星表中的顺序计算对应切分后目标星表文件的编号，若文件已存在于内存中，则与内存中的 KD-Tree 交叉证认。若未存在，则删除内存中的 KD-Tree，并读取对应的文件，建立新的 KD-Tree 进行交叉证认。统计不同阈值下的证认时间，结果如图4-8.b 所示。

可以看出，随着阈值的增大，两种格式的星表证认时间都先减小后增大。无论在何种阈值下，Protobuf 格式的星表交叉证认速度都快于 CSV 格式。CSV 格式的星表在阈值为 1000 左右时速度最快，需要 155 秒左右，而 Protobuf 格式的星表在阈值为 10,000 时速度最快，只需约 116 秒即可完成，因此在对星表存储和交叉证认时，Protobuf 格式的效率优于 CSV 格式。

4.4.3　缓存与遍历曲线测试

对不同切分层级，分别用 HEALPix NESTED 模式下自带的 Z 曲线和 Peano-Hilbert 曲线遍历整个天区，在遍历到第 i 个区域时，同时读取 i 和以 i 为中心的上、下、左、右、左上、左下、右上、右下九个区域的信息。若内存中已经缓存了该区域的信息，则为缓存命中。若不存在，则为缓存失效。此时将该区域的信息从磁盘中读取并加入到缓存，若缓存区已满，则调用替换算法选择一个区域替换，分别使用最近最久未使用（LRU）替换策略与先入先出（FIFO ）缓存替换策略进行测试。

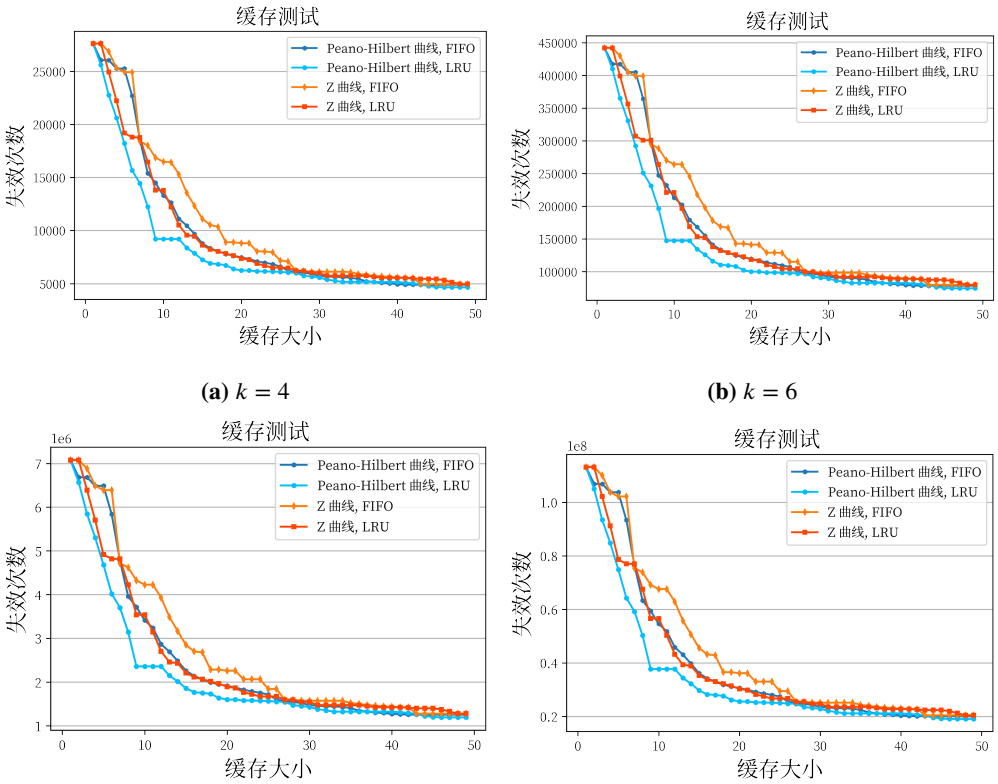
记录在不同层级和不同缓存大小的缓存失效次数，两种曲线和两种缓存算法的四种组合在四个不同的层级下的结果如图4-9所示。可以看出，

(1)　缓存越大，缓存失效的次数就会越小，但单位缓存增加带来的性能提升也随之减少。

(2)　LRU 的效率要好于 FIFO，Peano-Hilbert 曲线的效率要好于 Z 曲线。

(3)　遍历曲线和缓存替换算法的效率差距几乎不受 k 值大小的影响。

(4)　少量的缓存大小就可以大幅度减少缓存失效的次数，提高效率。例如使



(a) $k = 4$　　**(b)** $k = 6$

(c) $k = 8$　　**(d)** $k = 10$

图 4-9　不同曲线和缓存替换算法在 $k = 4, 6, 8, 10$ 时的失效次数曲线

Figure 4-9　Failure rate curves of different curves and cache replacement algorithms at $k = 4, 6, 8, 10$.

用 Peano-Hilbert+LRU 方法，与不缓存 +FIFO 相比，当缓存大小为 9 时，就可以减少大约 66.7% 的读取次数。缓存大小增大到 20 时，则可以减少 77.4% 的读取次数。

(5) 联合使用 Peano-Hilbert 曲线和 LRU 算法与 Z 曲线和 FIFO 之间的差异在缓存大小为 9 时达到了最大，接近 1 倍。

由此可见，通过使用 Peano-Hilbert 曲线和 LRU 的缓存替换策略可以有效提升在星表融合时对天体交叉证认的效率。

4.5　小结

本章节讨论了如何更高效地优化星表的存取与检索过程。通过使用开源的序列化工具，根据星表的特点定制序列化协议，既消除了不同星表之间的格式差异，又能够提升星表的访存速度。

本章节还介绍了星表融合的算法以及性能优化方法，引入缓存避免星表不必要的重复读取。通过使用更加高效的缓存曲线和缓存替换算法，能够在更少缓存的情况下，减少缓存失效的次数，提升缓存的利用率。

