UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DISEÑO E IMPLEMENTACIÓN DE UN SERVICIO DE CROSSMATCHING ASTRÓNOMICO USANDO APACHE CASSANDRA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

KAREN CECILIA MEDINA MONTIEL

PROFESOR GUÍA:
AIDAN HOGAN

PROFESOR CO-GUÍA:
FRANCISCO FÖRSTER BURÓN

MIEMBROS DE LA COMISIÓN:
CLAUDIO GUTIÉRREZ GALLARDO
CESAR GUERRERO SALDIVIA

SANTIAGO DE CHILE
2023

Resumen

ALeRCE (Automatic Learning for the Rapid Classification of Events) es un broker astronómico chileno encargado de procesar y clasificar alertas astronómicas provenientes de telescopios como Zwicky Transient Facility (ZTF). Estas alertas astronómicas corresponden a cambios de naturaleza astrofísica detectados en el cielo. El enfoque de ALeRCE es estudiar objetos transitorios como núcleos galácticos activos, estrellas variables y la clasificación de cuerpos del Sistema Solar.

Para filtrar e identificar los datos de las alertas astronómicas, se realiza un *crossmatch*, el cual consiste en buscar y comparar objetos de catálogos astronómicos en posiciones específicas. No obstante, ALeRCE depende actualmente de un servicio externo para este fin, CDS X-Match, desarrollado por el Centro de datos astronómicos de Estrasburgo (*Centre de Données astronomiques de Strasbourg*, o CDS), lo que limita su control sobre los catálogos disponibles y la disponibilidad del sistema. Esta dependencia puede afectar el rendimiento y la eficiencia del agente de alertas, lo que motiva la necesidad de desarrollar un servicio interno de crossmatch para ALeRCE que se adapte a sus requerimientos, garantizando un mejor control sobre los datos y rendimiento del *pipeline*.

Dadas estas circunstancias, se propone en este trabajo la implementación de un servicio compuesto por una API que se conecta a una base de datos en Cassandra, donde se encuentre almacenado el catálogo astronómico CatWISE2020. Esta API permite realizar consultas HTTP para obtener resultados de realizar crossmatch entre una lista de coordenadas y el catálogo. Con esta solución, se busca mejorar el control sobre el proceso de crossmatch, e idealmente lograr una eficiencia mayor, utilizando Cassandra como sistema de base de datos y las metodologías Hierarchical Equal Area iso-Latitude Pixelation (HEALPix) y Hierarchical Triangular Mesh (HTM) para representar ubicaciones de manera eficiente.

Luego de realizar pruebas para verificar la precisión y eficiencia del sistema desarrollado se determinó que no hay diferencias significativas entre las técnicas de proyección. En ambos casos se logró un 100 % de coincidencia con los matches obtenidos al utilizar los mismos parámetros en CDS. Se observó que para conjuntos de más coordenadas, CDS es más eficiente que el sistema desarrollado. No obstante, al utilizar *threads* y trabajar con un menor número de coordenadas, se logró mejorar los tiempos de ejecución respecto a los de CDS. Los resultados logrados cumplen con los requisitos para la integración del servicio de crossmatch en la API de ALeRCE, no obstante, se sugiere realizar mejoras adicionales a los tiempos de ejecución, considerando el uso de hilos como una opción para garantizar respuestas más rápidas.

I believe we've reached the end of our journey. All that remains is to collapse the innumerable possibilities before us. Are you ready to learn what comes next? —Solanum

Agradecimientos

A mi padre por siempre insistirme en que aprenda todo lo que quiera porque el conocimiento nunca estorba, por enseñarme que nuestras fronteras están más allá de esta galaxia y por siempre tratar de ayudarme, incluso cuando se sentía sobrepasado. A mi madre, por darme cariño y espacio cuando lo necesité, por regalonearme en las buenas y en las malas, y por ser un apoyo incondicional a lo largo de todos estos años. A mi hermana y hermano, quiénes siempre estuvieron a mi lado, me dieron su confianza y me animaron con memes o con abrazos.

Gracias también a los otros miembros de mi familia, sobre todo a mi weli Hilda, quién siempre nos acogió en su casa los viernes, con la mejor comida del mundo, para poder disfrutar en familia. También a mi prima Alejandra, quién siempre ha estado dispuesta a ayudar a todos incondicionalmente, y a quién aún le debo sushi. No puedo dejar de agradecer a mis abuelos quiénes ya partieron. Gracias por ser un ejemplo de personas nobles y sencillas, quiénes con motivación y sus manos, eran capaces de arreglar todo en el mundo.

Quiero agradecer también a Aidan y Francisco, que me guiaron a lo largo de este proyecto, quienes siempre dedicaron un tiempo para mí, respondiendo mis dudas, planteando nuevas rutas para estudiar y depositaron su confianza en mí. También agradezco a Alberto, quién colaboró durante los comienzos de este proyecto, enseñándome muchas cosas que no conocía, y a Benjamín, con quién siempre conversábamos acerca de nuestros proyectos y me ayudó a resolver muchas dudas.

Gracias a los profesores que me otorgaron la sabiduría y herramientas necesarias para llegar a este punto. También muchas gracias a todos esos compañeros en la Universidad con quiénes compartí, celebré y sufrí. Quiero agradecer especialmente al grupo con quién compartí desde el primer día de plan común, mis estimados Inadaptados Sociales, y a todos mis compañeros del Team Michil, quiénes hicieron que el paso por un ramo que parecía aterrador, fuera la mejor experiencia de todas.

Por último, mis más sinceros y especiales agradecimientos a mis panas de mi Queer Crew, Mariana y Pablo, con quiénes compartimos chismes, tomamos tecito como todas unas señoras y me apoyaron en temas complejos que nadie más podría haber comprendido como ellos. Finalmente, mis últimos agradecimientos son para mis dos payasos, viajeros con los que recorrer el cosmos y guerreros de Sovngarde favoritos en todo el mundo, Hugo (The Ultimate Troll King) y Vicente (¿quién es Vicente?), quiénes han sido los mejores amigos que alguien podría querer, y que sin su amistad, jamás habría alcanzado este punto de mi vida.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación

Desde el comienzo de la historia, el cielo ha sido una fuente de fascinación y misterio para la humanidad. Los antiguos egipcios construyeron monumentos para alinearse con las estrellas, y los navegantes se han guiado por los astros para encontrar su camino en el vasto océano. Hoy en día, el campo de la astronomía ha evolucionado hasta convertirse en una disciplina científica que utiliza tecnología avanzada para explorar el universo y responder algunas de las preguntas más profundas de la humanidad.

En la era actual, la astronomía ha entrado en la fase del "Big Data" [18]. Con el desarrollo de telescopios cada vez más avanzados y poderosos, los astrónomos son capaces de recoger cantidades masivas de datos cada noche. El proyecto Pan-STARRS [4], por ejemplo, recopila alrededor de 4 terabytes de datos cada noche. Estos datos, acumulados a lo largo del tiempo, representan una fuente de información invaluable para el estudio del cosmos.

Sin embargo, la manipulación de estos enormes volúmenes de datos plantea nuevos desafíos. El almacenamiento, la gestión y el análisis de estos datos requieren nuevas soluciones y herramientas informáticas, dando lugar al surgimiento de la astroinformática, un campo que combina la astronomía con la ciencia de datos y el aprendizaje automático. Dentro de estos desafíos, uno de los más significativos es la necesidad de clasificar y catalogar de manera eficiente los objetos astronómicos identificados en estas grandes cantidades de datos. El proceso de emparejamiento o "crossmatching" es esencial para este propósito, ya que permite comparar y vincular los datos recopilados de diferentes observaciones y catálogos para un mismo objeto celeste.

En este contexto, surge ALeRCE (Automatic Learning for the Rapid Classification of Events), una iniciativa chilena que busca proporcionar una solución a estos desafíos. ALeRCE se centra en el procesamiento y clasificación de alertas astronómicas, con un enfoque especial en el estudio de objetos transitorios.

Actualmente ALeRCE depende de servicios externos para realizar el proceso de cross-

matching, CDS X-Match[1], sin embargo esto puede representar problemas de disponibilidad y limitaciones de datos. Existen muchas alternativas para realizar crossmaching[2], sin embargo, dada la importancia de este proceso, el requerimiento de ser capaces de poder procesar todas las alertas de ZTF en directo y la necesidad de tener un control completo sobre los datos utilizados, existe una clara motivación para que ALeRCE desarrolle su propio servicio de crossmatching, adaptado a sus necesidades específicas y capaz de manejar la cantidad de datos con la que trabaja. Esto no sólo permitiría mejorar la eficiencia y la fiabilidad del sistema, sino que también contribuiría a fortalecer la capacidad de Chile para liderar en el ámbito de la astroinformática.

1.2. Objetivos

El objetivo principal de esta propuesta de memoria es desarrollar un servicio de crossmatching astronómico mediante conesearch, que pueda ser administrado por el equipo del broker ALeRCE. Entre las técnicas más utilizadas para el crossmatching se encuentra el "conesearch". Un cono es una región circular en el cielo, definida por una posición del cielo y un radio alrededor de esa posición. Conesearch es una consulta de información relacionada con un cono dentro de un catálogo. En este caso, la posición del cielo se define mediante su ascensión recta y declinación, en el sistema de referencia celestial, y el radio posee una distancia arbitraria en unidades de medida de segundos de arco hasta minutos de arco.

Esto permitirá tener un control sobre los catálogos astronómicos utilizados y abordar de manera más eficiente posibles problemas de disponibilidad del servicio de crossmatching. Entre las prioridades de este servicio se encuentran la eficiencia y la escalabilidad, ya que se maneja un gran volumen de datos que deben procesarse rápidamente, para que los clientes del broker puedan acceder a esta información en tiempo real. La solución propuesta debe tener la capacidad de procesamiento adecuada para manejar el flujo de alertas esperado. Se espera que pueda manejar al menos 350 consultas por segundo para satisfacer las demandas del sistema. Estas consultas están principalmente enfocadas en realizar búsquedas de tipo "conesearch" con un radio de 1 arcosegundo. Sin embargo, también será necesario que la solución pueda manejar consultas de 1 arcominuto.

Para lograr el objetivo general, se han establecido los siguientes objetivos específicos:

1. Investigar los métodos de indexación HEALPix, HTM y Q3C, además de las estructuras de las tablas en Cassandra, para proponer modelos de tabla a utilizar. El objetivo es determinar preliminarmente, en función de las definiciones de cada método de indexación, cuáles serán los niveles de resolución óptimos de los píxeles (tamaño de los píxeles, lo que determina su cantidad) para las consultas a realizar.
2. Comprender la estructura de CatWISE2020 y desarrollar funciones de preprocesamiento que faciliten la indexación utilizando los diferentes métodos y múltiples niveles de resolución.

[1] http://cdsxmatch.u-strasbg.fr
[2] https://learn.astropy.org/tutorials/4-Coordinates-Crossmatch.html

3. Crear y poblar una tabla de muestra para realizar pruebas iniciales con un nivel de resolución específico.
4. Diseñar funciones para el pipeline del servicio que permitan el procesamiento completo de consultas para el conjunto de muestra.
5. Habilitar los endpoints de la API.
6. Verificar el funcionamiento del pipeline utilizando el conjunto de muestra.
7. Crear y poblar múltiples tablas para cada técnica de indexación, y para diversos niveles de resolución. Con esto, se llevará a cabo la misma consulta para cada técnica de indexación y para los diferentes niveles de resolución, lo que permitirá determinar el nivel óptimo para los radios a consultar, corroborando o refutando las predicciones hechas en el primer paso.
8. Poblar completamente las tablas correspondientes a los niveles de resolución óptimos para cada técnica de indexación.
9. Realizar pruebas de rendimiento del sistema para verificar que se alcance un mínimo de 350 consultas por segundo, y pruebas de exactitud, mediante la comparación con los resultados obtenidos al realizar consultas con los mismos parámetros en el sistema externo CDS. Si es necesario, hacer ajustes para alcanzar el rendimiento mínimo requerido.
10. Comparar y analizar los resultados obtenidos para las distintas técnicas de indexación, contrastando con los resultados de ejecutar las mismas consultas en el servicio externo CDS.

1.3. Estructura del documento

El presente documento se compone de 7 capítulos, además de una tabla de contenidos e índices para las tablas y figuras, con el propósito de facilitar la comprensión y el análisis de los resultados obtenidos durante el desarrollo de esta memoria.

El capítulo actual corresponde al Capítulo 1, donde se presentaron los objetivos y la motivación para el estudio del área, proporcionando información preliminar para contextualizar el problema.

El Capítulo 2 ofrece detalles sobre los antecedentes, conceptos y tecnologías necesarios para comprender las complejidades del problema y las particularidades de la solución desarrollada a lo largo de este trabajo.

En el Capítulo 3 se describen el diseño y el razonamiento detrás de las decisiones tomadas, así como el pipeline generado en las funciones de la API programada como solución.

El Capítulo 4 profundiza en el preprocesamiento de los datos requeridos para su uso en la base de datos, incluyendo la metodología para encontrar una estructura final de las tablas utilizadas. Además, se presentan los resultados de las pruebas preliminares realizadas para determinar la mejor estructura para las tablas en la versión final de la API.

El Capítulo 5 proporciona una explicación detallada de la implementación de la interfaz de la API, abordando el formato de entrada para las consultas y el formato de salida que devuelve. También se incluye una breve descripción de cómo realizar una llamada a la API para obtener respuestas y una explicación breve acerca del pipeline realizado en la API para procesar cada una de las consultas.

En el Capítulo 6 se presentan los experimentos realizados, junto con una descripción de los datos utilizados para llevarlos a cabo. Se incluyen figuras y tablas para comparar los resultados obtenidos con la situación actual.

Por último, el Capítulo 7 contiene las conclusiones derivadas de esta memoria. Se describen los objetivos cumplidos, así como aquellos que no se lograron completamente. Además, se sugieren posibles opciones para trabajos futuros basados en el desarrollo realizado en esta memoria.

Capítulo 2

Estado del arte

En este capítulo se presentan los temas relevantes para comprender el problema abordado, así como la solución que se desarrollará a lo largo de este trabajo.

2.1. Catálogos astronómicos

Gracias a los avances tecnológicos, se dispone de una vasta cantidad de información acerca de objetos astronómicos, recopilada por telescopios y satélites, y accesible en internet para su análisis. Esta información se agrupa en listados conocidos como catálogos astronómicos, que reúnen cuerpos celestes según características comunes, tales como el medio de detección, la morfología, el tipo de objeto, entre otras[1].

Muchos catálogos actuales proceden de encuestas o sondeos astronómicos, que mapean el cielo completo o una región particular de este. En el pasado, dichos catálogos se centraban en una gama específica de longitud de onda (rayos X, ultravioleta, infrarrojo, luz visible, etc.) para generar un catálogo de un tipo particular de objeto (estrellas, galaxias, cúmulos de estrellas). Sin embargo, en la actualidad, se está desarrollando un enfoque más inclusivo, que contempla el estudio de múltiples tipos de objetos y espectros de la luz.

En el contexto de este trabajo, resultan relevantes los sondeos Zwicky Transient Facility (ZTF) y Legacy Survey of Space and Time (LSST), así como el catálogo CatWISE2020 [15]. ALeRCE [6], el sistema que procesa los flujos de alertas provenientes del ZTF, aspira a convertirse en un broker que procese la información del LSST cuando este inicie sus operaciones. Además, CatWISE es el catálogo que se utiliza para realizar el crossmatching durante la implementación de este servicio.

Es relevante mencionar que estas herramientas y técnicas conforman parte del estado del arte en la observación y estudio de los cuerpos celestes. La adopción de nuevas tecnologías y la creciente capacidad de procesamiento y almacenamiento de datos han permitido avances significativos en la astronomía, posibilitando, por ejemplo, la realización de sondeos de gran

[1] http://astro.vaporia.com/start/astronomicalcatalog.html

alcance y la detección de eventos astronómicos transitorios. Esta combinación de avances tecnológicos y nuevos enfoques en la recolección y análisis de datos está transformando la comprensión del universo.

- CatWISE2020 es un catálogo que integra la información recabada por los telescopios WISE y NEOWISE, pertenecientes a la Administración Nacional de Aeronáutica y el Espacio de Estados Unidos (NASA). Estos telescopios realizaron un sondeo de objetos en el espectro infrarrojo durante el período comprendido entre los años 2009 y 2011. Posteriormente, en 2013, ejecutaron un sondeo de objetos cercanos a la Tierra bajo el nombre de NEOWISE (Near-Earth Objects). Este catálogo consolidado cuenta con un total de 1,890,715,640 observaciones. [5]
- Zwicky Transient Facility (ZTF) es un proyecto de sondeo astronómico dedicado a la observación de fenómenos astronómicos de corta duración, como novas, supernovas o tránsitos de asteroides frente a estrellas. Se lleva a cabo mediante el uso de 16 cámaras CCD (dispositivo de carga acoplada o *charge-coupled device*) en el telescopio Samuel Oschin, situado en el Observatorio Palomar, en California, Estados Unidos. Desde su inicio, ha proporcionado una gran cantidad de datos que ha contribuido al estudio de estos fenómenos transitorios [2].
- Legacy Survey of Space and Time (LSST)[2] es un proyecto de sondeo astronómico que se realizará mediante el telescopio Vera C. Rubin, ubicado en el norte de Chile. Este proyecto tiene como objetivo llevar a cabo un mapeo exhaustivo de la Vía Láctea y detectar eventos astronómicos transitorios. Se estima que, en los 10 años de operación del proyecto, será capaz de producir unos 500 petabytes de imágenes y datos del cielo. Cuando esté completamente operativo, LSST representará un hito en la astronomía moderna, proporcionando una cantidad sin precedentes de datos para el estudio del universo [10].

2.2. Crossmatching

El procesamiento de información proveniente de los diversos sondeos astronómicos requiere la identificación de un objeto dentro de uno o más catálogos, proceso conocido como crossmatching. Este consiste en cruzar la información contenida en las listas según las coordenadas del cuerpo celeste. El crossmatching permite a los astrónomos realizar un seguimiento de los objetos, facilitando el análisis de las variaciones en sus características a lo largo del tiempo.

Para realizar el crossmatch se utilizan consultas de conesearch. Para realizar el conesearch de manera más eficiente, se pueden utilizar índices espaciales, que son columnas de una tabla que utilizan estructuras de datos con métodos de búsqueda eficientes. Dentro de los índices espaciales, existen tres opciones de especial relevancia: HEALPix [8], Hierarchical Triangular Mesh (HTM) [17] y Quadtree Cube (Q3C) [11]. Estos índices espaciales trabajan con coordenadas esféricas, al igual que las coordenadas ecuatoriales utilizadas en astronomía.

[2]https://www.lsst.org/about

Este sistema consiste en dos valores, la ascensión recta y la declinación, que son equivalentes a las latitudes y longitudes geográficas.

Dos de estos índices, HTM y Q3C, comparten la característica de que las divisiones que se realizan en la esfera no tienen áreas iguales, a diferencia del último, HEALPix.

Hierarchical Triangular Mesh es una descomposición de una esfera unitaria que comienza desde un octaedro inscrito en su interior, definiendo áreas o píxeles que representan coordenadas en el espacio. El proceso de descomposición recursiva consiste en dividir cada cara triangular del poliedro en cuatro triángulos más pequeños. En esta malla triangular, cada triángulo se conoce como *trixel*, y los triángulos que componen el octaedro original son conocidos como trixel de nivel 0.