第 5 章　分布式星表检索

随着大规模星表数据量越来越大，以及大视场暂现源搜寻这种既需要同时检索大量天体，又有时间限制的需求不断出现，对星表检索的性能需求不断提升。当检索性能无法满足当前的任务需求时，传统方法是通过增加系统的硬件配置，即垂直扩容的方式提升检索的速度。但是单个服务器受到 CPU 性能、I/O 接口带宽、内存容量及带宽等因素的限制，性能无法无限提高。此时可以通过水平扩容的方式，即增加服务器的数量，使用分布式并行检索技术是解决该问题的合适方案。

5.1　星表分布式检索架构

本文受到 MapReduce 架构的启发，根据星表检索的特点，提出了一套分布式的检索系统。该系统包含三类节点：主节点、计算节点、存储节点。在检索开始之前，使用本文切分算法切分后的星表文件会被按照规则分发到各个存储节点中。在星表检索时，按照流程可以分为如下三个步骤：

(1) 新的检索请求会被发送到主节点，主节点会根据待检索的坐标和距离阈值，以及计算节点的负载情况，星表在存储节点的分布等信息综合计算，将该请求分解并分发到一个或多个合适的计算节点中。

(2) 计算节点收到检索请求后根据需要检索的坐标信息，读取在存储节点中保存的星表或直接使用缓存中的星表，并使用对应的检索算法对星表进行检索，将检索结果返回给主节点。

(3) 主节点接收到一个或多个计算节点的检索结果后，对结果做整合，得到最终的检索结果并返回。

在当前环境下，本文在实际应用时选择使用 X86 架构服务器，并将计算节点与存储节点放置在同一台服务器上，即计算节点直接从硬盘中读取对应的星表。此时，计算节点与存储节点的关系是一对一的，即一台计算节点只能读取一个存储节点内的数据。这种架构可以简化系统的部署和维护，同时也可以提高数据的读取速度。

在未来，若计算节点使用 GPU、FPGA、嵌入式等非传统服务器检索星表时，星表可以保存在独立的存储节点或者云存储。此时，计算节点与存储节点的关系可以是一对多或者多对多，但检索的基本原理并未发生改变。

5.2　多个分布式节点之间的通信

主节点与多个计算节点一般通过高速网络互联。主节点通过事先约定好的通讯协议与各个子节点通信，获得各个子节点当前的状态，发送检索请求，接受

检索结果等。本文使用 gRPC （gRPC Remote Procedure Calls）作为节点之间的通信方式。

谷歌公司的开源项目 gRPC 是一种高性能、通用的开源 RPC 框架，它基于 Protocol Buffers 序列化协议开发，支持多种编程语言，包括 C++、Java、Python、Go、Ruby、C# 等。gRPC 使用 HTTP/2 协议进行数据传输，可以在客户端和服务器之间进行双向流式传输，从而提高了传输效率。gRPC 还支持多种负载均衡算法和服务发现机制，可以轻松地实现服务注册和发现。

gRPC 的核心是远程过程调用，它允许运行于一台计算机的程序调用其他地址空间的程序，即使这个程序运行于不同的计算机上。gRPC 使用 Protocol Buffers 作为接口描述语言（IDL），定义了服务和消息格式。通过 IDL 文件定义接口后，可以使用 gRPC 提供的工具自动生成客户端和服务器端的代码。

gRPC 的优点在于高效、跨语言、跨平台、易于扩展等。它可以帮助开发人员更快地构建分布式系统，并且可以轻松地与其他系统集成。gRPC 支持同步和异步两种调用方式。同步调用是指客户端发送请求后，必须等待服务器响应后才能继续执行下一步操作。而异步调用则是指客户端发送请求后，不必等待服务器响应即可继续执行下一步操作。在异步调用中，客户端可以通过回调函数或 Future 对象来获取服务器响应结果。

按照设计，每个计算节点作为服务端，在启动后会监听一个网络端口，并接收发往该端口的消息，根据消息的内容执行相应的检索或其他指令。主节点作为客户端，在启动时根据配置文件中保存的节点信息，寻找对应的计算节点，向每个计算节点发送信息，处理检索的结果等。

5.3　星表的分布式存储策略

大规模星表在各个存储节点中有多种存储方式。每一种类型都适用于不同的检索场景，在实际应用时可以根据具体的任务需求选择适合的存储方式。

5.3.1　镜像存储策略

这种方式是每个存储节点都保存有完整的大规模星表，如图5-1(a) 所示。当有检索的请求时，主节点根据调度的策略选择合适的节点，也可以随机将请求发送到任意一个可用的节点。节点收到请求后按照请求的内容检索星表，并获得检索的结果。

该分布式检索方式的好处是冗余程度高，除非所有的节点全部都出现故障，否则保证服务一直可用。在同一时间出现大量检索请求的时候，能够将检索请求均匀地分发到各个节点中，每个节点都需要处理相近数量的任务。计算节点给出的结果即是最终的结果，主节点不需要再对检索的结果做后处理，主节点的负载较轻。

另外一个优点是节点的扩容简单，如果当前节点的数量不足以满足检索的需求时，只需要增加新的计算节点即可。在理想的情况下，当从 N 个节点增加

(a) 镜像模式　　(b) 散列模式　　(c) 混合模式

图 5-1　星表在分布式场景下不同的存储模式

Figure 5-1　Different storage models for catalogs in distributed scenarios.

到 $N+1$ 个节点时，此时的分布式检索的性能也会变成原来的 $\frac{N+1}{N}$。之前的节点中的数据也无需重新散列，扩容时不会中断系统运行。

但是该方式的缺点也较为明显。每个节点都要保存完整的星表，将会占用大量的硬盘空间。对于每天都有大量观测数据产出的巡天计划而言，N 个检索节点就要付出 N 倍的存储成本。

5.3.2　散列存储策略

另外一种存储方式是每个节点只保存一部分星表文件，在检索时根据坐标选择合适的节点检索，如图5-1(b) 所示。一个好的星表分发策略可以使检索时的并行程度最大化，提高检索速度。

星表在经过切分之后得到了大量的星表文件，每个文件都有一个 UNIQ 编号。考虑到每次检索时需要检索的区域范围可能会覆盖多个星表文件，因此，如果能将区域相邻的星表文件分发到不同的节点中，检索时由多个节点共同参与对一个坐标的检索，从而提高并行程度。

常用的散列方式是对编号取余，例如对于 UNIQ 编号为 id 的星表文件，若分布式节点的数量为 N，则该文件应该被分发到编号为 $id\%N$ 的节点中。这种方式的优点是编号相邻的文件一定会被发送到不同的节点中，有助于提高检索时的并行程度。而且每个节点只保存星表的 $\frac{1}{N}$，相较于镜像存储方式能够大幅节约存储成本。若某个检索请求需要检索面积较大的区域时，能够根据检索请求分散到多个节点中并行检索，实现负载的均衡。该方式的缺点是容错率低，N 个节点中只要有一个出现故障，则整个检索系统就无法运行。当需要增加节点时，需要将星表重新散列和分发，耗时较长，过程中检索服务会被中断。

5.3.3　混合存储策略

混合存储策略是指将计算节点组成不同的分区，一个分区内的节点采用相同的存储策略，分区之间采取另一种存储策略。例如，可以将 $N \times M$ 个节点划分为 N 个分区，每个分区有 M 个节点。分区与分区之间使用散列存储策略，分区内的节点使用镜像存储策略，如图5-1右图所示。即此时的星表文件按照编号分发到不同的分区中，一个分区内的所有节点都存储相同的星表。检索时先根据待检索坐标选择合适的分区，再根据调度策略选择分区内的某个节点进行检索。同理，也可以在分区与分区之间使用镜像存储策略，分区内的节点使用散列存储策略。

混合存储策略相较于散列存储有一定的冗余能力，又相较于镜像存储可以减轻存储的压力。但是该存储策略对节点的数量要求较高，如果只有少量节点（少于 4 个），则不适合该存储方式。

5.4　检索调度策略

不同的分片策略对应不同的检索调度策略。若待检索区域的星表存在多个节点中，则需要使用调度从中挑选出一个最合适的计算节点。对每个检索请求有两种常见的检索的调度调度策略：

(1) 轮询，即按照节点的顺序发送请求。

(2) 随机，即每次检索时都选择随机的节点发送请求。

以上两种调度的策略均未考虑到计算节点当时到负载情况。容易出现多个节点的负载不一致的情况，导致检索效率的下降。

若某个检索任务的接收时间为 t_{rec}，经过检索之后将检索结结果发送的时间为 t_{send}，此时的检索耗时为 $t_{\text{rec}} - t_{\text{send}}$。一个理想的调度算法，应该将总的检索耗时最小，即：

$$COST = \min \sum_{Allquery} (t_{\text{rec}} - t_{\text{send}}) \tag{5-1}$$

可以考虑根据不同节点的负载建立一套负载均衡的调度方式。主节点增加一个负载列表记录每个节点的当前负载情况，向某个节点发送检索请求时将该节点的负载值加 1，接收到该节点的返回结果时将该节点的负载值减 1。此时，负载列表里的数值即代表了某个节点内等待检索的坐标数量。数量越多，该节点负载越大。在有新的坐标需要检索时，主节点优先选择向负载值小的节点发送检索请求，实现了检索时的负载均衡。

当一个节点接收到检索的请求时，若此时该计算节点的内存中没有缓存该请求所需要检索的 KD-Tree，则需要先读取硬盘，才能开始检索，本文称此过程为“冷检索”。与之相对应，若该节点的内存中存在对应的 KD-Tree，则直接就可以检索，本文称之为“热检索”。显然，热检索的耗时小于冷检索。若同时存在两个空闲的节点，其中一个的检索方式是冷检索，另一个是热检索，则此时应

该将该请求发送到热检索的那个节点中以实现更快的检索速度。这要求主节点在调度之前就知道每个计算节点当前缓存内容，即计算节点当发生缓存的新增或替换时需要将信息实时传输到主节点中。

5.5 分布式系统的快速部署

由于不同的计算节点运行环境可能各不相同，在配置部署系统时难免会出现不兼容等情况。例如，本文使用的 HEALPix 官方 Python 软件包 healpy 仅支持 Linux 和 macOS 系统，而不兼容 Windows[1] 。同时，当节点数量较多时，若需要更新程序，则需要手动下载软件代码到每一个节点中并编译安装，消耗大量人力和时间。近年来较为流行的容器技术可以解决上述问题。本文采用 Docker 实现了分布式系统的快速部署。Docker 是一种开源的容器化平台，它可以让开发人员将应用程序及其依赖项打包成一个独立的容器，从而实现跨平台、快速部署和可移植性等优势。

本文使用了 Docker 提供的 Ubuntu 镜像作为系统底层。将所使用到的代码和第三方软件库打包为一个容器并发布。当需要配置新的节点时，在新的节点中安装 docker 并加载制作好的镜像，根据实际的网络地址等情况，修改主节点和计算节点中的配置文件，将星表数据传输到新节点之后，即可完成节点的配置工作。

5.6 分布式系统的实现

为了实际测试本文中构建的分布式检索系统的性能，本文使用了 4 台国家天文科学数据中心提供的云虚拟机作为计算节点，4 台虚拟机分别运行于 4 台不同的物理服务器，每台计算节点的硬件配置为：

- CPU：Intel(R) Xeon(R) CPU E5-2690 4 Core @ 2294.686 MHz
- 内存：8 GB
- 硬盘：100GB ，I/O 速度：82.5 MB/s
- 操作系统：Ubuntu 20.04.4 LTS
- 编程语言：C++、Python

主节点使用一台如下配置的计算机：

- CPU：11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
- 内存：64 GB
- 硬盘：1024GB 固态硬盘
- 操作系统：Windows 11

主节点与 4 个计算节点通过内网互联，带宽为 1000 Mbps

[1] https://healpy.readthedocs.io/en/latest/install.html#requirements

5.6.1　主节点的异步请求实现

主节点由多个星表读取线程、多个通信线程和多个数据处理线程组成，线程的具体数量由计算节点的数量确定，在本文中读取线程、通信线程、数据处理线程均为 4 个。星表读取线程负责读取硬盘中待检索的星表到内存中。通信线程每个负责与对应个计算节点通信。数据处理线程负责对结果做处理。

主节点在启动时会先加载每个计算节点中存储的星表 UNIQ 编号列表，根据该列表在之后的检索时选择合适的计算节点。之后 4 个通信线程分别负责与 4 个计算节点通信。当星表读取线程读取一个天体时，首先解析出对应的天体坐标，并根据该坐标和检索范围计算得到待检索区域所覆盖到的 UNIQ 星表列表，并计算出每个星表文件所属的计算节点，生成对应的检索请求发送到对应节点中。