En la imagen izquierda de la figura 2.1 es posible ver el octaedro inicial correspondiente al trixel de nivel 0. Cada una de las figuras siguientes corresponde a un nivel más profundo de resolución, donde a través del patrón de recursividad descrito en la imagen de la derecha, se alcanza una esfera más definida formada por una mayor cantidad de triángulos de pequeño tamaño. El nivel 5 corresponde a 8192 triángulos.

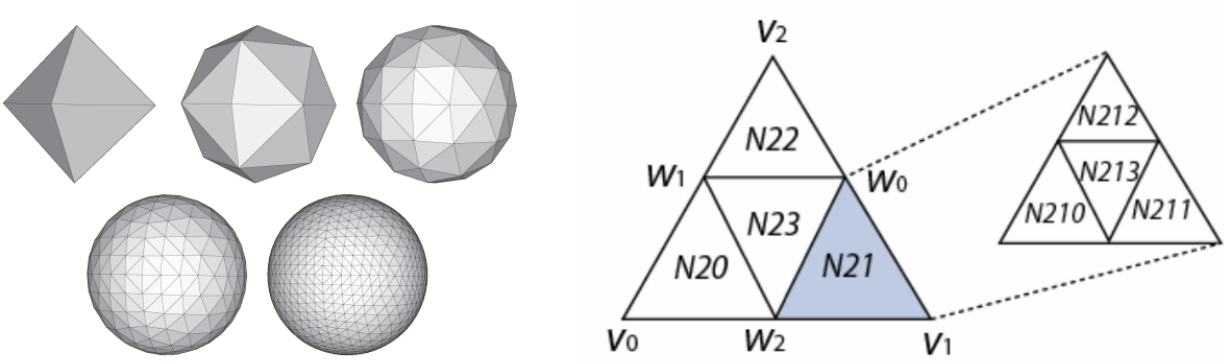

Figura 2.1: Representación de descomposición de la esfera en HTM [17].

Por su parte, Q3C surge como una respuesta a problemas de rendimiento de HTM cuando el nivel de los trixels es muy alto. Esta técnica divide la esfera inscribiendo un cubo en su interior, creando un quadtree en las seis caras del cubo, lo que permite una descomposición recursiva de la esfera. Esto se representa en la figura 2.2, donde se puede observar que el esquema de división parte de un cubo inflado hasta alcanzar la forma de una esfera, luego por cada cara se realiza una división trazando en sus centros líneas paralelas a las aristas, con lo que se alcanza la esfera de la izquierda. Luego con una nueva subdivisión se alcanza la esfera de la derecha.

Por último, HEALPix (Hierarchical Equal Area isoLatitude Pixelisation of a 2-sphere)[3] es una técnica que divide la esfera utilizando cuadriláteros curvilíneos que se dividen recursivamente en cuatro secciones, donde todas las secciones comparten un área igual. Las áreas o píxeles que genera son paralelos entre sí, ya que se distribuyen a lo largo de iso latitudes o líneas de latitud constante.

Para ejemplificar el funcionamiento de HEALPix, se presenta la figura 2.3. En la imagen

[3]https://healpix.jpl.nasa.gov/

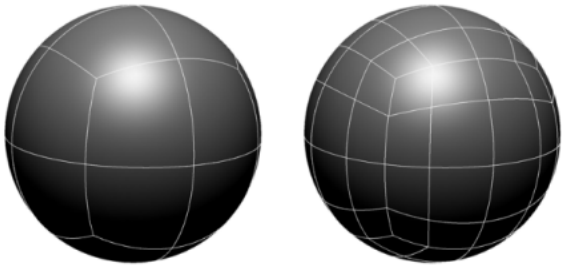

Figura 2.2: Representación de proyección en Q3C [11].

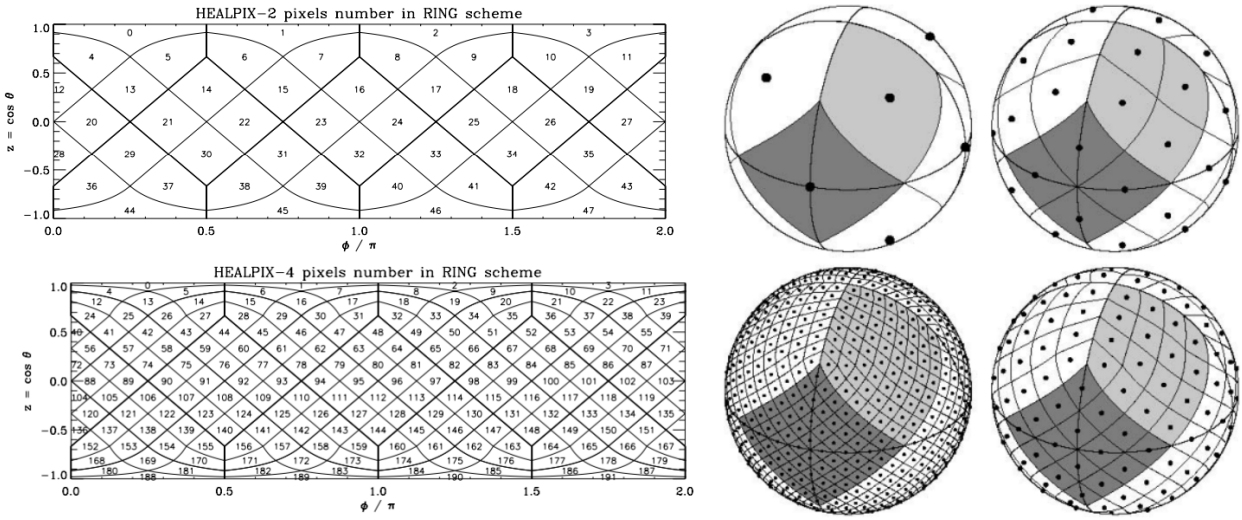

Figura 2.3: Representación de pixelización HEALPix [8].

de la derecha se presenta el patrón de subdivisión seguido por HEALPix, destacando en la primera esfera en color gris claro el área correspondiente a uno de los 8 píxeles base polares, mientras que en color gris oscuro el área de uno de los 4 píxeles base ecuatoriales. En el resto de las esferas se representa la subdivisión recursiva. En la imagen de la izquierda se puede observar una subdivisión de los píxeles y el patrón de indexación de cada píxel.

2.3. ALeRCE

Automatic Learning for the Rapid Classification of Events (ALeRCE) [6] es un broker astronómico chileno que procesa y clasifica alertas astronómicas provenientes de telescopios como ZTF y, en un futuro próximo, LSST. Un broker astronómico es una plataforma o servicio que actúa como intermediario entre la información recopilada por telescopios, astrónomos e investigadores, facilitando el acceso y análisis a datos astronómicos. Las alertas astronómicas[4] refieren a cambios detectados en el cielo durante un sondeo, cuyo origen es astrofísico. Estas alertas contienen información sobre lo que se detecta en la zona de alerta.

[4]https://zwickytransientfacility.github.io/ztf-avro-alert/schema.html

El enfoque principal de ALeRCE radica en el estudio de objetos transitorios[5], núcleos galácticos activos y estrellas variables. También tiene un objetivo secundario que consiste en la clasificación de objetos del Sistema Solar. Para alcanzar estos objetivos, ALeRCE emplea un proceso que involucra la ingestión de datos en tiempo real, agregación, crossmatching, clasificación mediante aprendizaje de máquinas y visualización de los flujos de alertas de ZTF.

Debido al volumen de datos que ALeRCE procesa cada noche, aproximadamente 300 mil alertas por noche del flujo público de ZTF, es imprescindible que todos los pasos que se realizan en el pipeline para el análisis de la información sean eficientes. Esto implica que el proceso de crossmatching debe ser capaz de procesar información rápidamente para mantener la eficiencia general del sistema. La eficiencia en el manejo de estos enormes volúmenes de datos es un desafío constante en la era de la "big data" en la astronomía.

ALeRCE emplea una secuencia de procedimientos que abarcan la captura de datos en tiempo real, operaciones de consolidación, emparejamiento, clasificación mediante aprendizaje automático y visualización de los flujos de alertas de ZTF, cuyos distintos procesos pueden observarse en la figura 2.4. Este se puede leer de manera más detallada en su sitio web[6]. El presente trabajo se enfoca en generar una alternativa al proceso de crossmatching.

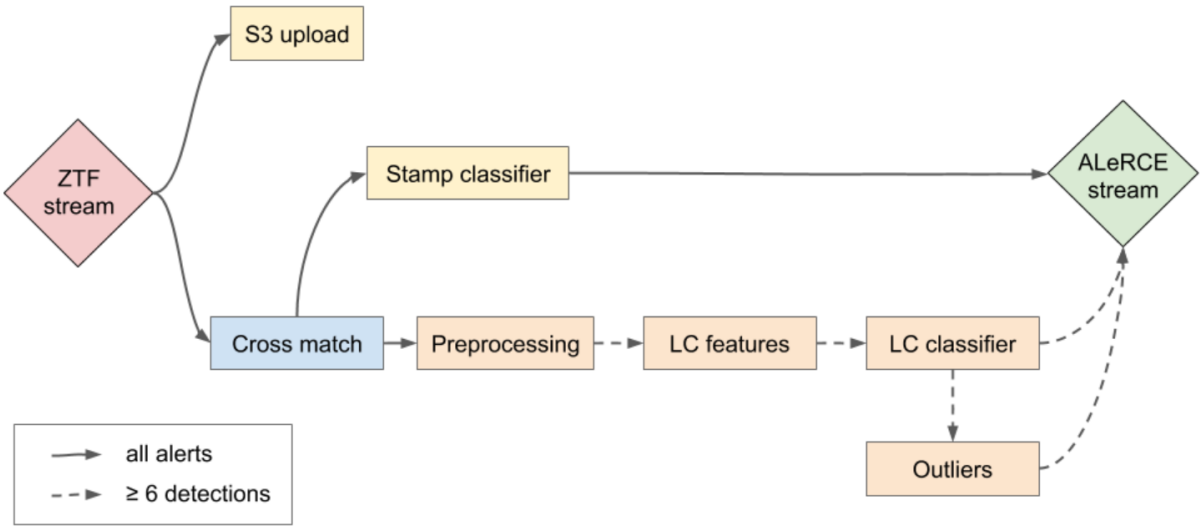

Figura 2.4: Diagrama del pipeline usado por ALeRCE *Fuente:* `https://alerce.science/alerce-pipeline/`

2.4. CDS X-Match

Actualmente, ALeRCE utiliza un servicio externo para llevar a cabo el proceso de crossmatching. Este servicio se conoce como CDS X-Match y depende del Strasbourg Astronomical Data Center (CDS) [7], de donde derivan sus siglas. CDS X-Match permite a los usuarios

[5]https://astrobites.org/2022/10/30/guide-to-transient-astronomy/
[6]https://alerce.science/alerce-pipeline/

realizar un crossmatch de catálogos astronómicos de manera eficiente, incluso para catálogos que contienen miles de millones de objetos, o cargar su propio archivo de coordenadas[7]. Este servicio se puede utilizar a través de su interfaz web o su API XMatch[8].

Aunque la información detallada acerca de su algoritmo de crossmatching no está disponible en su documentación, se sabe que CDS X-Match incorpora índices de HEALPix como alternativa a las áreas de búsqueda. Esta implementación permite una eficiente localización y comparación de objetos astronómicos.

El uso de CDS X-Match como servicio de crossmatching presenta algunas limitaciones. Estas incluyen la falta de control sobre la disponibilidad del servicio en caso de problemas técnicos, la falta de control sobre los catálogos disponibles o su actualización, y una restricción en el tiempo de ejecución de las operaciones. Estos problemas hacen que dependa en gran medida de la estabilidad y rendimiento del servicio CDS X-Match, lo que puede tener impacto en la eficiencia y la eficacia del análisis astronómico que realiza ALeRCE. Por lo tanto, es vital explorar alternativas que permitan mantener la eficiencia del proceso de crossmatching al tiempo que proporcionen un mayor control sobre los aspectos críticos del mismo.

2.5. NoSQL

Las bases de datos NoSQL, también conocidas como bases de datos no relacionales, se caracterizan por su esquema flexible, en contraposición a los esquemas fijos de las bases de datos SQL. Este tipo de bases de datos pueden manejar grandes volúmenes de datos y se representan de diversas formas, incluyendo bases de datos de gráficos, de clave-valor, de documentos, entre otras.

Estas bases de datos están sujetas al Teorema CAP [12], también conocido como la conjetura de Brewer, que establece que un sistema distribuido no puede garantizar simultáneamente tres propiedades fundamentales:[9]

- Consistencia (Consistency): Se refiere a que cualquier lectura debe recibir la escritura más reciente o arrojar un error.
- Disponibilidad (Availability): Garantiza que cualquier solicitud debe recibir una respuesta, sin errores, aunque no se asegura que contenga los datos más recientes.
- Tolerancia al particionado (Partition Tolerance): Implica la capacidad del sistema para continuar operando correctamente incluso en caso de fallos en la red.

Otra ventaja clave de las bases de datos NoSQL sobre las SQL es la capacidad para escalar horizontalmente. Esto significa que, si se agregan más computadoras al sistema, estas compartirán la carga de procesamiento, mejorando así el rendimiento del sistema.

[7] http://cdsxmatch.u-strasbg.fr/xmatch/doc/table-upload.html
[8] http://cdsxmatch.u-strasbg.fr/xmatch/doc/
[9] https://www.ibm.com/topics/cap-theorem

Finalmente, es importante mencionar que debido a la flexibilidad de los esquemas en las bases de datos NoSQL, las operaciones que se pueden realizar con sus datos no pueden alcanzar un nivel de complejidad alto, al contrario de lo que sucede con las operaciones realizables en una base de datos SQL.

2.6. Apache Cassandra

Apache Cassandra es un sistema de gestión de bases de datos NoSQL distribuido, diseñado para manejar grandes cantidades de datos a través de muchos servidores. Proporciona alta disponibilidad con tolerancia a fallos, siendo ideal para aplicaciones que no pueden permitirse tiempo de inactividad. Cassandra ofrece una robusta replicación de datos, lo que garantiza que la información no se perderá si ocurre una falla [13].

Esta base de datos se desarrolló originalmente en Facebook para alimentar su función de búsqueda de Inbox y luego fue liberada como un proyecto de código abierto. Cassandra es ampliamente utilizada por una variedad de organizaciones que manejan grandes volúmenes de datos, incluyendo GitHub, Netflix y Twitter.

El diseño de Cassandra se basa en la premisa de que las fallas pueden ocurrir en cualquier momento y el sistema debe estar preparado para ello. Su modelo de datos ofrece la conveniencia de las columnas, las filas y las tablas de los RDBMS tradicionales, pero lo hace en una forma que está diseñada para escalar horizontalmente, a través de muchos servidores, y no tiene un único punto de falla.

En términos del Teorema CAP, Cassandra se enfoca en la disponibilidad y la tolerancia al particionado, con una consistencia eventual. Esto significa que las lecturas podrían reflejar una escritura anterior o más reciente, pero eventualmente se sincronizarán. Esto es adecuado para muchas aplicaciones de la vida real donde un pequeño retraso es aceptable.

El modelo de datos de Cassandra es una tabla de hash particionada por la clave de fila, con agrupamiento de filas ordenadas por la clave de columna.[10] Las tablas pueden crearse en tiempo de ejecución, y las filas y columnas se pueden agregar dinámicamente, lo que proporciona mucha flexibilidad.[11]

Apache Cassandra ha demostrado ser una solución sólida y efectiva para administrar grandes conjuntos de datos, con un rendimiento que escala linealmente, proporcionando latencias predictivas a medida que se agregan más nodos al clúster. Como tal, se ha establecido como una parte importante en el estado del arte de las bases de datos NoSQL.

[10] https://docs.datastax.com/en/dseplanning/docs/data-model.html
[11] https://www.datastax.com/blog/why-does-scalability-matter-and-how-does-cassandra-scale

Capítulo 3

Solución

En esta sección se proporciona una descripción general del problema que impulsa este trabajo, así como una visión general de la solución propuesta y las tecnologías empleadas.

3.1. Descripción del problema actual

Se han identificado múltiples problemas al utilizar el servicio externo existente. Por un lado, la disponibilidad inconsistente de CDS para realizar el crossmatching entre los objetos de las observaciones y el catálogo resulta en una demora en el proceso de clasificación. Además, la cantidad de peticiones que el broker realiza al servicio para llevar a cabo el crossmatch es alta, lo que puede sobrecargar el servicio debido al gran volumen de solicitudes. Por último, al tratarse de un servicio externo, el equipo de ALeRCE no tiene control sobre los catálogos disponibles ni la capacidad de reiniciar el servicio en caso de problemas. Ante estas problemáticas, se plantea la creación de un nuevo servicio en forma de API, que permita realizar el crossmatching actualmente realizado en CDS, abordando de manera eficiente y escalable estas dificultades al generar consultas de conesearch en diferentes catálogos.

3.2. Solución propuesta

La solución propuesta consiste en implementar una API que se pueda alojar en uno de los servidores de ALeRCE. Esta API será responsable de recibir los datos de las alertas de ZTF y, en el futuro, de LSST, y proporcionar resultados utilizables en el resto del flujo de trabajo de ALeRCE. Además de la API principal que maneja las consultas, se desarrollará un conjunto de funciones para el preprocesamiento de datos específicos del catálogo CatWISE2020, con el objetivo de que estas funciones sean aplicables a otros catálogos. Dado que hay una variedad de catálogos astronómicos y no existe un formato estándar para todos ellos, se requerirá modificar el formato de otros catálogos para que sean compatibles con el de CatWISE2020, de manera que puedan ser indexados correctamente. Por último, se creará el modelo de tablas en Cassandra utilizando índices optimizados para cada algoritmo de indexación utilizado, y

se cargarán los datos de CatWISE2020 en estas tablas. En el futuro, este proceso podrá repetirse para generar nuevas tablas y cargar los datos de otros catálogos.

La solución propuesta se divide en dos etapas principales, las cuales son abordadas en detalle en los próximos dos capítulos:

En la primera etapa, se lleva a cabo el preprocesamiento de los datos, se analiza la estructura de la tabla en la base de datos de Cassandra y se define una metodología para determinar los índices óptimos mediante la pixelización de las tablas utilizadas en la versión final desarrollada. Además, se presentan los resultados de experimentos preliminares realizados utilizando una versión inicial del pipeline de la API.

La segunda etapa se centra en el desarrollo de la API y los endpoints correspondientes. Se proporciona información detallada sobre las funciones desarrolladas para realizar consultas de crossmatching en este nuevo servicio. Se explica cómo se implementa la lógica de búsqueda y se describe el formato de entrada y salida de las consultas.

En resumen, la primera etapa se enfoca en el procesamiento y análisis de los datos, así como en la determinación de los índices óptimos, mientras que la segunda etapa se centra en la implementación de la API y la funcionalidad de consulta.