发送请求后，每个通信线程不会停下来等消息返回，而是使用异步处理的方式，如图5-2所示，将该请求放入到一个处理队列中，并继续处理发送下一个请求。当通信线程接收到计算节点发回的检索结果后，则在处理队列中找到对应的请求，并将检索结果加入其中。数据处理线程会不断尝试从队列出取出所有已经检索完成的请求，并根据设定好的规则做数据的整合，得到最终的检索结果。

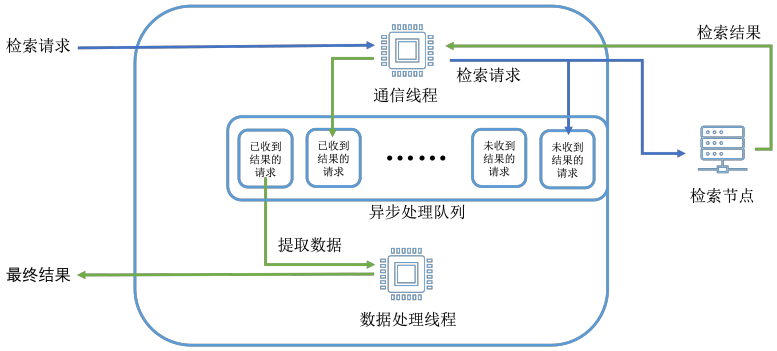


图 5-2　主节点的异步请求示意图

Figure 5-2　Diagram of asynchronous request of the master node.

5.6.2　计算节点的异步检索实现

由于每个计算节点的 CPU 有 4 个核心，同时可以并行 4 个线程。本文的计算节点中在运行时会启动 1 个主线程和 3 个检索线程。主线程负责初始化和开启网络通信，包括监听端口，接收检索请求与发送检索结果。星表检索工作由检索线程负责。

主线程在启动后，会读取当前节点所包含的 Protobuf 格式的星表文件列表以及解析每个星表文件的 UNIQ 编号，读取过程如图5-3所示，将这些 UNIQ 编号放置到一个 Map 数据结构中，以便快速检索是否存在对应编号的文件。之后，主线程将转变为通信线程，开始监听端口，接收主节点的检索请求。

如图5-4所示，当接收到检索请求时，通信线程会将请求放置到一个名为

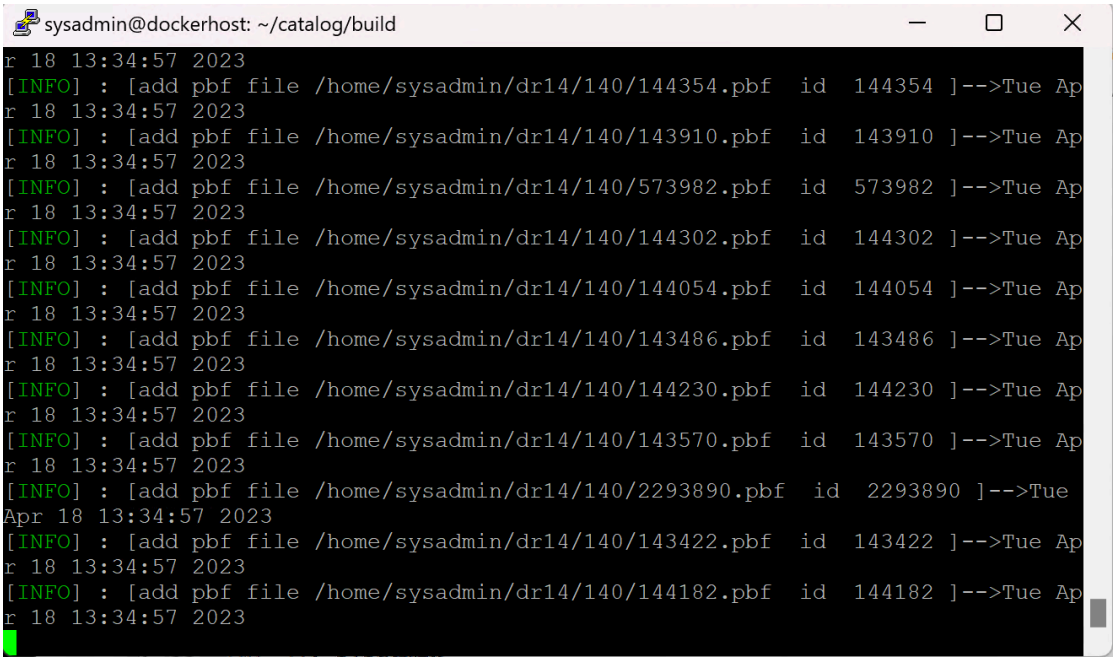


图 5-3　计算节点的初始化过程

Figure 5-3　Picture of initialization of computing nodes.

CompletionQueue 的异步处理队列中，并继续接收下一个检索请求。3 个检索线程中，如果某个线程处于闲置状态，则会从该队列读中读取待检索的请求，开启检索过程。

在检索时，首先会根据待检索的坐标和距离阈值，计算所覆盖到的文件列表。并从 Map 里判断该列表中的每个文件是否存在于当前节点中，若存在，则需要检索该文件。同时，所有的检索线程都共同维护一个已读取到内存中的 KD-Tree 列表。若待检索星表文件的 KD-Tree 不在列表中，则从硬盘中读取对应的文件并建立 KD-Tree 放置到该队列中，若队列中已经存在 KD-Tree，则直接检索。

在建立 KD-Tree 时，需要加并发控制锁，以避免不同线程之间产生冲突。在检索时，即使多个线程共同检索一个 KD-Tree，也只会读取 KD-Tree 的数据，不会对 KD-Tree 做任何修改，所以不会产生并发冲突，能够充分发挥并行检索的性能。若当前计算节点内存中的 KD-Tree 达到数量上限，则使用缓存替换算法，找到最近最久未被使用的那个，将其替换。

当检索到对应的结果后，将结果放回到处理队列对应的位置中，将该请求的队列设置为完成，通信线程会将完成后的请求返回给主节点。

5.6.3　测试星表概况

测试星表使用了包含部分参数的 SDSS DR14 星表 (Abolfathi et al., 2018)，共有 1,231,051,050 个目标，总大小为 235GB，每个目标都包括对象标识符（objID）、赤经和赤纬（ra，dec）、赤经和赤纬的误差（raErr，decErr）、银经和银纬（b，l）、HTM 标识符（htmID）、各波段亮度（u，g，r，i，z）以及天体类型（type）等信

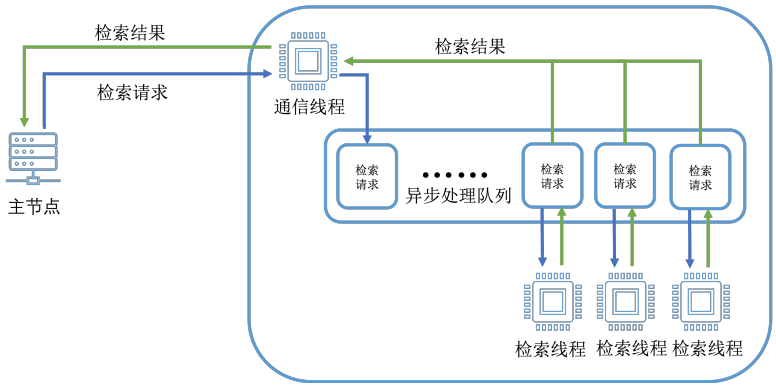


图 5-4　计算节点的异步检索示意图

Figure 5-4　Diagram of asynchronous retrieve of the computing node.

息。使用本文中提出的“中立”模式动态层级星表切分算法，阈值设定为 5000。星表在主节点被切分，切分以后共得到 141,943 个星表文件，以 Protobuf 格式存储，总大小 265GB。

由于分布式的节点数量有限，且硬盘空间不足以容纳完整的星表。因此本文的测试使用散列分布策略，按照 UNIQ 编号对 4 取模作为星表分发的规则。使用文件传输协议将切分后的星表分别发送到 4 个计算节点中。分发之后每个节点获得星表数量与总数据量如表5-1所示。

表 5-1　星表在各个计算节点内的分布情况

Table 5-1　Distribution of catalogs within each node.

节点编号	节点内星表文件数量	总数据量
0	35,310	66.0 GB (70,972,613,626 字节)
1	35,649	66.8 GB (71,743,566,472 字节)
2	35,492	66.5 GB (71,501,647,355 字节)
3	35,492	66.3 GB (71,280,476,447 字节)

5.6.4　检索性能测试

5.6.4.1　单次请求坐标数量与检索速度的关系测试

检索的速度受网络传输的影响很大。主节点通过 gRPC 发送的每个检索请求都需要打包成数据包，经由网卡发送，并在计算节点解包，该过程需要消耗一定的时间与性能。当单次检索的坐标较少时，有较大比例的时间花费在网络传输中。因此，在同时有大量检索的坐标时，将多个待检索的坐标打包在一起发送可以达到更高的效率。

本文测试了一个请求内的天体数量与检索速度的关系。测试仅使用单个编号为 1 号的计算节点，计算节点内存储有分发后的星表。主节点从切分后取模为 1 的 SDSS 星表中，按照 UNIQ 编号的顺序读取前 100 万个天体，分别在单个请求内包含不同的坐标数量，发送到 1 号计算节点中计算最近邻天体，检索距离阈

值设定为 0.1 度，结果如表5-2所示。

可以看出，一个请求内只包含有一个坐标时，检索速度最慢。此时每个坐标都要发送一个数据包，大量的时间和性能将消耗于数据包的打包与解包过程中。随着请求内坐标数量的增加，检索的耗时也逐渐减低，直到单个请求内有 1000 坐标时，速度达到最快。在热检索的情况下每秒钟可以检索超过 20 万个天体。当请求内坐标数量继续增加时，此时数据包的大小超过了网卡的最大传输单元（MTU）的限制，需要将数据包拆分后才能发送，到达计算节点后需要将数据包再次组装，导致检索速度开始下降。因此，若需要同时检索大量坐标时，在本文的测试环境下，以 1000 个坐标作为一个请求做分布式检索，可以获得最高的检索效率。

表 5-2　单个计算节点单个请求不同坐标数量 100 万天体检索速度测试

Table 5-2　The retrieval speed test for 1 million objects by single computation node and single request.

	热检索		冷检索		
单个请求坐标数量	耗时 (秒)	平均速度（天体每秒）	耗时 (秒)	平均速度（天体每秒）	耗时差值
1	75.533	13,239	72.498	13,793	-4.295
10	13.325	75,047	15.909	62,587	2.584
100	6.414	155,908	9.558	104,624	3.144
500	5.211	191,901	8.507	117,550	3.296
1000	4.776	209,380	7.594	131,682	2.818
2000	4.957	201,734	7.969	125,486	3.012
5000	16.152	61,911	19.110	52,328	2.958
10,000	15.434	64,792	19.700	50,761	4.266