Durante el desarrollo de este trabajo, se emplean diversas herramientas que resultan fundamentales para alcanzar los objetivos propuestos. A continuación, se detallan las principales herramientas utilizadas:

- **Cassandra:** Se utiliza la base de datos distribuida Cassandra debido a su capacidad de manejar grandes volúmenes de datos y su alto rendimiento en entornos distribuidos. La elección de Cassandra permite almacenar y gestionar eficientemente los datos astronómicos utilizados en el proceso de crossmatching.
- ***dsbulk*:** Se utiliza la herramienta *dsbulk* para cargar los datos a la base de datos de Cassandra. Esta es una herramienta de línea de comandos diseñada para facilitar la carga masiva de datos en Cassandra de manera rápida y eficiente. Su uso permite cargar los datos del catálogo CatWISE2020 en la base de datos de manera eficiente y escalable.[1]
- **FastAPI:** Se utiliza el framework FastAPI para implementar la API. FastAPI es un framework moderno y de alto rendimiento que permite crear servicios web de forma rápida y sencilla. Proporciona una interfaz intuitiva para definir los endpoints de la API, manejar las solicitudes y respuestas HTTP, y facilitar la interacción con la base de datos.
- **Docker:** Se utiliza Docker para *containerizar* la API y facilitar su despliegue en el servidor de ALeRCE. Docker permite empaquetar la aplicación junto con todas sus dependencias en un contenedor, lo que garantiza la portabilidad y disponibilidad del servicio en diferentes entornos.
- **Python:** Se utiliza el lenguaje de programación Python debido a su amplia gama de bibliotecas y su facilidad para el procesamiento de datos. Python permite implementar

[1] https://docs.datastax.com/en/dsbulk/docs/reference/dsbulk-cmd.html

de manera eficiente la lógica del sistema, realizar cálculos complejos y manipular los datos necesarios para el funcionamiento de la API. Durante el desarrollo del trabajo, se emplean las siguientes bibliotecas:

- **Astropy:** La biblioteca Astropy es una poderosa herramienta en Python utilizada para el manejo y procesamiento de datos astronómicos. Proporciona una amplia gama de funcionalidades para trabajar con unidades y sistemas de coordenadas astronómicas, realizar cálculos relacionados con la astronomía, y manipular datos de diferentes formatos comunes en la comunidad astronómica. Astropy simplificó las tareas relacionadas con las coordenadas astronómicas utilizadas en este trabajo, permitiendo la conversión entre diferentes sistemas de coordenadas, la manipulación de coordenadas celestiales y la realización de cálculos precisos en el contexto astronómico.[2]
- **Healpy:** La biblioteca Healpy[3] fue utilizada para trabajar con los índices de HEALPix en el contexto de la indexación astronómica. Healpy es una biblioteca de Python que proporciona herramientas para trabajar con mapas esféricos y realizar operaciones en los índices de pixelización de HEALPix. Estos índices son ampliamente utilizados en la astronomía y permiten dividir una esfera en regiones discretas para facilitar el análisis y la búsqueda espacial eficiente. Healpy facilita el manejo de los índices de pixelización y las operaciones relacionadas, como la asignación de objetos astronómicos a los píxeles correspondientes.
- **HTMpy:** La biblioteca HTMpy[4] se utiliza para trabajar con la malla triangular jerárquica (HTM) en el contexto de la indexación espacial. HTMpy es una biblioteca de Python que proporciona funcionalidades para realizar búsquedas y consultas espaciales eficientes utilizando la malla triangular jerárquica. HTM es una estructura de datos utilizada en astronomía para dividir el espacio en pequeñas celdas triangulares, lo que permite realizar búsquedas rápidas y eficientes basadas en la ubicación espacial. HTMpy simplifica la implementación de consultas espaciales y contribuyó a la eficiencia en el proceso de cross-matching.
- **Cassandra Python Driver:** Cassandra Python Driver[5] se utiliza en este proyecto como una herramienta fundamental para la conexión y consulta de la base de datos Apache Cassandra. Con el fin de aprovechar las capacidades de almacenamiento y procesamiento distribuido de Cassandra, se emplea esta biblioteca que proporciona una interfaz de programación en Python. El Cassandra Python Driver permite establecer conexiones eficientes con el clúster de Cassandra y ejecutar consultas de manera óptima, aprovechando características como el equilibrio automático de carga y la replicación de datos. Con esta poderosa herramienta, se logra una integración fluida entre el análisis de datos en Python y el almacenamiento y recuperación de información en Apache Cassandra.
- **Requests[6]:** La biblioteca Requests se utiliza para realizar experimentos y pruebas durante el desarrollo de la solución. Requests es una biblioteca de Python

[2]https://docs.astropy.org/en/stable/index.html
[3]https://healpy.readthedocs.io/en/latest/
[4]https://hmpty.readthedocs.io/en/master/
[5]https://docs.datastax.com/en/developer/python-driver/index.html
[6]https://requests.readthedocs.io/en/latest/

que permite enviar solicitudes HTTP de manera sencilla y eficiente. Proporciona una interfaz amigable para realizar solicitudes GET, POST y otras operaciones comunes en servicios web. En este caso, se utilizó para interactuar con servicios externos y realizar solicitudes de datos necesarios para los experimentos.

- **Uvicorn:** Uvicorn[7] es un servidor web de alto rendimiento utilizado para servir aplicaciones web desarrolladas en Python. Este servidor se basa en la biblioteca de red ASGI[8] (Asynchronous Server Gateway Interface), que permite la implementación de aplicaciones web asíncronas y escalables. Uvicorn se utiliza en este trabajo para desplegar y gestionar la API desarrollada con FastAPI. Proporciona una configuración sencilla y eficiente para manejar las solicitudes HTTP entrantes y garantizar una comunicación rápida y segura entre el cliente y el servidor.
- **Pandas:** La biblioteca Pandas[9] se utiliza para el procesamiento y manipulación de datos tabulares. Esta herramienta permite realizar operaciones de limpieza, filtrado y transformación de los datos de los catálogos astronómicos, preparándolos para su indexación y consulta.
- **Numpy:** Se emplea la biblioteca NumPy[10] para el procesamiento numérico y el cálculo eficiente de operaciones matemáticas. NumPy resulta fundamental para realizar cálculos complejos en el proceso de indexación y para optimizar el rendimiento de las consultas de crossmatching.

Estas herramientas proporcionan la base tecnológica necesaria para implementar la solución propuesta, permitiendo el procesamiento eficiente de los datos, la gestión de la base de datos y la disponibilización de la API en el servidor de ALeRCE. Su integración y correcto uso resultan cruciales para lograr los objetivos planteados en este trabajo.

Inicialmente se planteó trabajar tres técnicas de proyección distintas, las cuáles incluían la técnica de proyección de coordenadas astronómicas Q3C. Sin embargo, no existe disponible una implementación en Python que permita realizar consultas de píxeles correspondientes a un área específica utilizando un punto y radio, sin cambiar la base de datos a utilizar por una SQL. La alternativa más cercana fue QLSC[11], que utiliza SQLite. Se asume que el rendimiento de una base de datos SQL debiese ser inferior al esperado para una base de datos NoSQL, dada la naturaleza y cantidad de los datos utilizados, por lo que utilizar esta implementación quedaría descartado. La última opción sería reimplementar las funciones de Q3C, las cuáles fueron originalmente escritas en C. Sin embargo, esto requeriría más tiempo del que se dispone, y por tanto, se desechó la opción de utilizar Q3C como una técnica de proyección durante este trabajo .

Para cerrar el capítulo se describirá a un alto nivel el proceso planteado por la API a desarrollar que resuelve una consulta de crossmatch.

Para comenzar es esencial identificar cuáles son las entradas para un crossmatch. Estas corresponden a un conjunto de coordenadas y un radio. Las coordenadas utilizadas se

[7] https://www.uvicorn.org
[8] https://asgi.readthedocs.io/en/latest/specs/main.html
[9] https://pandas.pydata.org/docs/
[10] https://numpy.org/doc/
[11] https://qlsc.readthedocs.io/en/latest/index.html

representan en una esfera unitaria. Por cada coordenada se debe generar un área circular correspondiente al área entre el punto correspondiente a la coordenada y el radio. Lo que interesa de esta área es encontrar todos los objetos del catálogo que se encuentren en esta área.

Para lograr encontrar rápidamente los objetos dentro de un área se utilizan las técnicas de pixelización, en este caso HEALPix y HTM. Con estas, es posible asignar a cada objeto del catálogo un índice, el cual corresponde a un número de un píxel que describe un área específica. Estos píxeles dependen de un valor de resolución, donde mientras más resolución hay, más pequeños son los píxeles, lo que permite describir una zona de forma más precisa, y por tanto hay más píxeles.

Ambas técnicas de pixelización tienen implementados métodos que permiten, de acuerdo con un área y nivel de resolución, encontrar el conjunto de todos los píxeles que cubren el área. Con esto es posible realizar consultas a la base de datos de acuerdo con estos píxeles para obtener los objetos incluidos en estos píxeles y que están incluidos en el área de consulta.

Sin embargo, el tamaño de los píxeles puede ser más grande que el área de interés, y en algunos casos se incluirían objetos cuya distancia a la coordenada de match sea mucho mayor al radio, por lo que es necesario realizar un proceso de filtrado, donde se calculan las distancias de los objetos resultantes de las queries a la base de datos y se mantienen solo aquellos cuya distancia sea menor al radio de búsqueda usado.

A los objetos resultantes del filtrado se les agrega a la información proveniente de la base de datos tres columnas, correspondientes a las dos componentes de la coordenada con la que matchearon, y la distancia a estas. Finalmente, esto se devuelve de acuerdo con algún formato indicado por la consulta de crossmatch.

3.3. Trabajos previos

Se ha realizado previamente un trabajo por Alejandra Alarcón en su memoria titulada *"Servicio de Crossmatching de Objetos Astronómicos"* [1], donde desarrolló una API para abordar la misma problemática utilizando MongoDB como sistema de base de datos y los índices geoespaciales de MongoDB para mejorar la eficiencia en las consultas geográficas.

En comparación con los resultados de CDS, la API desarrollada en MongoDB presenta un pequeño porcentaje de diferencia, aproximadamente del 1 %, que se encuentra principalmente en los bordes de las consultas. Esta discrepancia puede resolverse ampliando los radios de las consultas para incluir los objetos en los bordes. Además, en pruebas de velocidad de respuesta, esta API resultó más eficiente que CDS para muestras de datos más pequeñas con radios menores a 8 arcosegundos.

Sin embargo, se plantea la pregunta de por qué es necesario investigar otro sistema de base de datos y metodologías de indexación. Una de las razones radica en resolver las diferencias en los resultados, ya que, aunque sean de aproximadamente el 1 %, podrían tener impactos imprevistos. Asimismo, es crucial explorar alternativas que optimicen los tiempos de respuesta

para lograr el mejor sistema posible. Por último, la API desarrollada por Alejandra no abordó radios de 1 arcominuto, lo cual representa una necesidad a trabajar dadas las necesidades de ALeRCE. Siguiendo la tendencia de los resultados de esta API, los tiempos alcanzarían valores sobre los 450 segundos para 50,000 puntos e incluso ser mucho más elevados.

Capítulo 4

Preprocesamiento

En este capítulo, se presenta con mayor detalle la primera etapa del trabajo, que se enfoca en el preprocesamiento de los datos del catálogo utilizado y en la construcción de la base de datos correspondiente. Además, se describen las pruebas preliminares realizadas con una versión inicial de la API desarrollada a partir de los detalles expuestos en el capítulo siguiente. El objetivo principal de esta etapa es realizar el adecuado procesamiento de los datos, incluyendo la conversión del formato .tbl a .csv, para que puedan ser insertados en la base de datos mediante el uso de dsbulk.

En primer lugar, se proporciona una descripción detallada del catálogo utilizado, CatWISE2020, y se explica el análisis realizado para decidir la estructura final de la tabla de la base de datos. También se detalla el proceso llevado a cabo para preparar los datos y se describe el método utilizado para realizar la conversión del formato .tbl a .csv, que es necesario para la carga de los datos en la base de datos mediante dsbulk. Asimismo, se presentan los experimentos realizados y sus resultados, lo cual es fundamental para evaluar el éxito del proceso de preprocesamiento de los datos.

En resumen, este capítulo se centra en la descripción de la primera etapa del trabajo, que abarca el preprocesamiento de los datos del catálogo, la construcción de la base de datos y las pruebas preliminares. El objetivo principal es lograr el adecuado procesamiento de los datos y la preparación de estos en formato csv para su carga en la base de datos utilizando dsbulk.

4.1. Catálogo CatWISE2020

CatWISE2020 es un catálogo completo del cielo que recopila datos de WISE y NEOWISE en las longitudes de onda de 3.4 µm y 4.6 µm. Esta versión del catálogo cuenta con dos años adicionales de objetos observados en comparación con la versión preliminar que abarca el período de 2010 a 2016. En total, CatWISE2020 contiene 2.232.515.025 objetos, que incluyen tanto los del catálogo principal como los rechazados. La diferencia entre ellos radica en las

condiciones que deben cumplir.[1]

Para ser considerado en el catálogo principal, un objeto debe cumplir las siguientes condiciones[2]:

- Debe ser "primario" y estar ubicado en la sección de la grilla donde la fuente está más alejada de los bordes.
- Debe tener una relación señal/ruido (SNR, por sus siglas en inglés) igual o mayor a 5 para W3 (longitud de onda de 3.4 µm), sin artefactos identificados (un valor de 0 para "ab flags" en la columna 1784 correspondiente a W1).

En caso de no cumplir estas condiciones, un objeto puede ser incluido si cumple lo siguiente:

- Tiene una relación señal/ruido (SNR) igual o mayor a 5 para W2 (longitud de onda de 4.6 µm), sin artefactos identificados (un valor de 0 para "ab flags" en la columna 178 correspondiente a W2).

El conjunto de datos original se divide en 36.481 archivos. La mitad de estos archivos pertenecen al catálogo principal, denominado "catalog", que contiene 187 columnas, mientras que la otra mitad corresponde a archivos rechazados, conocidos también como "rejected", que tienen 188 columnas, donde en promedio cada archivo contiene la información de 80.000 fuentes. Cada archivo tiene el formato tbl.gz, que es un archivo ASCII comprimido con formato de tabla. Para este trabajo, se seleccionaron exclusivamente los archivos del conjunto "catalog" de CatWISE2020. Este conjunto representa un total de 1.890.715.640 objetos observados y registrados por NEOWISE y WISE. Estos objetos cumplen con los criterios y condiciones previamente establecidos, lo que garantiza la integridad y calidad de los datos utilizados en el estudio.

Inicialmente, se realizó un análisis del formato de las tablas para poder convertirlo en una estructura de datos manejable. Para lograr esto, se desarrolló un programa en Python que leyera cada línea de un archivo, separando los datos de cada fila de la tabla y almacenándolos en un arreglo de strings.

Una vez procesada cada fila de un archivo, se construyó un Dataframe a partir de los arreglos, lo que permitió transformar los archivos .tbl en .csv. Estos archivos .csv fueron posteriormente insertados en la base de datos de Apache Cassandra.

Para llevar a cabo esta inserción eficientemente, se utilizó la herramienta dsbulk. Dsbulk es una herramienta de carga de datos masiva que facilita la inserción de grandes volúmenes de datos en una tabla de Cassandra desde archivos csv. Esta herramienta permite un proceso rápido y eficiente, optimizando la inserción de datos en la base de datos.

[1] https://portal.nersc.gov/project/cosmo/data/CatWISE/PrelimREADME.txt
[2] https://portal.nersc.gov/project/cosmo/data/CatWISE/2020README.txt

Dentro de los primeros experimentos realizados, se encontraron un par de inconsistencias con respecto a los resultados obtenidos al realizar consultas utilizando el servicio generado en este trabajo y al servicio externo CDS bajo los mismos parámetros, de los cuales una de estas derivaría en añadir un paso extra dentro de la fase de preprocesamiento, dado que los resultados entregados por el servicio propio bajo el método de indexación HEALPix entregaba una menor cantidad de objetos en el área cercana a los bordes de las consultas.

Para comprender mejor estos resultados, es importante entender el concepto de "tiles" en CatWISE2020. En este contexto, un "tile" se refiere a una región específica del cielo que ha sido dividida y asignada a una sección del catálogo. Estas divisiones se realizan en función de las coordenadas de ascensión recta (RA) observadas en un determinado período de tiempo.

Para facilitar el manejo y análisis de los datos, CatWISE2020 divide una imagen completa del cielo en múltiples "azulejos" o "tiles", similar a un mosaico. Cada uno de estos "tiles" representa una sección más pequeña y manejable del cielo.

Cada "tile" está identificado por un código alfanumérico único, como "0000m016" mencionado anteriormente. Esta identificación codifica la ubicación específica del "tile"en el cielo. La información sobre los desplazamientos y correcciones para cada "tile" se registra en la tabla mencionada anteriormente, llamada "CatWISE2020_Table1_20201012.tbl".

En el caso de CatWISE2020, se descubrió que los resultados de las consultas variaban entre el servicio generado y el servicio externo CDS. Esto se debió en parte a que los "tiles" en CatWISE2020 tenían pequeñas desviaciones astrométricas. Estas desviaciones se deben a que los "epoch coadds" utilizados en el pipeline preliminar de CatWISE incluyen un ajuste al sistema de coordenadas mundial para los "epoch" de AllWISE, mientras que los "epoch coadds" utilizados en CatWISE2020 no incluyeron este ajuste.

Los desplazamientos astrométricos medianos para la posición y el movimiento propio de cada "tile" se proporcionan en la tabla mencionada. Estos desplazamientos deben aplicarse a los valores del catálogo para corregir las desviaciones. Por ejemplo, se menciona un objeto llamado "0000m016_b0-001851" medido en el "tile" 0000m016 en el catálogo CatWISE2020. Los valores corregidos se obtienen sumando las correcciones sistemáticas correspondientes a ese "tile" a los valores del catálogo.

En resumen, las inconsistencias encontradas durante los experimentos se relacionan con las desviaciones astrométricas presentes en los "tiles" de CatWISE2020. Estas desviaciones se corrigieron mediante la aplicación de las correcciones proporcionadas en la tabla correspondiente a cada "tile". Esto asegura la integridad y calidad de los datos utilizados en el estudio.

Se solicitó la opinión de los astrónomos del equipo de ALeRCE para determinar las columnas más relevantes que se incluirían en el conjunto de datos. Este proceso permitió reducir el número de columnas de 186 a 20, seleccionando cuidadosamente aquellas que son más significativas para el trabajo en cuestión.

Las columnas de las tablas del catálogo a utilizar corresponden a las presentadas en la tabla 4.1, cuyas descripciones fueron extraídas de documentación del catálogo[3] :

[3]https://portal.nersc.gov/project/cosmo/data/CatWISE/2020cwcat.sis20200318.txt

Tabla 4.1: Nombres y descripciones de las columnas de CatWISE2020 utilizadas en la API.

Nombre de columna	**Descripción de columna**
source id	tile name + código de procesamiento + índice wphot
ra	ascensión recta (frame ICRS)
dec	declinación (frame ICRS)
sigra	incerteza de ra
sigdec	incerteza de dec
source name	designación hexasegimal del objeto
w1mag	magnitud de apertura estándar con corrección, banda 1
w1sigm	incertidumbre de la magnitud de apertura estándar, banda 1
w2mag	magnitud de apertura estándar con corrección, banda 2
w2sigm	incertidumbre de la magnitud de apertura estándar, banda 2
w1flg	flag de apertura instrumental estándar, banda 1
w2flg	flag de apertura instrumental estándar, banda 2
w1k	índice de variabilidad de Stetson, banda 1[16]
w2k	índice de variabilidad de Stetson, banda 2[16]
w1mlq	-ln(1 - P(chi^2)), banda 1
w2mlq	-ln(1 - P(chi^2)), banda 2
PMRA	movimiento propio en ra
sigPMRA	incerteza de PMRA
PMDec	movimiento propio en dec
sigPMDec	incerteza de PMDec

El proceso de generación del preprocesamiento desde archivos .tbl a .csv y su posterior inserción en la base de datos en Cassandra consta de varios pasos. En primer lugar, se utiliza un código de Python que hace uso de multiprocessing para procesar en paralelo múltiples archivos .tbl. Este código lee línea por línea los archivos, descartando el encabezado, y utilizando un archivo generado a partir de la descripción del archivo de CatWISE Catalog Output Table (Rev. 2.2, Mar. 18, 2020)[4] y otros recursos relacionados. Con esta información, se seleccionan las 20 columnas de interés y se genera un dataframe de Pandas que contiene todos los objetos de cada archivo .tbl, junto con la información correspondiente a esas columnas. Luego, se guarda este dataframe en un archivo .csv con el mismo nombre que el archivo .tbl, pero con la extensión cambiada.

A continuación, se utiliza otro código de Python también basado en multiprocessing para corregir en paralelo múltiples archivos .csv. Este código lee cada archivo .csv como un dataframe de Pandas y, tomando como referencia el archivo de CatWISE2020_Table1_20201012.tbl[4], realiza correcciones matemáticas en la información de las columnas ra, dec, pmra y pmdec, según corresponda al tile correspondiente de acuerdo con lo descrito por Marocco et al. [14]. Los resultados se guardan en un nuevo dataframe de Pandas, y se añade una modificación al nombre del archivo para identificar claramente los archivos corregidos.

Finalmente, dependiendo del método de indexación utilizado, se utiliza otro código de Python que trabaja en paralelo por cada archivo utilizando multiprocessing. Este código

[4]https://irsa.ipac.caltech.edu/data/WISE/CatWISE/gator_docs/CatWISE2020_Table1_20201012.tbl

lee cada archivo .csv corregido, extrae los valores de ra y dec, y calcula el píxel o píxeles correspondiente al nivel o niveles de resolución deseados. Estos valores se guardan en nuevas columnas del dataframe (una columna por nivel de resolución), que identifican claramente a qué nivel de resolución corresponden. Estos últimos archivos .csv generados son los que se pueden insertar en la base de datos en Cassandra.

4.2. Base de datos

Teniendo en cuenta la gran cantidad de objetos contenidos en el catálogo, se optó por utilizar Apache Cassandra, un sistema de base de datos distribuida, para manejar eficientemente estos volúmenes de datos sin comprometer la estabilidad del sistema.

Para realizar consultas a la base de datos y obtener información de manera eficiente, se utilizó el Cassandra Python Driver. Esta biblioteca proporciona una interfaz de programación sencilla y flexible para interactuar con Apache Cassandra desde aplicaciones Python.

El Cassandra Python Driver[5] permite establecer una conexión con el clúster de Cassandra y ejecutar consultas de manera eficiente. Ofrece características como el equilibrio automático de carga, la tolerancia a fallos y la replicación de datos, lo que garantiza un rendimiento óptimo y una alta disponibilidad en entornos distribuidos.

Esta librería proporciona métodos para enviar consultas a la base de datos y recuperar los resultados de manera programática. Permite ejecutar consultas de lectura y escritura, así como realizar operaciones avanzadas como agregaciones y consultas preparadas.

En conjunto, el uso de Apache Cassandra como sistema de base de datos distribuida, el formato CSV para el procesamiento y almacenamiento de datos, y el Cassandra Python Driver para realizar consultas y obtener resultados eficientemente, permitieron desarrollar un sistema robusto y escalable para este trabajo.

Finalmente, los datos almacenados en Apache Cassandra se pueden consultar y analizar fácilmente utilizando la combinación del Cassandra Python Driver y otras herramientas de análisis de datos en Python, lo que brinda flexibilidad y potencia para el análisis y la extracción de información relevante de los datos almacenados.

4.2.1. Estructura básica de las tablas

En Apache Cassandra, las tablas se organizan en una estructura distribuida que permite el almacenamiento y acceso eficiente a grandes volúmenes de datos. A continuación, se describen los componentes clave de la estructura de las tablas en Cassandra:

- Partition: Una partición es la unidad básica de almacenamiento y distribución en Cassandra. Representa un conjunto lógico de filas de datos y se identifica mediante una

[5]https://docs.datastax.com/en/developer/python-driver/3.28/api/cassandra/

clave de partición. Cada partición se almacena en un nodo del clúster.

- Partition Key: La clave de partición es un valor único que se utiliza para determinar la ubicación física de una partición en el clúster. Define cómo se distribuyen los datos en los nodos del clúster. La clave de partición puede estar compuesta por uno o más campos de la tabla y se define en la declaración de la tabla.
- Cluster Key: En combinación con la clave de partición, la clave de agrupación se utiliza para ordenar las filas dentro de una partición. La clave de agrupación está formada por uno o más campos de la tabla y se utiliza para ordenar las filas dentro de una partición. Las filas dentro de una partición se almacenan en orden ascendente o descendente basado en los valores de la clave de agrupación.
- Primary Key: La clave primaria de una tabla en Cassandra está compuesta por la clave de partición y la clave de agrupación. Define de manera única cada fila dentro de una tabla y se utiliza para la distribución y ordenación de los datos. La clave primaria puede ser simple, con solo la clave de partición, o compuesta, con la clave de partición y uno o más campos de la clave de agrupación.

La estructura de las tablas en Cassandra es flexible y permite un modelado de datos denormalizado para optimizar las consultas y distribuir los datos de manera eficiente en el clúster. Al diseñar las tablas en Cassandra, es importante considerar la forma en que se accederá y consultará la información para determinar la clave de partición, la clave de agrupación y la estructura de la clave primaria que mejor se adapten a los patrones de acceso y consultas previstos. Esto permitirá aprovechar al máximo las capacidades de escalabilidad y rendimiento de Apache Cassandra[6].

Teniendo esto en cuenta, las tablas utilizadas para el crossmatching deben contener en su estructura las 20 columnas del catálogo descritas en la tabla 4.1, donde 19 de estas se consideran columnas regulares o no clave. Estas columnas almacenan los datos que se desean conservar en la tabla y recuperar a través de las consultas o queries. Las consultas utilizan las claves primarias (clave de partición y la clave de agrupación) para organizar y recuperar estos datos de manera eficiente. Una excepción a esta regla es la columna "source_id", que es única para cada objeto del catálogo y forma parte de la clave primaria, específicamente, de la clave de agrupación en las tablas. El componente restante de la clave primaria corresponde a la clave de partición, que en este caso, siempre corresponderá al índice de un píxel de los objetos.

Para ilustrar esto, se tiene como ejemplo una tabla utilizada para la pixelización mediante HEALPix. Los mapas de HEALPix tienen un parámetro llamado nside, que define la resolución de la pixelización esférica y proporciona un valor cuantificable. Este parámetro es un número entero relacionado con el número total de píxeles utilizados para cubrir toda la esfera. El número total de píxeles en un mapa HEALPix es igual a $12 \times nside^2$. Un nside mayor resulta en una resolución más alta, con una mayor cantidad de píxeles más pequeños cubriendo la esfera. Por ejemplo, un nside de 1 produciría una esfera cubierta con 12 píxeles, mientras que un nside de 2 produciría 48 píxeles, y así sucesivamente [9].

[6]https://www.datastax.com/blog/advanced-data-modeling-on-apache-cassandra

Así, nside controla la escala de los píxeles y, por ende, el nivel de detalle que se puede representar en el mapa del cielo.

Si, por ejemplo, se trabaja con una tabla para nside = 10, se tienen 19 columnas regulares y una clave de partición nside10, que se usa para determinar en qué nodo de datos se almacena una fila. Las filas con la misma clave de partición se almacenan en el mismo nodo. En este caso, todas las filas que tienen el mismo valor para nside10 se almacenan juntas. Aunque nside10 no necesita ser único en la tabla, su propósito es dividir los datos en grupos manejables que se pueden distribuir a través del clúster de Cassandra.

La clave de clustering es source_id y se utiliza para ordenar y organizar los datos dentro de una partición. Es decir, las filas con la misma clave de partición (nside10 en este caso) se ordenarán según source_id. Aunque source_id no necesita ser único en toda la tabla, sí debe ser único dentro de cada partición nside10, lo cual siempre se cumplirá, ya que en CatWISE2020, source_id es un valor único por objeto.

Un desafío con este tipo de estructura de tabla es que requerirá tener una tabla por cada nivel de nside a utilizar, lo cual implica un mayor uso de espacio de disco que si se pudiera almacenar todos los índices nsides dentro de una misma tabla. Esto se debe a que CatWISE2020 tiene casi 2 mil millones de objetos catalogados, donde cada uno contará con 20 columnas de datos. Se podrían considerar dos opciones para resolver este problema, pero una de ellas se descarta inmediatamente ya que, aunque resolvería el problema del consumo de memoria, anularía toda la velocidad que se obtiene al usar Cassandra como base de datos.

La opción descartada involucraría generar dos tablas. Una de ellas tendría como columnas regulares las 19 columnas que se mencionaron anteriormente, y una clave primaria formada por un entero único para cada objeto del catálogo como clave de partición y source_id como clave de agrupación.

La segunda tabla sería un poco más compleja, con una columna no clave que contiene un entero, el mismo identificador único para cada objeto presente en la primera tabla. Para la clave, se tendría que definir un rango de nsides a utilizar, y se tendría una estructura pseudo-jerárquica. Por ejemplo, si el rango se define entre nside = 10 y nside = 18, se tendría una clave de partición dada por nside 10, y una clave de agrupación compuesta por nside desde 11 hasta 18, y source_id para garantizar la unicidad de cada objeto.

La idea detrás de estas tablas es lograr una separación de los datos que serían comunes a las diferentes tablas que existirían bajo el primer esquema descrito, y extraerlos de acuerdo con cada objeto buscándolos con su valor de nside. Las funciones de mapeo de HEALPix permiten conocer los valores de los píxeles que se encuentran dentro de un cono para un nivel de nside definido, por lo que la segunda tabla permitiría conocer el identificador único de cada objeto, para luego, con la primera, extraer sus datos.

Existen dos problemas con estos enfoques. El primero, que aunque añade complejidad, no hace inviable la estructura de la segunda tabla, es que de acuerdo con los requisitos de Cassandra, para consultar los nside se tendría que consultar todos los nsides hasta llegar al nside de interés, dado que se tendría que tener los distintos nside como claves de agrupación para aprovechar las ventajas de utilizarlos como claves. Por ejemplo, si se quiere consultar por objetos usando nside 13, será necesario consultar por los índices de píxeles que describan

el área del cono en nside 10, en nside 11, en nside 12 y finalmente en nside 13, y así sucesivamente, lo que ralentizará las consultas, ya que sería necesario calcular los índices para más niveles de nsides.

El segundo problema corresponde a la necesidad de realizar dos consultas a las bases de datos para emular las funcionalidades de una consulta con "*JOIN*", dado que esta cláusula no está implementada dentro del lenguaje de queries CQL de Cassandra. Realizar dos consultas a la base de datos aumentaría de manera considerable la cantidad de tiempo necesario para procesar las consultas de crossmatch, en especial en los casos que exista un alto número de objetos resultantes.

Aunque es posible eludir la necesidad de realizar una consulta jerárquica a nivel de nside a través de generar una tabla donde se indexe únicamente un nivel de nside, persiste el problema fundamental del uso de espacio de disco. Por lo tanto, no será totalmente factible minimizar el uso de espacio y el tiempo requerido para las consultas.

Una estrategia viable podría ser la reducción del número de nsides en uso. Es evidente que una tabla para nside = 1 no aportaría utilidad significativa, dado que el enfoque de ALeRCE se concentra en búsquedas con un radio de 1 arcosegundo principalmente, y ocasionalmente en consultas de un radio de un arcominuto. Un nside 1 implicaría la presencia de 12 píxeles, cada uno cubriendo un cono sustancialmente mayor que el requerido, lo cual no resultaría beneficioso. En consecuencia, sería factible limitar los valores de nside a un rango que cubra los radios especificados. Sin embargo, esto suscita la interrogante de cuáles serían los valores de nside óptimos para consultas en estos radios.

Este último punto motiva una subsección posterior en este mismo capítulo, en la que se llevaron a cabo experimentos para abordar precisamente esta cuestión. Además, se realizó una comparación de los tiempos de consulta al emplear una tabla por nside, para determinar los tiempos de procesar por completo consultas de crossmatch con distintos radios, analizar cómo evolucionan los tiempos al aumentar el radio y responder la interrogante de si es necesario realizar una estructura jerárquica para optimizar tanto tiempos como utilización de disco. Aunque podría parecer excesiva la preocupación por el espacio, se debe tener en cuenta que este es un solo catálogo, y existen cientos de catálogos astronómicos. Según la evolución de las necesidades de ALeRCE, puede ser necesario indexar más catálogos en el futuro, lo que incrementará aún más el consumo de memoria física, a un nivel que podría volverse insostenible.

4.2.2. Inserción de datos

Para la inserción de datos, como se mencionó anteriormente, se utiliza dsbulk, una herramienta integral de carga masiva de datos para Apache Cassandra, alojada dentro de un contenedor de Linux en el servidor de ALeRCE operado por Docker. La principal funcionalidad de dsbulk es facilitar la inserción eficiente de altos volúmenes de datos, provenientes de archivos CSV u otros formatos compatibles, en las tablas de Cassandra. Esta herramienta ha sido diseñada para operaciones de carga y descarga de datos de manera rápida y eficiente en un clúster de Cassandra, permitiendo inserciones, actualizaciones y eliminaciones masivas de datos en paralelo, aprovechando al máximo los recursos del clúster.

La introducción de datos mediante dsbulk se desarrolla en múltiples fases. En el primer paso, se verifica que el archivo CSV esté adecuadamente formateado y estructurado, siguiendo las directrices del esquema de la tabla de Cassandra donde se proyecta introducir la información. Seguidamente, se genera un archivo de configuración que detalla los aspectos de la conexión al clúster de Cassandra, la localización y formato del archivo CSV (que, en este contexto, se almacenan junto con la instalación de dsbulk en el sistema de archivos de Linux operado por Docker en el servidor), así como cualquier otra opción de configuración pertinente. Tras la adecuada preparación de la configuración, se inicia el procedimiento de carga, ejecutando el comando dsbulk en la línea de comandos junto con el archivo de configuración.[7]

Durante el procedimiento de carga, dsbulk proporciona información en tiempo real sobre el progreso de la operación, como el número de filas procesadas y el tiempo transcurrido. Esta función permite el monitoreo y seguimiento del proceso para garantizar su correcta finalización.

Con la utilización de dsbulk, se aprovecha su capacidad de procesamiento masivo y paralelo para cargar grandes cantidades de datos de manera eficiente en las tablas de Cassandra. Este recurso representa una herramienta poderosa y escalable para la gestión de datos en clústeres de Cassandra.

4.3. Experimentos preliminares y resultados

El objetivo de estos experimentos preliminares es identificar el nivel de tamaño óptimo de los píxeles para optimizar la rapidez del proceso de crossmatch. Se anticipa que los óptimos estarán vinculados al radio utilizado en las consultas, por lo que puede ser necesario indexar múltiples niveles de profundidad para cada metodología aplicada. Esto podría llevar a un consumo significativo de memoria si se utilizan múltiples catálogos, sugiriendo la necesidad de examinar el uso de tablas con niveles de resolución jerárquicos.

Se resalta la necesidad de comparar diferentes niveles de resolución o tamaño de píxeles (referidos en adelante como nside para HEALPix y htm_depth para Hierarchical Triangular Mesh). Esto se debe a que, durante las consultas a la base de datos, se manejarán los píxeles que describen el espacio completo entre una coordenada de entrada y el radio especificado.

Si el tamaño del píxel es grande, se espera que la consulta en Cassandra sea rápida. Sin embargo, se obtendrán muchos objetos cuya distancia real a la coordenada de entrada sea considerablemente mayor que el radio especificado, lo que requerirá un tiempo sustancial para filtrar y retener sólo aquellos que cumplen con esta condición. Por otro lado, si el tamaño del píxel es pequeño, será necesario consultar un número mayor de píxeles para cubrir completamente el área, pero el filtrado debería ser rápido, ya que la mayoría, si no todos los objetos, cumplirán con la condición de estar a una distancia menor que el radio de consulta. No obstante, el aumento en la cantidad de píxeles podría provocar un incremento considerable en el tiempo necesario para ejecutar las consultas.

Es relevante señalar la necesidad de realizar estas pruebas utilizando una muestra de

[7] https://docs.datastax.com/en/dsbulk/docs/reference/load.html

los datos, pues los tiempos de ejecución de las consultas en Cassandra variarán según la cantidad de índices únicos y objetos que comparten estos índices en cada tabla. Aunque estas cantidades cambiarán al usar el conjunto completo de los datos, una muestra del 25 % será representativa del comportamiento general y de los tiempos requeridos para el filtrado de resultados. Esta última cantidad no debería variar significativamente ya que se espera que la mayoría de los objetos resultantes de un crossmatch con coordenadas de entrada entre 0° y 90°, también se encuentren dentro de este conjunto indexado.

Así,para estos experimentos se propone usar aproximadamente un 25 % de los datos de CatWISE2020, indexando todas las tablas con un RA entre 0° y 90°. Aunque la indexación de estos datos no representará exactamente un cuarto del catálogo, dado que los objetos en la galaxia se distribuyen de forma no uniforme, con una mayor densidad en el plano galáctico, para los propósitos de estas pruebas será útil trabajar con una fracción de los datos. Se escoge este porcentaje dado que corresponde a un número considerable de objetos del catálogo, lo que permite realizar pruebas similares a la realidad con todos los datos, considerando que la mayoría de los objetos matcheables dentro de un crossmatch se encuentran en coordenadas similares, y que además es posible procesar e indexar todas las filas en un tiempo razonable.

Para las dos metodologías de pixelización, HEALPix y HTM, se genera una tabla en Cassandra donde se indexa un índice dentro de un rango de índices de resolución, con las que se ejecutan consultas de crossmatching utilizando las mismas funciones que se utilizarán en la versión final de la API, midiéndose los tiempos desde el envío de la consulta hasta la escritura de un archivo de salida. Este proceso se repite 10 veces para cada medición, con un archivo de entrada compuesto por 20 mil coordenadas aleatorias con RA generadas entre 0° y 90°, similares a los datos indexados.

El rango de índices a probar es entre 10 y 22, tanto para nside (HEALPix) como para htm_depth. Se elige este rango para trabajar con un amplio espectro de resoluciones, asumiendo que diferentes niveles podrían funcionar mejor con diferentes radios. Además, es necesario incorporar en los análisis cómo se comportan los tiempos de filtrado requeridos. Según HEALPix, un nside de 16 correspondería a una resolución de aproximadamente 3 arcosegundos, lo que sugiere que el óptimo para las consultas estaría cerca de este orden, por lo que se espera que el rango seleccionado deba incluir el óptimo para los radios a utilizar, que son de 1, 2, 4, 8, 16, 60 arcosegundos.

En resumen, se generan para cada tabla con RA entre 0° y 90°, un total de 13 archivos CSV, en los que a todos los objetos se les calculan sus índices de HEALPix y HTM para los niveles de resolución entre 10 y 22, y se agrega una columna correspondiente a esto en el CSV. Luego se generan las tablas en Cassandra correspondientes a estos niveles, y se usará dsbulk para cargar los CSV a la base de datos. Después, se realizan consultas con una versión de la API que incorpora las funciones básicas de la versión final, es decir, las funciones para generar los píxeles a consultar, ejecutar consultas y filtrarlas, devolviendo resultados en formato CSV para un conjunto de muestra de 20 mil puntos con RA entre 0° y 90°, para los diferentes niveles de resolución, para ambos métodos.