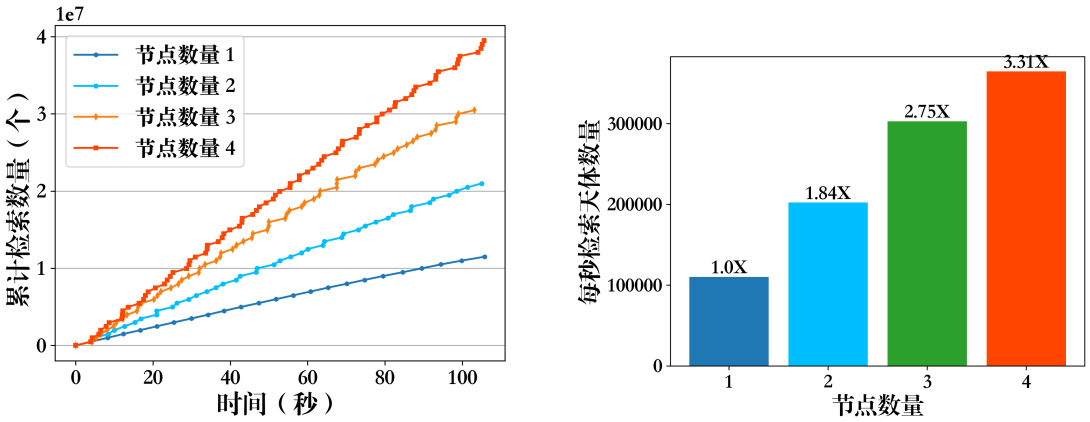
5.6.4.2　冷检索与热检索的性能对比

为了比较冷检索与热检索之间的性能差异，本文在上节测试的基础上，以相同的检索方式做了 100 万个天体在冷检索与热检索之间的速度对比，检索 100 万个天体共需要读取 84 个切分后的星表文件。结果如表5-2所示。可以看出，在单个请求内坐标数量在 100 至 5000 的范围内，冷检索与热检索之间的时间差值均在 3 秒左右。在坐标小于 100 或大于 5000 时，受网络波动等因素影响较大，导致耗时出现差异。可以看出，在请求坐标数量为 1000 时，节点的检索速度从热检索的每秒钟大约 20 万个坐标每秒降低到了冷检索的大约 13 万个坐标每秒。由此证明了，在检索之前将检索数据尽量发送到有缓存的热检索节点可以获得较大的性能提升。

5.6.4.3　多节点分布式并行检索测试

为了测试不同的节点数量在并行检索时的检索性能差异，本文在上述测试的基础上，按照检索节点的数量为 1、2、3、4 时的 4 种情况分别进行了最近邻

检索测试，检索半径设定为 0.1 度。每个请求内包含 1000 个坐标，计算节点初始时缓存为空。每次测试程序运行时长在 100 秒左右，每发送 50,000 个检索坐标，记录一次当前的运行时间和接收到的检索结果数量，测试结果如5-5所示。



(a) 不同数量节点的检索数量与运行时间的关系　　(b) 不同数量检索节点的平均速度和加速比

图 5-5　不同数量分布式节点的检索性能测试

Figure 5-5　The retrieval speed test results with different numbers of computation nodes.

从图中可以看出，与单个检索节点相比，2 个、3 个、4 个节点的检索加速比分别为 1.84、2.75、3.31。随着分布式节点的增加，对主节点的数据处理能力和网络的带宽要求也在提高。当节点数量小于 3 时，增加一个节点可以近似获得 0.8 倍于单个节点的性能增加。但随着检索节点的增加，尤其是 4 个节点时，主节点的硬件性能和网络带宽逐渐成为瓶颈，新增加节点产生的“收益”减小为 0.6 倍的单机性能。

之后，本文测试了在 4 个节点的情况下，完整检索一遍 235G，共 12 亿行 SDSS 星表的时间消耗为 105 分钟。检索过程中所消耗的带宽峰值已经接近千兆网络的上限。本文的测试验证了使用分布式检索可以获得远远超过了单机节点的检索速度，能够满足 GWAC 等大视场暂现源搜寻任务的检索性能要求。

5.7　小结

为了解决单机节点的速度瓶颈，本章节讨论了星表的分布式检索问题。通过将分布式检索拆分为“请求分发”和“结果汇总”两个过程，将节点分成了“主节点”和“计算节点”两种类型，两种节点各司其职，互相配合。首先将切分好的星表按照分发规则分发到各个计算节点里。在接收到检索请求后由主节点进行调度，选择最合适的一个或多个节点做检索，得到结果后再由主节点做整合，实现了检索过程的高度并行化。

在上述原理的基础上，本文搭建了一套实际可用的分布式系统，并测试了不同情况下的检索速度。在大规模星表的测试中，能够获得远超单机节点的检索速度。证明了分布式检索的可行性和有效性。

第 6 章　总结与展望

随着天文学逐渐迈向多波段、多信使以及时域观测时代，越来越多大视场时域巡天观测计划被提出，尤其是暂现源的观测等项目，将会对现有的数据处理能力带来巨大的挑战。本文针对如何提高大规模星表的检索效率，提出了多种优化方法。本文研究成果主要包括以下几个方面：

为了能够在有限内存的情况下将星表均匀切分成合适大小的文件，尽可能提高星表检索效率，本文做了详细的研究。本文在 HEALPix 的基础上，提出了一种按照天体的区域密度进行切分的动态层级星表切分算法。通过引入区域重复覆盖的特性，使用递归加回溯的方式，将切分出的子星表内的天体数量限制在 1 倍阈值至 4 倍阈值之间，实现了星表的均匀切分。通过使用 Gaia、LAMOST 等实际观测得到的星表进行测试，实验结果表明该算法有着最小的变异系数，切分结果的均匀程度优于已有的其他切分策略。

为了解决不同星表格式不统一的问题，同时避免字符串与数字的重复转换以提高效率，本文尝试使用开源的序列化库 Protocol Buffers 定义了一套星表的序列化规则，优化了数据存取性能，提高了检索的速度与效率。同时也通过该序列化方法实现了对 KD-Tree 的序列化，能够将 KD-Tree 保存到硬盘中，减少了重复建立 KD-Tree 的过程，加快了检索速度。

在多星表的融合优化方面，本文尝试使用增加缓存的方式提高星表融合时的速度。在此基础上还对星表访问的顺序进行了优化，使用具有更高缓存效率的 Peano-Hilbert 曲线取代了 HEALPix 原有的遍历曲线，提高了缓存的利用率。

为了提高检索的速度，本文尝试使用多个节点的分布式检索替代单机检索。研究了多节点之间的通信、星表在多节点中的分发方式、检索的调度方式等内容，并进行了系统实现与应用测试。实验结果表明，多节点分布式检索能够有效提升检索时的速度。

在下一阶段的工作中，本文所提到的工作仍有可以进一步优化与改进的空间。例如，本文阐述了星表分布式检索原理，并根据该原理自研出一套分布式检索系统。但是受到条件限制，本文只测试了最多 4 个节点的分布式星表在散列存储下的检索情况，虽然能够证明了本文所提出算法的有效性，以及证明了星表分布式检索相对于单机检索的性能优势，未来可以测试在更多节点下的工作情况。在当前环境下，大规模星表在检索前仍然需要手动切分，并将星表手动传输到各自的计算节点中。未来可以对此做进一步研究，实现一套星表自动切分的算法，实现星表由主节点自动切分并传送到各个计算节点中。

作为中国虚拟天文台的研发成果之一，本文所提出的系统在未来将以在线服务的形式，与其他系统一起，构建成为一整套的在线科研平台，发布到国家天文科学数据中心，供全世界的天文科研人员使用。

参考文献

崔辰州, 于策 & 肖健等. 2015. 大数据时代的天文学研究[J]. 科学通报, 60(445-449).