Los resultados de estos experimentos se presentan a continuación.

4.3.1. HEALPix

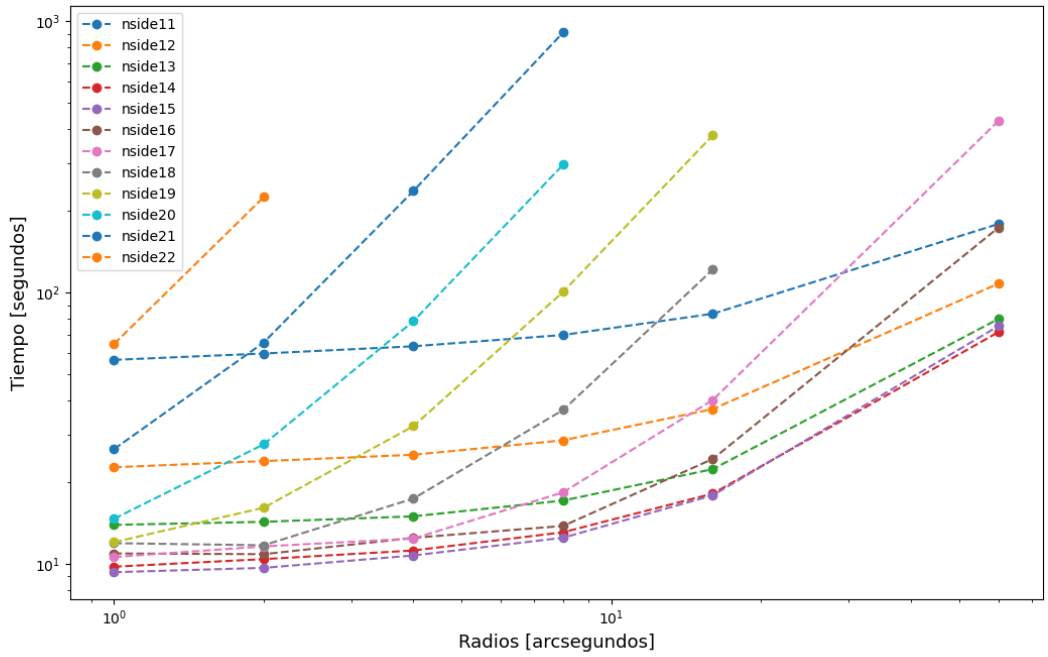

Figura 4.1: Plot de tiempos crossmatch HEALPix distintos nside, 20 mil puntos, con un sample de 25 % de los datos y un radio de 1 arcosegundo fijo.

En la figura 4.1 se puede observar que no es posible ejecutar consultas para todos los niveles de nside en cada uno de los radios considerados, debido a que la complejidad de las consultas resulta en fallas de la base de datos. Por ejemplo, en el caso de nside18, no es factible realizar consultas con un radio de 60 arcosegundos. Este problema surge porque al incrementar el valor de nside, se reduce el tamaño de los píxeles. En combinación con un radio extenso, esto resulta en un número elevado de píxeles a consultar. Dado que se emplean consultas concurrentes, se acaban realizando numerosas consultas a píxeles de manera paralela. Para realizar consultas concurrentes Cassandra no utiliza múltiples hilos, sino que maneja múltiples instancias de conexiones dentro de un solo proceso para mejorar el rendimiento de las operaciones de base de datos concurrentes

Este proceso requiere una gran cantidad de operaciones de lectura de disco y un amplio uso de memoria, lo que ocasiona que algunas consultas no puedan completarse. Este fenómeno es evidente en los gráficos subsecuentes, donde más allá de ciertos valores de radio y nside, las mediciones simplemente dejan de existir. Esta misma problemática se presenta al utilizar el Hierarchical Triangular Mesh (HTM), como se discutirá más adelante.

La figura 4.1 muestra que los tiempos de consulta no varían de manera significativa para diferentes radios. En cambio, el nside óptimo para los distintos radios se mantiene entre 14 y 15 para todas las consultas. Para radios más pequeños, nside 15 es ligeramente más rápido

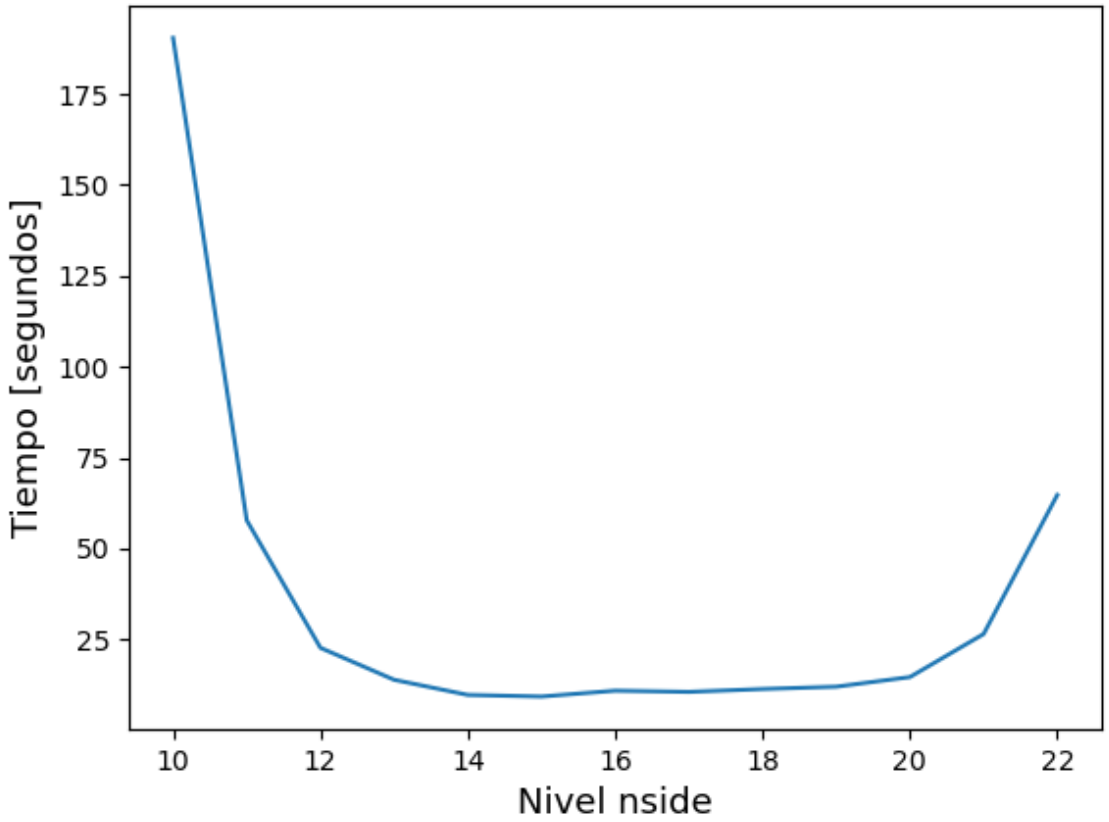

Figura 4.2: Plot de tiempos crossmatch HEALPix distintos nside, 20 mil puntos, con un sample de 25 % de los datos y un radio de 1 arcosegundo fijo

que nside 14. Sin embargo, para un radio de 60 arcosegundos, nside 14 resulta ser la mejor opción.

Aunque el gráfico no permite discernir directamente la relación entre nside y el tiempo de procesamiento del crossmatching, es posible elaborar otro gráfico que compare los tiempos para los distintos nside con un radio fijo, el cuál corresponde a la figura 4.2. Dado que no todos los niveles de nside tienen mediciones para todos los radios empleados, por la razón antes citada, se utilizará un radio de 1 arcosegundo para observar la tendencia de los tiempos a lo largo de todo el rango de nsides empleado. En este gráfico es posible observar que la hipótesis de que aumentaría el tiempo necesario para crossmatch en los nsides extremos era correcta, encontrándose un óptimo en los nsides más centrales, los cuáles para el radio de 1 arcoseugndo no presentan diferencias significativas en promedio.

4.3.2. Hierarchical Triangular Mesh

Los resultados obtenidos al utilizar HTM, siguiendo la misma metodología, son análogos a los logrados con HEALPix, lo que se puede observar en la figura 4.3. Nuevamente, no es posible

completar las consultas para todos los radios, dado que los niveles altos de htm_depth y radios extensos causan que la base de datos deje de funcionar. A diferencia del caso de HEALPix, los tiempos mínimos para todos los radios se alcanzan para htm_depth de profundidad 15 y 16. Además se tiene que en este caso la diferencia de tiempos para radios de 1 arcominuto no es tan grande como con HEALPix.

Al analizar únicamente utilizando un radio fijo de 1 arcosegundo, dado que este corresponde al radio más utilizado por ALeRCE, lo cual corresponde a la figura 4.4, los resultados se asemejan a los previos, donde existe un único nivel de htm_depth (21) que rompe con la tendencia general, sin embargo al igual que en el caso anterior, se tiene que para resoluciones máximas y mínimas se tienen los tiempos más altos, mientras que en los centrales las diferencias de tiempos no son tan altas.

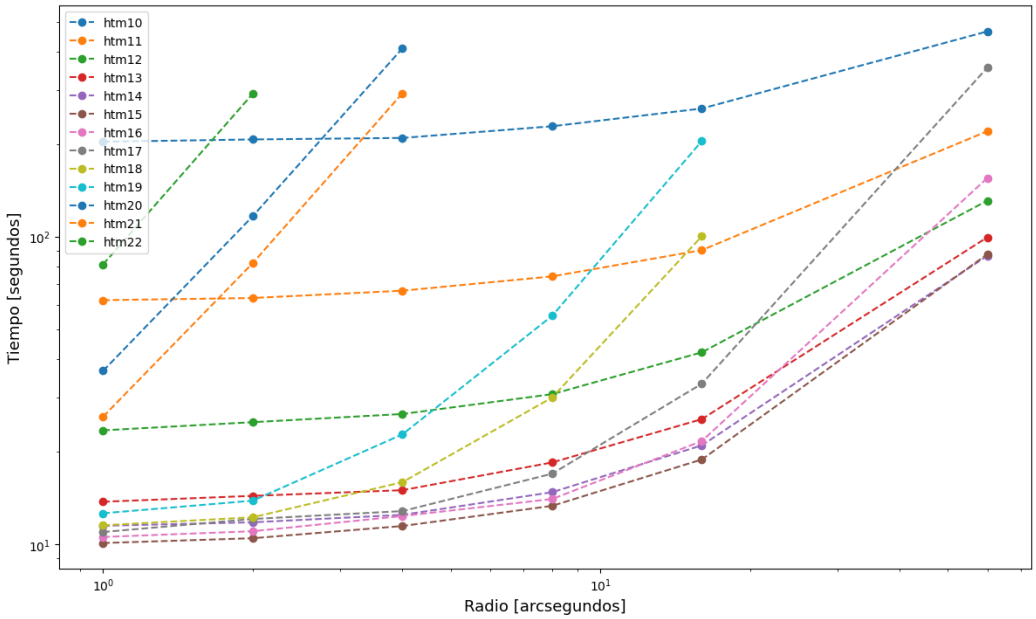

Figura 4.3: Plot de tiempos crossmatch HTM distintos htm_depth, 20 mil puntos, con un sample de 25 % de los datos, ejes en escala logarítmica.

4.4. Estructura final de la base de datos

Basándose en los resultados obtenidos, se concluye que no resulta necesario implementar una estructura de índices jerárquicos, dado que los tiempos de consulta y sus óptimos no variaron de la manera prevista inicialmente con respecto al nivel de nside o htm_depth. De mayor importancia aún, la eficiencia de los tiempos para ejecutar las consultas de crossmatch supera el mínimo requerido en el caso de un radio de un arcosegundo. Se teoriza, además, que haciendo uso de threads se podría alcanzar el mínimo de 350 consultas por segundo para el caso de 1 arcominuto, lo cual será estudiado en detalle en el capítulo 6.

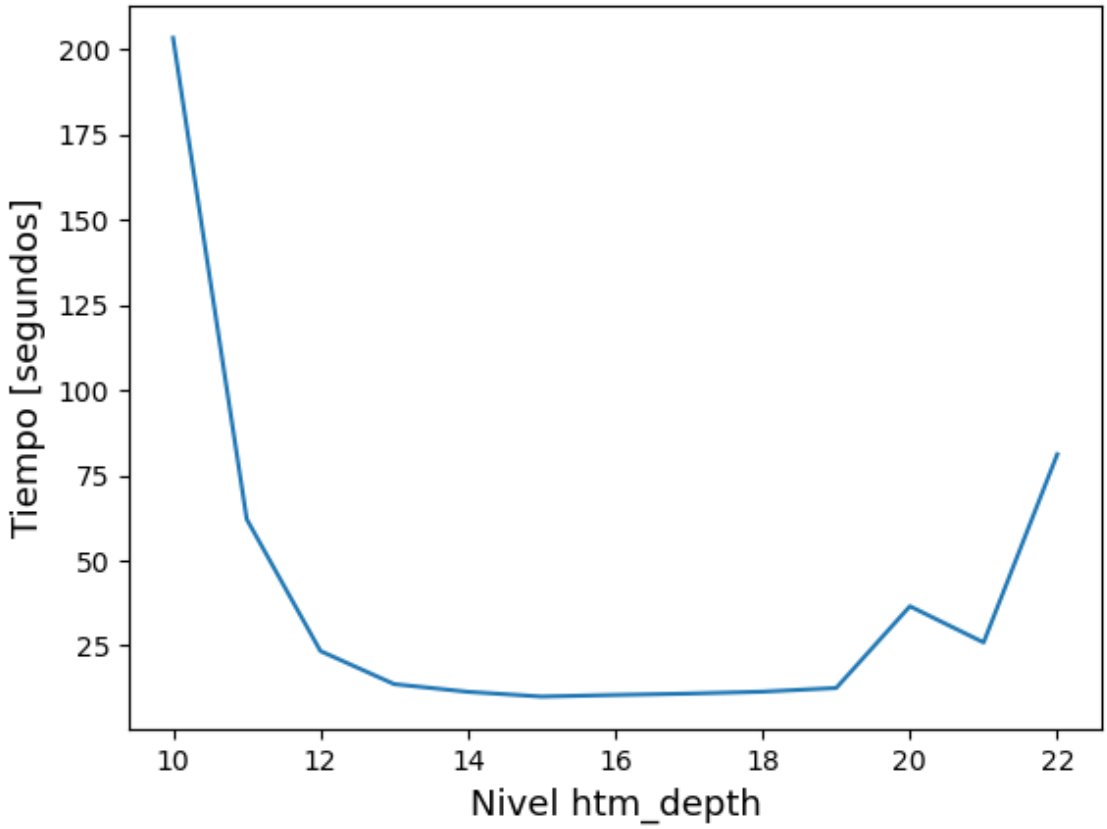

Figura 4.4: Plot de tiempos crossmatch HTM distintos nside, 20 mil puntos, con un sample de 25 % de los datos y un radio de 1 arcosegundo fijo.

En función de los gráficos, se optó por emplear un único índice tanto para HEALPix como para HTM. En el caso de HEALPix, se decidió finalmente por nside 14, debido a la notable diferencia de tiempo en las consultas de 1 arcominuto, aunque para el resto de los radios, nside 15 sería levemente una mejor opción. Por otro lado, para HTM se utilizará el índice htm_depth 15, que, aunque presenta un desempeño ligeramente inferior para el radio de 1 arcominuto en comparación con htm_depth 14, muestra que la diferencia de tiempo es menor que en el caso de HEALPix.

Por ende, se establece la siguiente estructura en la base de datos. Se configura un keyspace denominado crossmatch_cassandra, que alberga dos tablas. La primera es la tabla catwise2020_nside14, destinada a las consultas usando HEALPix. Esta tabla comprende las 20 columnas detalladas en la tabla 4.1, además de una columna adicional denominada nside 14, que representa el índice de HEALPix para el nivel de resolución nside = 14, calculado para cada objeto a partir de su respectiva coordenada Ra y Dec. Esta tabla tiene como clave de agrupación la columna source_id, que es un valor único para cada objeto, y como clave de partición correspondiente a nside 14, ambas constituyen la clave primaria.

Para HTM, se tiene la tabla catwise2020_htm15 en el mismo keyspace crossmatch_cassandra, donde la única diferencia radica en que, en lugar de contar con el valor de nside 14 como

columna, se tiene la columna htm 15, desempeñando una función equivalente.

Capítulo 5

Implementación del servicio

En el siguiente apartado, se proporciona una descripción detallada de la configuración de la interfaz que se integrará con el pipeline de ALeRCE, lo cual permitirá la ejecución de consultas de crossmatch de manera eficiente y precisa.

Durante la implementación de esta interfaz, se consideraron cuidadosamente las funciones previamente desarrolladas y probadas para el crossmatch, las cuales se exploran en profundidad en el capítulo 6.

Es importante resaltar que todo el código ha sido desarrollado utilizando Python como lenguaje de programación principal. Esta elección proporciona una combinación perfecta de flexibilidad y facilidad de uso, especialmente en el contexto de esta integración. Este capítulo se divide en dos secciones principales para abordar de manera exhaustiva la implementación del pipeline de crossmatching. La primera sección se enfoca en las funciones esenciales necesarias para generar el pipeline, mientras que la segunda sección se centra en la implementación de estas funciones dentro de una API desarrollada en FastAPI.

5.1. Crossmatching

En esta sección del capítulo, se centra en la descripción del pipeline utilizado para generar resultados en las consultas de conesearch, donde se destacan los aspectos más relevantes del proceso. Se proporcionará una visión detallada de las etapas clave del pipeline, abarcando desde la preparación de los datos hasta la generación de los resultados finales. Se explorarán los componentes esenciales, los métodos utilizados y las consideraciones importantes para tener en cuenta durante el proceso. Esta descripción permitirá una comprensión clara y completa del funcionamiento del pipeline y su relevancia en el contexto de las consultas de conesearch.

5.1.1. Preparar las consultas

Dada la simplicidad de la estructura de las tablas en Cassandra, el primer paso consiste en generar consultas en el lenguaje CQL (Cassandra Query Language) que se puedan ejecutar en la base de datos. Para ello, es necesario conocer la tabla a consultar y el listado de píxeles de interés, con el fin de generar consultas similares a *'SELECT * FROM keyspace.table WHERE indice IN ('pixel1', 'pixel2')'*.[1]

El pipeline procesa un archivo de entrada que contiene una lista de coordenadas astronómicas, representadas por Ra (Ascensión recta) y Dec (Declinación), en el sistema de referencia ICRS y con unidades en grados. Este archivo, como se describe en la siguiente sección, puede tener formato .txt o .csv, aunque se profundizará en este aspecto más adelante. Además, se proporcionan entradas para especificar el radio de la consulta, tanto su magnitud como la unidad de medida, que puede ser arcosegundos o arcominutos.