樊东卫, 何勃亮 & 李长华等. 2019. 球面距离计算方法及精度比较[J]. 天文研究与技术, 16(1): 69-76.

许允飞. 2020. 博士论文[D]. 北京: 中国科学院国家天文台.

赵青. 2011. 博士论文[D]. 天津: 天津大学: 11-12.

高丹. 2008. 博士论文[D]. 北京: 中国科学院国家天文台: 32-36.

Abdurro'uf, Accetta K, Aerts C, et al. 2022. The Seventeenth Data Release of the Sloan Digital Sky Surveys: Complete Release of MaNGA, MaStar, and APOGEE-2 Data[J]. ApJS, 259(2): 5.

Abolfathi B, Aguado D S, Aguilar G, et al. 2018. The Fourteenth Data Release of the Sloan Digital Sky Survey: First Spectroscopic Data from the Extended Baryon Oscillation Spectroscopic Survey and from the Second Phase of the Apache Point Observatory Galactic Evolution Experiment [J]. ApJS, 235(2): 42.

Bern M, Eppstein D & Teng S H. 1999. Parallel construction of quadtrees and quality triangulations [J]. International Journal of Computational Geometry & Applications, 9(06): 517-532.

Boch T, Pineau F & Derriere S. 2012. The CDS Cross-Match Service[C]//Ballester P, Egret D & Lorente N P F. Astronomical Data Analysis Software and Systems XXI: Vol. 461. San Francisco: ASP Conf.Ser.: 291.

Budavári T, Szalay A S, Gray J, et al. 2004. Open SkyQuery – VO Compliant Dynamic Federation of Astronomical Archives[C]//Ochsenbein F, Allen M G & Egret D. Astronomical Society of the Pacific Conference Series: Vol. 314 Astronomical Data Analysis Software and Systems (ADASS) XIII. 177.

Budavári T & Szalay A S. 2008. Probabilistic Cross-Identification of Astronomical Sources[J]. The Astrophysical Journal, 679(1): 301-309.

Cordier B, Wei J, Atteia J L, et al. 2015. The SVOM gamma-ray burst mission[Z]. arXiv:1512.03323. arXiv: 1512.03323.

de Berg M, van Kreveld M, Overmars M, et al. 1997. Computational geometry: Algorithms and applications[M]. Berlin, Heidelberg: Springer Berlin Heidelberg: 289-304.

Dean J & Ghemawat S. 2008. Mapreduce: simplified data processing on large clusters[J]. Communications of the ACM, 51(1): 107-113.

Du P, Ren J, Pan J, et al. 2014. New cross-matching algorithm in large-scale catalogs with Thread-Pool technique[J]. Science China Physics, Mechanics, and Astronomy, 57(3): 577-583.

Fan D, Budavári T, Norris R P, et al. 2020. Optimal probabilistic catalogue matching for radio sources[J]. Monthly Notices of the Royal Astronomical Society, 498(1): 565-573.

Fernique P, Allen M G, Boch T, et al. 2015. Hierarchical progressive surveys. Multi-resolution HEALPix data structures for astronomical images, catalogues, and 3-dimensional data cubes[J]. A&A, 578: A114.

Fernique P, Boch T, Donaldson T, et al. 2014. MOC - HEALPix Multi-Order Coverage map Version 1.0[Z]. 12. arXiv: 1505.02937.

Gaia Collaboration, Prusti T, de Bruijne J H J, et al. 2016. The Gaia mission[J]. A&A, 595: A1.

Gaia Collaboration, Brown A G A, Vallenari A, et al. 2018. Gaia Data Release 2. Summary of the contents and survey properties[J]. A&A, 616: A1.

Gaia Collaboration, Vallenari A, Brown A G A, et al. 2022. Gaia Data Release 3: Summary of the content and survey properties[Z]. arXiv:2208.00211. arXiv: 2208.00211.

Górski K M, Hivon E, Banday A J, et al. 2005. HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere[J]. ApJ, 622(2).

Gray J, Nieto-Santisteban M A & Szalay A S. 2007. [Z]. cs/0701171. arXiv: cs/0701171.

Hanisch R J, Farris A, Greisen E W, et al. 2001. Definition of the Flexible Image Transport System (FITS)[J]. A&A, 376: 359-380.

Ivezić Ž, Kahn S M, Tyson J A, et al. 2019. LSST: From Science Drivers to Reference Design and Anticipated Data Products[J]. ApJ, 873(2): 111.

Kaiser N. 2004. Pan-STARRS: a wide-field optical survey telescope array[C]//Oschmann J, Jacobus M. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series: Vol. 5489 Ground-based Telescopes. 11-22.

Kleppmann M. 2017. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems[M]. Sebastopol: O'Reilly Media, Inc.: 112-113.

Koposov S & Bartunov O. 2006. Q3C, Quad Tree Cube – The new Sky-indexing Concept for Huge Astronomical Catalogues and its Realization for Main Astronomical Queries (Cone Search and Xmatch) in Open Source Database PostgreSQL[C]//Gabriel C, Arviset C, Ponz D, et al. Astronomical Data Analysis Software and Systems XV: Vol. 351. San Francisco: ASP Conf.Ser.: 735.

Liu J, Soria R, Wu X F, et al. 2021. The SiTian Project[J]. An. Acad. Bras. Ciênc. vol.93 supl.1, 93: 20200628.

Malkov O, Dluzhnevskaya O, Karpov S, et al. 2012. Cross Catalogue Matching with Virtual Observatory and Parametrization of Stars[J]. Baltic Astronomy, 21: 319-330.

Martinez-Castellanos I, Singer L P, Burns E, et al. 2022. Multiresolution HEALPix Maps for Multiwavelength and Multimessenger Astronomy[J]. AJ, 163(6): 259.

Masci F J, Laher R R, Rusholme B, et al. 2019. The Zwicky Transient Facility: Data Processing, Products, and Archive[J]. PASP, 131(995): 018003.