Originalmente, el pipeline comenzaba transformando las coordenadas Ra, Dec en coordenadas de astropy.SkyCoords[2], lo que facilita el manejo de las coordenadas. Luego, se utilizaba astropy.HEALPix[3] para obtener los conjuntos de píxeles que cubren completamente el cono definido por cada coordenada y el radio de búsqueda. Sin embargo, se identificaron inconsistencias durante pruebas preliminares, como se describe en la sección 4.1. Ahora, el otro conjunto de inconsistencias que se observaron, está relacionado con la falta de objetos encontrados por el servicio HEALPix en comparación con CDS. Estos objetos se distribuían en áreas no cercanas a los límites de las consultas.

Después de realizar pruebas exclusivamente utilizando las coordenadas de los objetos que faltaban, se descubrió una inconsistencia. Aunque los valores de los índices de píxeles para un nivel específico de resolución, calculados para sus coordenadas Ra, Dec eran correctos, en algunos casos, como en el ejemplo del píxel "pix1" , al consultar los píxeles que describen un cono para una coordenada a una distancia "rçon un radio ligeramente mayor, el píxel "pix1" no se encontraba en el listado generado. Esto es claramente incorrecto y contradice la expectativa de que todos los píxeles dentro del cono deberían estar incluidos.

Se resolvió rápidamente este problema al cambiar la biblioteca de Python utilizada para calcular el listado de píxeles de HEALPix de astropy.HEALPix a healpy. Este cambio requirió agregar un par de pasos adicionales, ya que healpy trabaja con vectores y radianes en lugar de utilizar directamente astropy skycoords y astropy.units, que es lo que se utilizaba para astropy.HEALPix. Sin embargo, esto no generó un aumento significativo en los tiempos de procesamiento, y se logró que los resultados obtenidos fueran consistentes entre el sistema desarrollado y los de CDS.

Una vez considerados estos aspectos, se vuelve sencillo generar la consulta CQL que se utilizará para el conesearch. Basta con utilizar el nivel de resolución para cada método, en conjunto con el radio de búsqueda y las coordenadas correspondientes para obtener el conjunto completo de píxeles que abarcan el área descrita por el cono. Luego, simplemente transformando esta lista de píxeles al formato de texto requerido por Cassandra, se puede

[1] https://docs.datastax.com/en/cql-oss/3.1/cql/cql$_u$*sing/useQueryIN.html*

[2] https://docs.astropy.org/en/stable/api/astropy.coordinates.SkyCoord.html

[3] https://astropy-healpix.readthedocs.io/en/latest/

combinar toda la información para generar cada consulta a ejecutar. De esta manera, se logra generar las consultas de manera efectiva.

5.1.2. Ejecutar las consultas

Para ejecutar las consultas en Cassandra, se utiliza el controlador de Python para Cassandra (Cassandra Python Driver). Esta biblioteca permite establecer una conexión con el clúster y crear una sesión que se utilizará para ejecutar las consultas. En este momento, es importante destacar que la próxima subsección abordará el filtrado, el cual se explicará detalladamente más adelante. Sin embargo, lo relevante en este momento es que el filtrado requiere tanto una lista con toda la información de los objetos resultantes como una lista de astropy.SkyCoords para esos objetos.

Para facilitar el proceso de obtención de toda la información de los objetos resultantes de las consultas, se puede configurar el formato de respuesta de las consultas para que los resultados de las consultas se devuelvan como un diccionario, y así se puede obtener una lista de diccionarios. Luego, mientras se van guardando los diccionarios, es posible generar simultáneamente una lista de coordenadas RA (Right Ascension) y otra lista de coordenadas DEC (Declination). A partir de estas dos listas, se puede crear un arreglo de astropy.SkyCoords, lo cual resulta considerablemente más rápido que generar cada objeto astropy.SkyCoords individualmente.

Dentro de las optimizaciones realizadas para poder realizar las consultas de la manera más ágil posible, se encuentra el uso de *prepared statements*[4] lo que permitió reducir los tiempos de ejecuciones debido a que es posible saltarse la fase de parseo de la consulta CQL, y utilizar la función execute_concurrent[5], que permite entregar el listado de parámetros a aplicar a los prepared statements y ejecutar de manera concurrente un alto número de consultas. En este caso, se utilizó el parámetro que controla el nivel de concurrencia como concurrent = 350.

5.1.3. Filtrar los resultados

El siguiente paso en el proceso del pipeline consiste en filtrar los objetos resultantes de las consultas para asegurar que la distancia entre ellos y la coordenada de entrada coincidente sea menor al radio especificado para la búsqueda de conesearch. Este filtrado es necesario debido a que, aunque el uso de los píxeles que describen completamente el cono garantiza la inclusión de todos los objetos dentro de la distancia indicada, existe la posibilidad de que un píxel sea necesario para describir el área en su totalidad, pero solo contenga una pequeña fracción de dicho píxel. Esto significa que, si un objeto se encuentra en el extremo más alejado de ese píxel parcialmente contenido en relación a la coordenada utilizada en la búsqueda de conesearch, su distancia al objeto resultante podría ser considerablemente mayor que el radio de búsqueda utilizado.

Además, es necesario realizar un filtrado adicional en ciertos casos. Por ejemplo, puede

[4]https://docs.datastax.com/en/developer/java-driver/3.0/manual/statements/prepared/
[5]https://docs.datastax.com/en/developer/python-driver/3.28/api/cassandra/concurrent/

ser de interés realizar el crossmatch solo con el objeto más cercano a la coordenada del archivo de entrada, o limitar el número máximo de objetos resultantes que se matchean con la coordenada. Por esta razón, se incluirá un parámetro de entrada llamado "limit" en la API, el cual será un número entero mayor a 0 y opcional. Si se proporciona un valor distinto de None, indicará la cantidad máxima de objetos resultantes con los que se realizará el match para cada coordenada del texto de entrada.

Para realizar el matcheo entre las coordenadas del conjunto resultante de las consultas al catálogo y las coordenadas del archivo de entrada correspondientes al conesearch, se emplean dos funciones del paquete astropy.coordinates. La elección de la función depende de si se está utilizando el parámetro de entrada "limit" con un valor igual a 1. En caso afirmativo, se utiliza una función específica, mientras que, en caso contrario, se utiliza otra función.

En el escenario en el que se limita el resultado a una coincidencia, se utiliza la función match_coordinates_sky. Esta función toma dos listas de coordenadas SkyCoords: una que representa un catálogo y otra que contiene las coordenadas de referencia que se desean relacionar con el catálogo.[6]

El resultado de la función es un arreglo de índices correspondientes a las coordenadas del catálogo que son más cercanas a cada coordenada de referencia. También devuelve dos arreglos que contienen las distancias en 2D y 3D entre las parejas de coincidencias.

En el caso del proceso de crossmatching que se está llevando a cabo, las coordenadas del catálogo son el conjunto de coordenadas obtenidas al ejecutar las consultas en la base de datos, mientras que las coordenadas de referencia son aquellas del archivo de entrada para el crossmatching.

Es posible imponer una distancia máxima utilizando una máscara booleana con el arreglo de distancias en 2D. Esto genera dos arreglos de índices: uno para las coordenadas en el catálogo y otro para las coordenadas de referencia. Cada par de índices indica que la primera coordenada del catálogo se encuentra relacionada con la primera coordenada de referencia, y así sucesivamente. Con estos índices es posible extraer el diccionario completo de la información del objeto matcheado resultante, además de agregarle información extra correspondiente al Ra/Dec de las coordenadas de referencias y la distancia real que los separa, para permitir una mejor identificación al leer los resultados finales.

Por otra parte, en el caso donde se utiliza un valor entero para limit, se utiliza la función search_around_sky. Esta función es similar a match_coordinates_sky, pero difiere en que busca coincidencias cercanas dentro de un radio específico alrededor de cada coordenada de referencia en un catálogo de coordenadas. Devuelve los índices de los objetos del catálogo que se encuentran dentro del radio especificado para cada coordenada de referencia, junto con las distancias correspondientes. Esto significa que puede haber múltiples coincidencias o incluso ninguna entre los conjuntos de coordenadas.

Para aplicar el límite variable (limit), se consideran todos los objetos que corresponden a una única coordenada de entrada. Esto es posible gracias a que la función devuelve dos arreglos: uno con todos los pares matcheados y sus índices en el conjunto de catálogo y el

[6]https://docs.astropy.org/en/stable/coordinates/matchsep.html

conjunto de referencia, respectivamente. Para cada coordenada de referencia que tiene al menos un match, habrá “n” objetos matcheados en el catálogo, siendo “n”mayor que 1.

Extrayendo las distancias de estos objetos matcheados, es posible ordenarlas de menor a mayor y seleccionar las primeras distancias correspondientes al menor valor entre n y limit. Conociendo las distancias y sus índices en el arreglo de diccionarios, es posible generar un nuevo arreglo donde se incluyen únicamente los objetos matcheados filtrados. En este nuevo arreglo, se añade información sobre la coordenada de referencia (RA/DEC) con la que hicieron match y su distancia a esta coordenada.

5.1.4. Retornar los objetos resultantes

Es fácil obtener la información de los objetos resultantes a partir del listado de diccionarios filtrados, accediendo a los distintos campos a partir de los nombres asignados en las tablas de Cassandra. Dado que hay múltiples puntos finales (endpoints) en la API según el formato de entrada y salida para el crossmatch, hay dos opciones para transformar el arreglo de diccionarios, pero no hay dificultades significativas.

La primera opción es convertir la salida a un archivo JSON, lo cual se puede hacer utilizando la biblioteca json de Python.

Por otro lado, también es posible convertir el resultado a un archivo CSV. Esto se logra recorriendo la información de todos los diccionarios y utilizando la biblioteca pandas para generar un dataframe y luego retornarlo.

5.2. API

Para desarrollar la API se empleó la biblioteca FastAPI[7], donde se habilitaron distintos endpoints, los cuales se describen a continuación:

Lista de Catálogos (método GET)

Muestra un listado de los nombres de los catálogos disponibles en el sistema. Esta información puede ser obtenida mediante la ruta “/catalog”. No se requiere ningún parámetro para acceder a este endpoint. Estos nombres de catálogos son utilizados para los endpoints de crossmatch.

Un ejemplo de respuesta se presenta a continuación:

```
[
" catwise2020_nside14 ",
```

[7] https://fastapi.tiangolo.com/

```
" catwise2020_htm15
]
```

Crossmatch con output JSON (método POST)

El endpoint ofrece una lista de los objetos resultantes del proceso de crossmatching entre un catálogo específico y una lista de posiciones. Se puede acceder a él a través de la URL "/crossmatch_json". El archivo de entrada es un .txt y la salida es un archivo .json, los cuales se explicarán en detalle más adelante. También hay otro endpoint similar que tiene un formato diferente para los archivos de entrada y salida. Los parámetros necesarios para este endpoint son los siguientes:

- *radius* (float): El valor de la distancia máxima en segundos de arco que se utilizará para realizar la búsqueda de cono. Si bien es posible utilizar un amplio rango de radios, la aplicación fue probada con el objetivo de utilizar radios entre 1 arco segundo y 1 arco minuto (o su equivalente, 60 arco segundos)
- *unit* (Unit): La unidad de medida para el radio a utilizar en esta funcionalidad puede ser "arcsec" o "arcmin". Se creó una clase denominada "Unit" para limitar los valores de unidad a estas dos opciones. Aunque es posible utilizar otras unidades como "degree", no se realizaron pruebas de eficiencia para estas unidades adicionales, por lo que se decidió restringir el uso a las unidades que han sido probadas.
- *input_file* (archivo .txt): El conjunto de ubicaciones para el proceso de coincidencia se proporciona en un archivo de texto. Esta estrategia se emplea para optimizar la eficiencia del servicio, ya que se evita enviar una posición por cada solicitud HTTP. El formato de este archivo es compatible con las consultas de coincidencia en la API web de CDS. A continuación, se muestra un ejemplo del archivo a cargar:

  ```
  ra , dec
  53.3667744 , -85.5547224
  262.40749148 , -22.20264135
  254.32098384 , 37.03822843
  97.23693453 , -25.08230158
  157.35843221 , -71.17841342
  ```

- *catalog* (Catalog): Este es el catálogo astronómico utilizado en el sistema para realizar el crossmatch. Es una clase específica que restringe los catálogos a los que se han indexado y también permite a la API generar los píxeles necesarios para la técnica de pixelización requerida y realizar consultas a la tabla correspondiente.
- *limit* (int, opcional): La cantidad máxima de resultados permitidos por cada búsqueda de conesearch realizada en cada punto se define mediante el parámetro "limit". Al establecer un valor para "limit", se seleccionan los "limit" objetos matcheados más cercanos a cada punto del archivo de entrada. Por defecto, si no se especifica un valor para "limit", este se establece como None.

  Aquí se presenta un ejemplo de una solicitud utilizando la biblioteca requests[8]:

[8]https://requests.readthedocs.io/en/latest/

```
catalog = Catalog.CatWISE2020_nside14
unit = Unit.arcsec.value
request_params = {'radius': 1.0, 'unit': unit,
                  'catalog' : catalog}
input_file = {'input_file': open ("coords_test.txt", 'rb') }
results = requests.post ("http://127.0.0.1:8000/crossmatch_json/",
          params =request_params , files = input_file)
```

La respuesta obtenida al realizar una petición a este endpoint se corresponde a un archivo con formato .JSON, el cual está conformado por los campos descritos a continuación:

- *Tiempo de pre-procesamiento* (float): El tiempo de preparación del archivo de entrada para ejecutarlo como una consulta en Cassandra se mide en segundos. Este proceso implica leer el archivo, dividirlo en coordenadas y generar, para cada coordenada de entrada, el conjunto completo de píxeles que describen el área entre la coordenada de entrada y el radio especificado. Además, se realiza la transformación necesaria para que el formato del conjunto de píxeles sea compatible con la base de datos.
- *Tiempo de ejecución de queries* (float): El tiempo de ejecución total de todas las consultas en la base de datos se mide en segundos. Se utiliza el método "execute_concurrent" para paralelizar la ejecución de estas consultas y agilizar el proceso.
- *Tiempo de filtrado de resultados* (float): El tiempo de ejecución para el proceso de matcheo de coordenadas se mide en segundos. Durante este proceso, se comparan las coordenadas de los objetos resultantes de las consultas con las coordenadas del archivo de entrada. Solo se seleccionan aquellos objetos que cumplen la condición de tener una distancia menor al radio establecido. Además, se aplica el límite (limit) en caso de ser necesario. Finalmente, se genera el archivo de salida en formato JSON, transformando los resultados desde un dataframe CSV.
- *Tiempo de ejecución total* (float): El tiempo total de ejecución para el proceso de crossmatching se mide en segundos. Este tiempo engloba todas las etapas del proceso, desde la preparación de los datos de entrada hasta la generación de los resultados finales. Incluye la lectura y procesamiento de los archivos, la ejecución de las consultas en la base de datos, el matcheo de coordenadas y la generación del resultado final.
- *Cantidad de píxeles consultados* (int): El número total de píxeles representa la cantidad de unidades utilizadas para describir el área de búsqueda en el catálogo astronómico. Estos píxeles se generan de acuerdo con la metodología específica del catálogo y se utilizan para realizar consultas eficientes en la base de datos Cassandra y obtener los objetos requeridos. Si se tienen múltiples objetos dentro de un píxel, este cuenta solo una vez en este resultado.
- *Cantidad de objetos sin filtrar* (int): El recuento total de objetos representa la cantidad de resultados obtenidos al ejecutar las consultas en la base de datos. Estos resultados requieren filtrado adicional y opcionalmente la aplicación de un límite para seleccionar los objetos más cercanos antes de ser devueltos como parte del resultado final.

- *Cantidad de objetos resultantes* (int): La cuenta total de objetos finales corresponde al número de resultados obtenidos después de aplicar el proceso de filtrado y, en caso de haber especificado un límite, seleccionar únicamente la cantidad indicada de objetos más cercanos. Esta cuenta representa la cantidad final de objetos que se devolverán como resultado en la consulta.

- *Objetos resultantes* (json): Los objetos resultantes del filtrado se encuentran en un formato inicial de un dataframe de pandas, el cual contiene la información almacenada en la tabla de Cassandra correspondiente al objeto matcheado. Posteriormente, se convierte este dataframe al formato JSON para ser devuelto como salida en este endpoint. Además de la información original, se agrega información adicional sobre las posiciones Ra y Dec de la coordenada del input con la cual el objeto fue matcheado, así como la distancia entre estas dos posiciones.

  Un ejemplo de una posible respuesta se presenta a continuación:


```
{
'Tiempo pre-procesamiento': '0.001413591904565692',
'Tiempo ejecucion queries': '0.005155744031071663',
'Tiempo filtrado resultados:': '0.022907204926013947',
'Tiempo ejecucion total:': '0.029639674816280603',
'Cantidad de pixeles consultados:': '1',
'Cantidad de objetos sin filtrar:': '1',
'Cantidad de objetos resultantes:': '1',
'dataframe': '[
    {
        "nside14":"1910368193",
        "source_id":"2839m137_b0084044",
        "dec":"-12.898655245006",
        "pmdec":"0.024325526",
        "pmra":"-0.0009243975",
        "ra":"284.68059540349924",
        "sigdec":"0.0502",
        "sigpmdec":"0.0168",
        "sigpmra":"0.0157",
        "sigra":"0.0496",
        "source_name":"J185843.33-125355.1",
        "w1flg":"1",
        "w1k":"0.88514",
        "w1mag":"15.241",
        "w1mlq":"29.25",
        "w1sigm":"1.0",
        "w2flg":"32",
        "w2k":"0.86002",
        "w2mag":"15.361",
        "w2mlq":"1.35",
        "w2sigm":null,
        "Ra conesearch":284.6805954035,
```

```
            "Dec conesearch":-12.898655245,
            "Distance arcsec":0.0
        }]'
    }
```

Crossmatch con output CSV (método POST)

El endpoint proporciona una lista de los objetos resultantes del proceso de crossmatching entre un catálogo específico y una lista de posiciones. Se puede acceder a través de la URL "/crossmatch_csv". El formato de entrada es un archivo .csv, al igual que el formato de salida. Los parámetros para este endpoint, omitiendo la descripción de aquellos que se mantienen igual al endpoint anterior, son los siguientes:

- *radius* (float)
- *unit* (Unit)
- *catalog* (Catalog)
- *limit* (int, opcional)
- *input_file* (archivo .csv): Para este endpoint, el conjunto de coordenadas a matchear se proporciona en un archivo .CSV. El archivo tiene como encabezado "ra,dec", y cada fila contiene un par de coordenadas "ra" y "dec" separadas por una coma.

  Y la respuesta a la solicitud proporciona los siguientes campos:

- *Dataframe* (.CSV): La salida de este endpoint es el dataframe obtenido al final del proceso de crossmatching, el cual se devuelve directamente utilizando StreamingResponse. No obstante, otros campos como los tiempos de ejecución o la cantidad de píxeles no se pueden incluir en este dataframe, ya que el formato CSV está destinado a ser utilizado directamente en el pipeline de ALeRCE, sino que, en cambio, se imprimen al finalizar el proceso. Este CSV está conformado por las 24 columnas que aparecen en el ejemplo de la salida anterior en la sección de dataframe como encabezado, y cada fila los valores correspondientes a cada objeto resultante.

Capítulo 6

Experimentos y resultados

Con el propósito de validar la utilidad del nuevo servicio para los objetivos de ALeRCE, se realizaron pruebas para evaluar su tiempo de ejecución en comparación con CDS y determinar si es igual o incluso mejor. Asimismo, es crucial garantizar la precisión de los resultados, es decir, verificar que los objetos detectados están realmente dentro del radio de búsqueda al realizar la consulta. Además, se llevó a cabo una verificación de los resultados obtenidos utilizando el parámetro limit, con el fin de asegurarse de que los resultados finales correspondan a los más cercanos. Para mejorar los tiempos de ejecución, se implementaron scripts que permiten paralelizar las solicitudes dividiendo los archivos de prueba y utilizando hilos para generar solicitudes simultáneas.

La máquina en la cual se cargó el catálogo y se construyó el sistema es un servidor con 96 CPU. Cada CPU es un procesador Intel(R) Xeon(R) Gold 5220R con 24 núcleos. La memoria RAM instalada es de 376 GB. Para el almacenamiento, se utilizaron dispositivos de estado sólido (SSD), teniendo disponibles 56TB. Dentro de este servidor se utilizó un contenedor de Docker con imagen ubuntu, donde se instaló Cassandra para almacenar los datos dentro del mismo contenedor donde se monta la API. Este servidor corresponde al servidor de ALeRCE, donde se montará el pipeline completo en el futuro.

6.1. Explicación de experimentos y datos utilizados

Con el propósito de realizar los experimentos, se recolectaron diversas muestras de ubicaciones en el firmamento, los cuales se describen a continuación:

1. 20.000 coordenadas seleccionadas aleatoriamente.
2. 30.000 coordenadas de objetos de CatWISE2020.
3. 450 coordenadas en el plano galáctico de la Vía Láctea.
4. 180 coordenadas en el plano perpendicular al plano galáctico.
5. 1.000 coordenadas a un máximo de 5° del Ecuador.

6. 1.000 coordenadas a un máximo 5° de los polos.

Para el primer conjunto de datos, se utilizó un random de Python para generar valores aleatorios en pares Ra/Dec. Los valores de Ra podían variar entre 0° y 360°, mientras que los valores de Dec podían variar entre −90° y 90°, abarcando así toda la esfera de coordenadas.

Por otro lado, el segundo conjunto de datos se generó utilizando un valor aleatorio entre 0 y el número total de píxeles de una de las metodologías de pixelización, en este caso, HEALPix con un índice nside de 14. Esto implica que el valor aleatorio se encontraba en el rango de 0 a 3,221,681,152. Con estos píxeles random se generaron consultas, y se guardaron únicamente los valores Ra/Dec de los objetos resultantes. En caso de que no se encontrara ningún objeto en un píxel específico, se repetía la consulta utilizando otro píxel hasta alcanzar un total de 30,000 coordenadas generadas.

Para los conjuntos 3 y 4 se trabajó utilizando astropy, donde se generaron coordenadas SkyCoord bajo el frame galáctico, y que posteriormente fueron transformadas a ICRS, que corresponde al frame utilizado por las coordenadas de CatWISE2020. Las coordenadas galácticas están conformadas por dos valores: longitud l cuyo valor va entre 0° y 360°, y la latitud b, cuyo rango es entre −90° y 90°. Para la muestra 3, se fijó el valor de b en cero, y los valores de l se separaron en intervalos de 0,8 grados. Por otro lado, para la muestra 4, se fijó el valor de l en cero, y se varió el valor de b en el rango de −90 a 90 grados, con una separación de 1 grado.

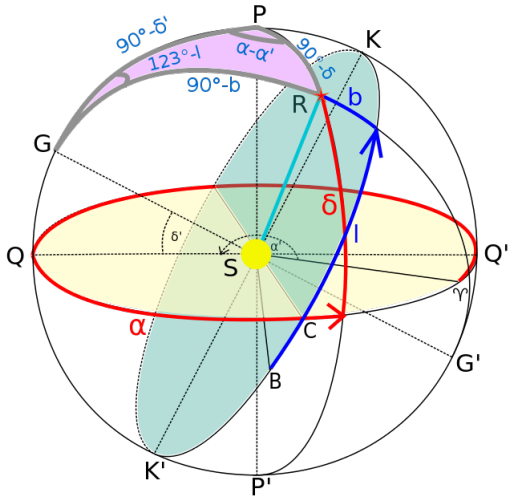

Figura 6.1: Orientación de sistema coordenadas ecuatorial y galáctico representados en la esfera [3].

En la figura 6.1 se muestra un diagrama que representa el sistema de coordenadas galáctico y el sistema de coordenadas ecuatorial. El plano amarillo corresponde al Ecuador celestial y el verde al plano galáctico. El triángulo esférico resaltado en rosado se conoce como el tercer triángulo astronómico y se utiliza para convertir entre los dos sistemas utilizando trigonometría.

Estos primeros dos conjuntos fueron seleccionados de esta forma para realizar una comparación respecto a los resultados obtenidos por Alejandra Alarcón en su memoria “Servicio

de Crossmatching de objetos Astronómicos"[1], quien utilizó cantidades similares en el caso de las muestras 1 y 2, y los samples 3 y 4 se construyeron a partir de los descritos en su memoria.

Por último, los conjuntos 5 y 6 fueron elaborados usando random, donde los valores de Ra se restringieron para que la ubicación de los objetos sea a lo más a 5° respecto de los polos o el Ecuador. El objetivo de este conjunto es permitir realizar una comparación con los resultados que se pueden obtener realizando crossmatch utilizando otras metodologías para realizar la proyección del cielo en un plano 2D, las cuales pueden presentar dificultades debido a los cálculos de las distancias de objetos cercanos a los polos.

Al realizar la proyección del cielo en un plano 2D, se produce una distorsión significativa en las regiones cercanas a los polos. En algunas de las técnicas de proyección usadas en astronomía, las distorsiones que se presentan al pasar desde coordenadas celestes a planos 2D en los polos se vuelven extremas. En estas regiones, las distancias angulares entre los objetos celestes se distorsionan enormemente, y algunas áreas pueden parecer estiradas o incluso infinitas.

Esta distorsión en los polos se debe a la naturaleza esférica del cielo y la necesidad de representarlo en un plano. Es importante tener en cuenta esta distorsión al utilizar mapas celestes o al realizar mediciones precisas de las posiciones de objetos celestes en las regiones polares. Los astrónomos suelen utilizar técnicas específicas para compensar esta distorsión cuando trabajan en estas áreas, como utilizar proyecciones adaptadas o realizar correcciones adicionales a las coordenadas usadas para sus mediciones.

Utilizando estos conjuntos de coordenadas como entrada, se realizaron consultas en el CDS y en el servicio desarrollado en este trabajo para radios de 1, 2, 4, 8, 16 y 60 arcosegundos. Los resultados de estas consultas se describirán en las secciones siguientes. En particular, se llevaron a cabo experimentos para cada radio, utilizando las dos metodologías de pixelización trabajadas: HEALPix y HTM. Además, se realizaron mediciones al alterar el parámetro "limit", con consultas sin límite, con limit = 1 y con limit = 3. Por último, se llevaron a cabo pruebas en las que se dividieron los archivos de coordenadas utilizados en 5 partes, las cuales se enviaron a solicitudes que se procesaron simultáneamente utilizando hilos, y se combinaron los resultados en el resultado final. Para cada una de estas configuraciones se realizaron 10 mediciones de los tiempos de ejecución, presentando los promedios de los tiempos en la sección de resultados.

6.2. Resultados

6.2.1. Correctitud

Para verificar la precisión, se compararon los resultados obtenidos de cada uno de los sistemas, refiriéndose a CDS, el servicio desarrollado durante este trabajo utilizando HEALPix, y el servicio utilizando HTM. Para realizar esta comparación, no se realizó un recuento directo de los resultados, ya que esto no garantiza la corrección de las coincidencias. En su

lugar, para cada objeto obtenido en uno de los sistemas, se verificó su existencia en los otros sistemas, comprobando si había un objeto con el mismo identificador de origen (source id), y las coordenadas de ra/decs de la entrada con las que un objeto del catálogo fue matcheado. No se verificó directamente la distancia, ya que no se tiene un conocimiento completo de la metodología utilizada por CDS para medir este valor, lo que podría dar lugar a pequeñas diferencias.

Debido a que CDS no proporciona una opción para limitar la cantidad de objetos matcheados por coordenada de entrada en su servicio de crossmatch web, se requirió crear scripts personalizados para validar el correcto funcionamiento de la variable "limit". En estos scripts, se utilizó la función "merge" para combinar los dataframes resultantes, ordenándolos por distancia y seleccionando los objetos de acuerdo con el valor de "limit". Esto permitió verificar que los objetos matcheados con las coordenadas de entrada fueran los más cercanos, asegurando así la correcta aplicación del límite establecido.

Utilizando los archivos CSV generados a partir de las consultas y utilizando estas herramientas, se pudo determinar que había una coincidencia del 100 % entre los resultados obtenidos de CDS y los resultados del sistema desarrollado en este trabajo utilizando los dos métodos de pixelización. Además, se verificó que los objetos resultantes seleccionados utilizando la variable "limit" eran efectivamente los más cercanos. Esta verificación requirió un tiempo de procesamiento adicional, pero se pudo confirmar que el 100 % de los objetos proporcionados eran los más cercanos de manera precisa.

Posiciones de objetos seleccionados de manera aleatoria

A continuación en la tabla 6.1 se presentan los resultados totales de coincidencias para la muestra de 50 mil puntos, que consiste en una combinación de una muestra de 20 mil coordenadas aleatorias del cielo y 30 mil coordenadas seleccionadas a partir de objetos del catálogo CatWISE2020. Es importante destacar que el número de coincidencias con "no limit" fue exactamente el mismo utilizando CDS, HTM y HEALPix. Además, se verificó que cada coincidencia tuviera las mismas coordenadas de entrada y el identificador del objeto del catálogo coincidente. Para los resultados de las coincidencias con "limit = 1" y "limit = 3", se confirmó que correspondían a las coincidencias más cercanas para cada coordenada de entrada.

Tabla 6.1: Conteo de matcheos coincidentes para la muestra de 50 mil puntos aleatorios entre el crossmatch en CDS y el sistema implementado.

Radio (arcsec)	**Matches (no limit)**	**Matches (limit=1)**	**Matches (limit=3)**
1	30.252	30.249	30.252
2	30.943	30.923	30.943
4	34.273	33.568	34.273
8	57.453	40.752	56.925
16	174.589	48.418	123.585
60	2.209.383	49.949	149.822

Coordenadas de objetos en el plano galáctico

Se debe destacar que las coordenadas de las muestras del plano galáctico y el plano perpendicular al galáctico, a pesar de haber sido generadas siguiendo la descripción de Alejandra Alarcón en su memoria, no son idénticas. Esto se evidencia en las diferencias en el número de resultados obtenidos por CDS para todos los radios utilizados descritos en su trabajo respecto a los valores presentados en las tablas 6.2 y 6.3. Por ejemplo, en el caso de utilizar un radio de 16 arcosegundos para las muestras en el plano galáctico y plano perpendicular se obtuvieron en este trabajo 1.911 y 532 matches en CDS respectivamente. Por otro lado, la muestra utilizada por Alejandra arrojó un total de 1.892 y 511 matches en CDS respectivamente para las muestras que debiesen tener la misma cantidad de resultados. Estas discrepancias pueden atribuirse a posibles conversiones de sistemas de referencia que se hayan manejado de manera diferente o a modificaciones en el formato de la muestra para que se ajuste a los requisitos de MongoDB en su trabajo. Por lo tanto, no es posible realizar una comparación directa de las coincidencias obtenidas.

Tabla 6.2: Conteo de matcheos coincidentes para la muestra de puntos en el plano galáctico entre el crossmatch en CDS y el sistema implementado.

Radio (arcsec)	**Matches (no limit)**	**Matches (limit=1)**	**Matches (limit=3)**
1	15	15	15
2	42	41	42
4	126	121	126
8	489	309	482
16	1.911	413	1.131
60	27.022	446	1.335

Coordenadas de objetos en el plano perpendicular al galáctico

Tabla 6.3: Conteo de matcheos coincidentes para la muestra de puntos en el plano perpendicular al plano galáctico entre el crossmatch en CDS y el sistema implementado.

Radio (arcsec)	**Matches (no limit)**	**Matches (limit=1)**	**Matches (limit=3)**
1	2	2	2
2	10	10	10
4	21	21	21
8	135	103	134
16	532	170	399
60	6.905	180	540

Coordenadas de objetos aleatorios cerca del Ecuador

Las muestras de coordenadas de objetos cerca del Ecuador y cerca de los polos contienen 1.000 puntos cada una, cuyos resultados se encuentran en las tablas 6.4 y 6.5 respectivamen-

te. Estos resultados permiten observar con mayor claridad los efectos de utilizar limit con distintos valores, particularmente en el caso de 1 arcominuto. Por ejemplo, utilizar limit 1 en ambos se puede observar que la cantidad de matches se acerca al número de coordenadas. También es posible observar que en el caso de limit 3, estos valores se acercan a 3×1.000 matches, lo cuál tiene sentido ya que no se espera que todas las coordenadas incluso con un radio relativamente grande como lo es 1 arcominuto tengan 3 matches siempre, pero si una gran mayoría.

Tabla 6.4: Conteo de matcheos coincidentes para la muestra de puntos hasta 5° del Ecuador entre el crossmatch en CDS y el sistema implementado.

Radio (arcsec)	**Matches (no limit)**	**Matches (limit=1)**	**Matches (limit=3)**
1	15	15	15
2	45	45	45
4	156	150	156
8	607	457	603
16	2.410	882	1.935
60	34.517	999	2.993

Coordenadas de objetos aleatorios cerca de los polos

Tabla 6.5: Conteo de matcheos coincidentes para la muestra de puntos hasta 5° de los polos entre el crossmatch en CDS y el sistema implementado.

Radio (arcsec)	**Matches (no limit)**	**Matches (limit=1)**	**Matches (limit=3)**
1	10	10	10
2	45	45	45
4	185	179	185
8	729	538	725
16	2.927	953	2.330
60	40.331	994	2.982

Basado en estos hallazgos, se puede concluir que los resultados del proceso de crossmatching han superado la prueba de precisión. Se logró obtener los mismos resultados que CDS en los casos en que fue posible realizar una comparación directa de los archivos CSV resultantes. Además, se confirmó la precisión de la variable "limit" en la selección de los resultados más cercanos mediante el uso de scripts que operaban en los dataframes. En la siguiente sección, se realizará una comparación de los tiempos de ejecución de las dos metodologías de pixelización, considerando diferentes muestras y parámetros, y se compararán estos tiempos con los obtenidos por CDS.

6.2.2. Eficiencia de procesar las consultas

Con el fin de comparar la velocidad, se midió el tiempo promedio para realizar 10 consultas utilizando la API implementada en este trabajo y la API de CDS X-Match, que se encuentra en la siguiente URL: http://cdsxmatch.u-strasbg.fr/xmatch/api/v1/sync. Si bien existe una versión web de la API de CDS, no proporcionaba una medición precisa de los tiempos de ejecución de las consultas, ya que solo mostraba el tiempo en segundos como un número entero, e incluso para conjuntos de datos más pequeños simplemente se indicaba "<1s". Por lo tanto, se optó por utilizar requests con la API de CDS para obtener mediciones más precisas, a través de medir la diferencia de tiempo entre enviar la request y recibir respuesta[1].

Dentro de esta sección interesa buscar respuestas a las siguientes preguntas:

- ¿Qué tanto varían los tiempos de ejecución con distintos valores de limit?
- ¿Es posible lograr una mejora utilizando hilos?
- ¿Cómo se comportan las mediciones de tiempo a medida que aumenta el radio?
- ¿Cómo se comportan los tiempos al aumentar o disminuir la cantidad de coordenadas respecto a CDS?
- ¿Cómo se comparan las mediciones del sistema con respecto a CDS utilizando las mejores configuraciones del sistema desarrollado?

Posiciones de objetos seleccionados de manera aleatoria

20 mil objetos aleatorios

El primer conjunto de coordenadas revisado fue el de 20 mil posiciones escogidas aleatoriamente, donde se compararon los tiempos promedios de ejecución de consultas para distintos radios, utilizando en el caso de la API desarrollada limit None para esta primera prueba. Los tiempos se miden desde que se genera la request, hasta que se escriben los resultados en un archivo csv. Los resultados se pueden observar en la figura 6.2.

Según los resultados obtenidos, se puede observar que en todos los casos el método HTM presenta un menor tiempo de ejecución en comparación con HEALPix. Sin embargo, la diferencia de tiempo entre HTM y HEALPix no es muy significativa. Esta diferencia puede explicarse al analizar la cantidad de píxeles que se consultan en cada método. En el caso de HTM, ocurren casos donde se consulta una cantidad ligeramente menor de píxeles en comparación con HEALPix. La mayor diferencia en los tiempos de ejecución se encuentra en el momento de realizar las consultas, debido precisamente a esta diferencia en la cantidad de píxeles consultados.

En cuanto a los tiempos de ejecución, se puede observar que los tiempos obtenidos por CDS son considerablemente mejores que los tiempos alcanzados por el sistema desarrollado,

[1] http://cdsxmatch.u-strasbg.fr/xmatch/doc/API-calls.html

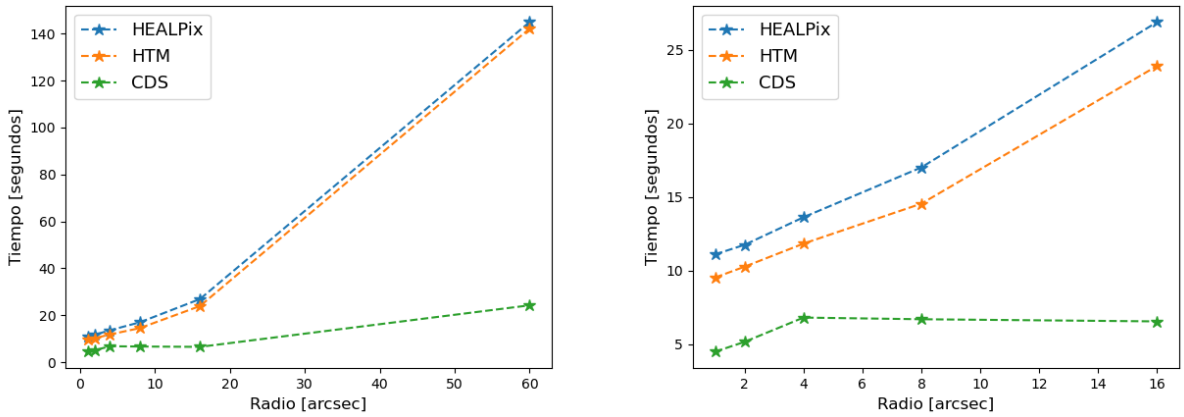

Figura 6.2: Izquierda: Plot comparación tiempos de ejecución crossmatch con el conjunto de 20 mil posiciones aleatorias, limit = None, Derecha: Mismo plot omitiendo radio de 1 arcominuto para facilitar la comparación de resultados en radios menores.

siendo de 4 a 5 veces más rápidos. A pesar de esta diferencia, ambos sistemas lograron cumplir con los requisitos mínimos para procesar los radios de 1 a 16 arcosegundos, con un promedio de procesamiento de entre 1800 y 740 consultas por segundo respectivamente. En el caso de las consultas de 1 arcominuto, se logró un promedio de casi 140 consultas por segundo, aunque este valor está por debajo del mínimo requerido de 350 consultas por segundo, que es necesario únicamente para las consultas de 1 arcosegundo.