O'Mullane W, Banday A J, Górski K M, et al. 2001. Splitting the sky - htm and healpix[C]//Banday A J, Zaroubi S & Bartelmann M. Mining the Sky. Berlin, Heidelberg: Springer Berlin Heidelberg: 638-648.

O'Mullane W, Budavári T, Li N, et al. 2005. OpenSkyQuery and OpenSkyNode - the VO Framework to Federate Astronomy Archives[C]//Shopbell P, Britton M & Ebert R. Astronomical Society of the Pacific Conference Series: Vol. 347 Astronomical Data Analysis Software and Systems XIV. 341.

Reinecke M & Hivon E. 2015. Efficient data structures for masks on 2D grids[J]. A&A, 580: A132.

Schäfer B M. 2005. Dissertation[D]. München: LMU München: Faculty of Physics: 99-104.

Shafranovich Y. 2005. Common format and mime type for comma-separated values (csv) files[R].

Soumagnac M T & Ofek E O. 2018. catshtm: a tool for fast accessing and cross-matching large astronomical catalogs[J]. Publications of the Astronomical Society of the Pacific, 130(989): 075002.

Sutherland W & Saunders W. 1992. On the likelihood ratio for source identification[J]. Monthly Notices of the Royal Astronomical Society, 259(3): 413-420.

Varda K. 2008. Protocol buffers: Google's data interchange format[J]. Google Open Source Blog.

Wan M, Wu C, Wang J, et al. 2016. Column store for gwac: a high-cadence, high-density, large-scale astronomical light curve pipeline and distributed shared-nothing database[J]. Publications of the Astronomical Society of the Pacific, 128(969): 114501.

Warren M S & Salmon J K. 1993. A parallel hashed oct-tree n-body algorithm[C]//Proceedings of the 1993 ACM/IEEE conference on Supercomputing. New York: Association for Computing Machinery: 12-21.

York D G, Adelman J, Anderson J, John E., et al. 2000. The Sloan Digital Sky Survey: Technical Summary[J]. AJ, 120(3): 1579-1587.

Youngren R W & Petty M D. 2017. A multi-resolution healpix data structure for spherically mapped point data[J]. Heliyon, 3(6): e00332.

致　谢

时光荏苒, 岁月如梭, 三年的时光在不经意间就悄然结束。回望我这三年的求学时光，从刚入学的懵懂无知，到如今即将毕业，不禁感慨万千。在攻读硕士期间和本次毕业论文的撰写过程中，我得到了许多人的支持和帮助，在此向他们表示最诚挚的感谢。

首先，我要感谢我的导师樊东卫副研究员。樊老师待人随和，有着丰富的经验和高超的技术水平。在整个硕士阶段，他给予了我大量无私的指导和帮助，他不仅给我指明了宝贵的研究思路和方法，还耐心地解答了我在研究中遇到的各种问题。他的言传身教让我受益匪浅，在学术上和生活中都给予了我很多启示和帮助。没有他的支持和鼓励，我无法完成这篇论文。

还要感谢我们组的首席科学家崔辰州研究员。回顾我与崔老师相识，还要追溯到 2012 年，当时还是一个初中生的我无意中发现了崔老师为天文爱好者所建立的宇宙驿站，便申请加入其中。在 2013 年 6 月，我参加了在兴隆举办的驿站站长交流会，正是在此期间对天文台站的参观和与崔老师、其他天文工作者、同好们的交流，让我开始有了从事天文研究的梦想。在 2019 年准备考研时，看到崔老师的招生信息，便决心报考，最终成功来到了国台读研。从在兴隆第一次见到崔老师，到如今毕业，已经过去整整十年了。在科研方面，崔老师是一位富有经验和远见卓识的老师，指引着我们组的研究方向。崔老师在我的学习期间给了我很大的支持和帮助，为我们提供了优良的学习和科研条件，在我的科研和论文写作方面给了我诸多指导，多次帮我修改论文并提出许多富有建设性的意见。

还要感谢国家天文台天文信息技术团组的其他各位老师，他们是韩军、何勃亮、李长华、李珊珊、米琳莹、陶一寒、王有芬、许允飞、杨涵溪、杨丝丝。非常荣幸能够在组里度过硕士期间的科研时光。

还要感谢与我同组的同学们。包括吴莹、杨嘉宁、张震同学，以及马鹏辉、邵务俊、朱珈莹、左肖雄师弟师妹。他们在生活上给予了我很多关心和帮助，在学习上与我共同进步，在研究中与我分享经验和思路。他们的陪伴和支持让我感到温暖和安慰，让我在学业上更加坚定和自信。

还要感谢国台教育处的各位老师，他们是梁艳春老师、李响老师、马怀宇老师、毛永娜老师、郑菲菲老师。他们在背后默默的付出，给我们提供了莫大的帮助，让我们能够安心科研。

还要感谢在雁栖湖期间国科大天文学院和天文协会的老师和同学们，尤其是梁汝汐、赵飞宇、刘广畅、孙莉焰等几位同学，有幸和他们在雁栖湖度过了一段难忘的时光。

感谢本篇论文的评阅老师和答辩委员会的各位老师，感谢他们在百忙之中评阅我的拙作，给我的论文提出指导意见，参与我的论文答辩。

我还要感谢我的家人。他们一直以来都是我的坚强后盾，在我遇到困难时给予了我无限的支持和鼓励。他们默默地付出着，在背后默默地支持着我的学业。没有他们的理解和支持，我无法顺利完成学业。

最后，再次感谢所有支持和帮助过我的人，祝愿大家一切顺利，前程似锦！

2023 年 6 月

作者简历及攻读学位期间发表的学术论文与其他相关学术成果

作者简历:

2016 年 09 月——2020 年 06 月，在山东财经大学计算机科学与技术学院获得学士学位。

2020 年 09 月——2023 年 06 月，在中国科学院国家天文台攻读硕士学位。

已发表（或正式接受）的学术论文:

(1) 张琦乾，樊东卫，崔辰州等. 基于文件的大规模星表高效索引与融合. 天文学进展，2023（已接收）.

申请或已获得的专利:

发明专利:张琦乾,樊东卫,崔辰州;切分星表的方法和装置;CN202210941039.8（已受理）

参加的研究项目及获奖情况:

参与国家自然科学基金项目《面向大视场时域巡天观测的大数据检索与融合方法研究》