Se puede observar una tendencia lineal en los tiempos de ejecución, aunque se aprecia un cambio significativo en la pendiente entre los 16 arcosegundos y 1 arcominuto. Es importante mencionar que la tendencia lineal de los resultados se asemeja a los tiempos obtenidos por Alejandra Alarcón en su trabajo, con una reducción de aproximadamente la mitad de los tiempos. Aunque no se puede realizar una comparación directa entre los resultados de este trabajo y el desarrollado por Alejandra debido a las diferencias en los entornos de trabajo, es interesante destacar que se logró alcanzar resultados más rápidos.

A continuación se presentan los resultados del conjunto de coordenadas más grande, que incluye 20 mil coordenadas aleatorias y 30 mil coordenadas de objetos del catálogo seleccionadas al azar. Este conjunto de datos es de especial interés debido a que más del 60 % de las coordenadas de entrada tienen al menos un resultado incluso con el menor radio posible. Esto significa que el tiempo de filtrado aumentaría considerablemente en el sistema desarrollado. Por esta razón, se realizaron múltiples pruebas específicas para este conjunto de datos, las cuales se encuentran en la subsección siguiente.

50 mil objetos aleatorios

La primera prueba consiste en calcular el promedio de los tiempos de procesamiento de 10 consultas por radio utilizando las API de CDS, HEALPix y HTM para los 50 mil puntos.

En la figura 6.3 se puede observar que la tendencia seguida tanto por HTM como por

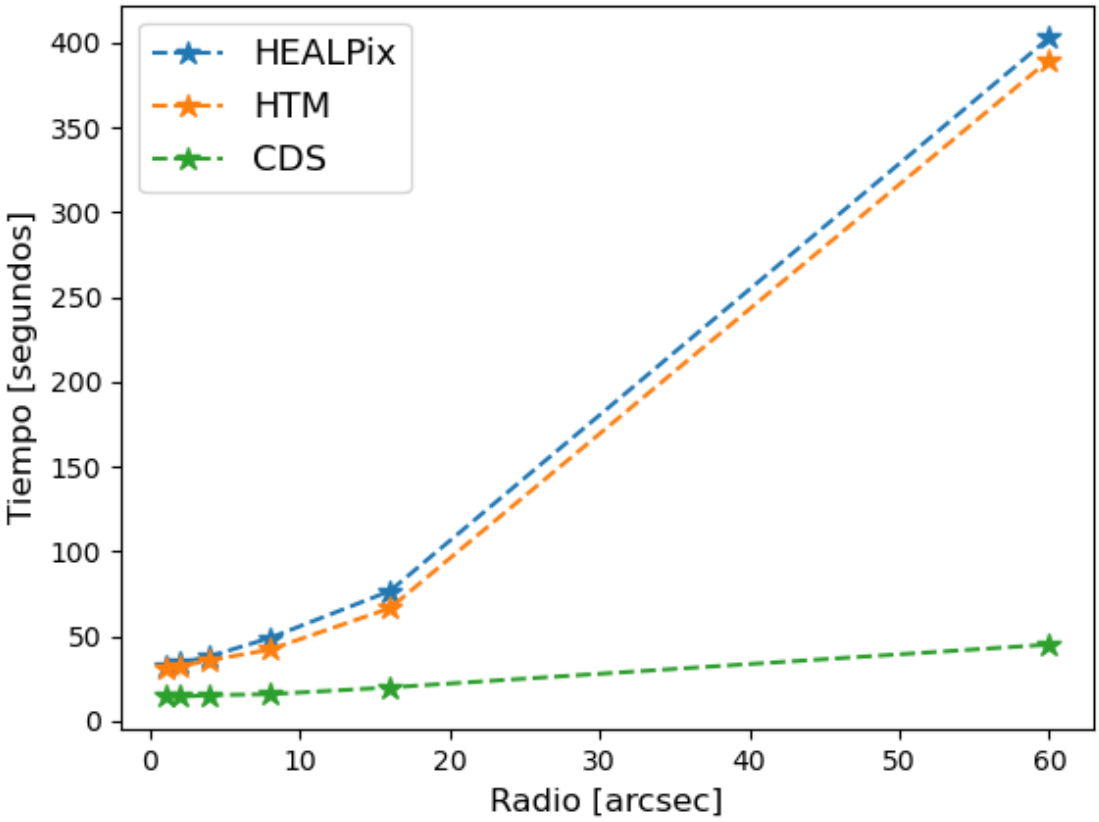

Figura 6.3: Plot comparación tiempos de ejecución crossmatch con el conjunto de 50 mil posiciones aleatorias limit = None.

HEALPix se asemeja a la observada en el caso de los 20 mil puntos aleatorios, a pesar de que la cantidad de objetos resultantes del proceso de crossmatching es mayor.

Un detalle que es interesante para responder a las preguntas planteadas al inicio de esta subsección es que el crecimiento de la curva a partir de los 16 arcosegundos no es directamente proporcional respecto al obtenido para 20.000 coordenadas. Este aumento significativo se debe a que al tener más coordenadas, el tiempo para matchear pares de coordenadas aumenta de forma no lineal. Esto es así dado que a cada coordenada de la entrada se le debe calcular un radio y comparar su distancia respecto a las coordenadas resultantes de la query a Cassandra.

La siguiente prueba por realizar fue dividir el archivo de 50 mil coordenadas, en 5 archivos de 10 mil coordenadas, y utilizando hilos de python, se enviaron paralelamente 5 requests con el mismo radio. Una vez que las 5 requests simultáneas terminan de procesar su subconjunto de datos, estos se fusionan en un único archivo, el cual se corroboró con los resultados originales, para asegurar que fueran exactamente iguales. Esta prueba nuevamente se desarrolló considerando la variable limit None, y los resultados se pueden observar en la figura 6.4.

Los resultados mostrados en la figura 6.4 indican que, para radios más pequeños, CDS no es tan eficiente como alternativa en comparación con el sistema desarrollado cuando se

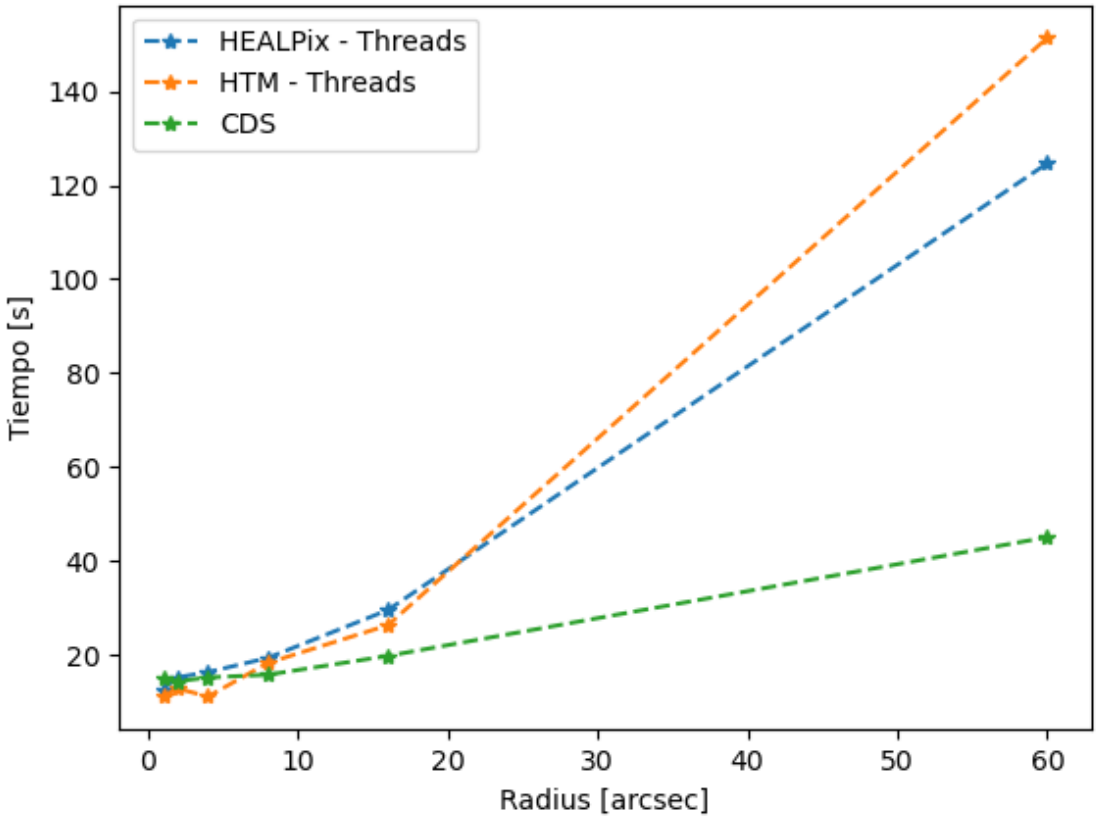

Figura 6.4: Plot comparación tiempos de ejecución crossmatch con el conjunto de 50 mil posiciones aleatorias utilizando hilos, limit = None.

utilizan HTM e hilos para acelerar los tiempos de respuesta. No obstante, tanto HTM como HEALPix resultaron más rápidos que CDS en el caso de 1 arcosegundo. A partir del radio de 8 arcosegundos, CDS se vuelve más rápido que los dos métodos desarrollados.

Cabe mencionar que, aunque se observa una reducción considerable del tiempo promedio al utilizar hilos, es necesario tener en cuenta que estos resultados son en promedio. Se observó variabilidad en los tiempos de ejecución, por lo que en la tabla 6.6 se muestran los promedios y la desviación estándar de los tiempos medidos para los diferentes radios utilizando las dos metodologías de pixelización. Se destaca el caso de HTM con 1 arcominuto, donde la variación es de alrededor de un 20 % del tiempo total promedio.

Dado que este es el conjunto más grande de datos, se procedió a realizar más pruebas, donde se compararon los tiempos de realizar peticiones con el sistema desarrollado alterando el valor de limit. En particular se trabajó realizando peticiones con limit = 1, limit = 3, y estos mismos experimentos pero utilizando hilos. El resultado para HEALPix se encuentra en la figura 6.5, y para HTM dada su similitud, en un anexo.

En la figura 6.5 se destaca que utilizar limit = 1 es más eficiente, independientemente de si se utilizan hilos o no. Esto se debe a que la función utilizada para hacer el match de las

Tabla 6.6: Promedios y desviación estándar en segundos de los tiempos para el conjunto de 50.000 coordenadas, utilizando hilos.

Radio arcsec	Prom. HEALPix	Std. HEALPix	Prom. HTM	Std. HTM
1	12,738	2,947	11,283	2,703
2	15,231	0,938	12,792	2,663
4	16,324	2,345	11,115	3,073
8	19,352	3,632	18,282	3,424
16	29,539	6,395	26,302	5,516
60	124,663	1,519	151,372	32,741

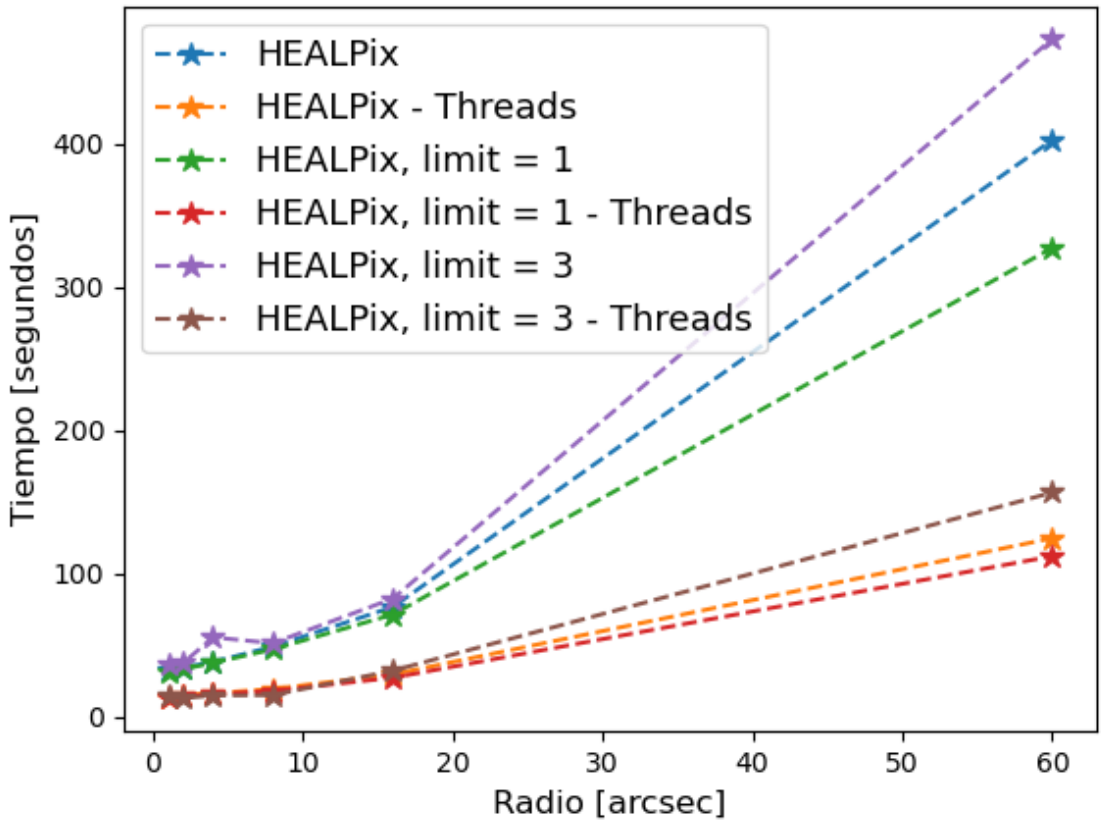

Figura 6.5: Plot comparación tiempos de ejecución crossmatch con el conjunto de 50 mil posiciones aleatorias, utilizando HEALPix variando limit y usando hilos.

coordenadas es diferente en el caso de limit = 1, ya que busca la coordenada más cercana en el catálogo para cada coordenada de entrada. En cambio, con un limit distinto de 1, la función utilizada debe consultar un radio para cada coordenada de entrada y devolver los índices de todas las coordenadas del catálogo que se encuentren dentro de ese radio.

Por otro lado, como se esperaba, limit = 3 es la opción más lenta en ambos casos. Esto se debe a que para seleccionar las 3 coordenadas más cercanas por cada coordenada de entrada, es necesario ordenar las coordenadas de menor a mayor, lo que implica un aumento de tiempo

debido al proceso de ordenamiento de las distancias resultantes. Además, se puede esperar que este tiempo aumente considerablemente en áreas más densas del espacio o en radios más grandes, ya que cada coordenada tendría más resultados que deben ser ordenados. Por último, se destaca que no se espera una variación importante en el tiempo entre tener un limit de 3 o 4, por ejemplo, ya que la mayor parte del tiempo adicional respecto a utilizar la variable limit = None se invierte en el proceso de ordenamiento de las coordenadas y no en extraer las limit más cercanas y agregarlas al conjunto final.

Con base en estos resultados, se puede inferir que la configuración más óptima para el crossmatch dependerá de los requisitos específicos que se tengan. En general, utilizar limit = 1 y dividir los archivos para enviarlos en paralelo mediante hilos se muestra como una opción prometedora. No obstante, debido a ciertos requerimientos particulares con respecto a los resultados, es posible que no siempre se pueda utilizar exclusivamente limit = 1. Para abordar esta situación, se realizó una comparación en la figura 6.6 entre las consultas utilizando las dos metodologías desarrolladas, tanto con limit = None como con limit = 1, en relación con los tiempos obtenidos en CDS.

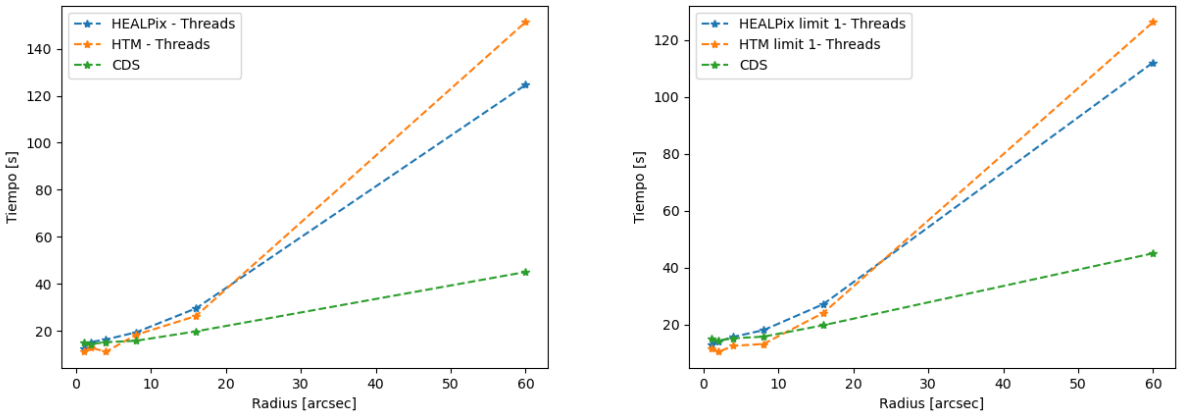

Figura 6.6: Izquierda: Plot comparación tiempos de ejecución crossmatch con el conjunto de 50 mil posiciones aleatorias, limit None y usando hilos. Derecha: Plot comparación tiempos de ejecución crossmatch con el conjunto de 50 mil posiciones aleatorias, limit = 1 y usando hilos.

A partir de estos resultados se puede observar que dentro de los radios más pequeños, utilizar hilos hace que los tiempos de ejecución CDS y el sistema desarrollado sean cercanos, sin embargo, para un radio de 1 arcominuto se tiene que existe una gran diferencia en los tiempos requeridos, requiriendo hasta el triple del tiempo para procesar las consultas.

Estos resultados responden algunas de las inquietudes planteadas al inicio de la subsección, sin embargo aun queda pendiente estudiar cómo varían los tiempos a medida que se trabaja con una menor cantidad de coordenadas en regiones particulares del cielo. Se pudo observar que existe un aumento considerable cuando se trabajó con el conjunto de 50 mil coordenadas frente al inicial de 20 mil, donde estudiando la distribución de los tiempos durante las fases de ejecución, se encontró que existe un mayor uso del tiempo en la fase de matching y filtrado que no responderían a un aumento de tiempo proporcional al aumento de coordenadas. Para responder a estas interrogantes se trabajará con otros de los conjuntos de coordenadas

utilizado, los cuales tienen menor cantidad de coordenadas. En los casos de figuras muy similares, se pueden encontrar junto a una breve descripción en las figuras del anexo.

Posiciones en el plano galáctico

A continuación se trabajó con el conjunto de coordenadas en el plano galáctico, correspondiente a 450 coordenadas. Para este conjunto se realizaron pruebas utilizando los métodos de pixelización con limit = None sin threads y posteriormente con threads, cuyos resultados se encuentran dentro de la figura 6.7. Según la imagen izquierda de la figura 6.7, se puede observar que los métodos desarrollados en este trabajo son capaces de procesar el crossmatch con las posiciones en el plano galáctico hasta un radio de 16 arco segundos más rápidamente que CDS. Sin embargo, a partir de 1 arco minuto, se vuelven aproximadamente 1,5 segundos más lentos en comparación con CDS. Un efecto similar se puede observar en el caso de las posiciones en el plano perpendicular al plano galáctico presente en el anexo, donde se lograron obtener resultados más rápidos que CDS hasta un radio de 16 arco segundos, pero nuevamente CDS fue más rápido con el radio más grande utilizado.

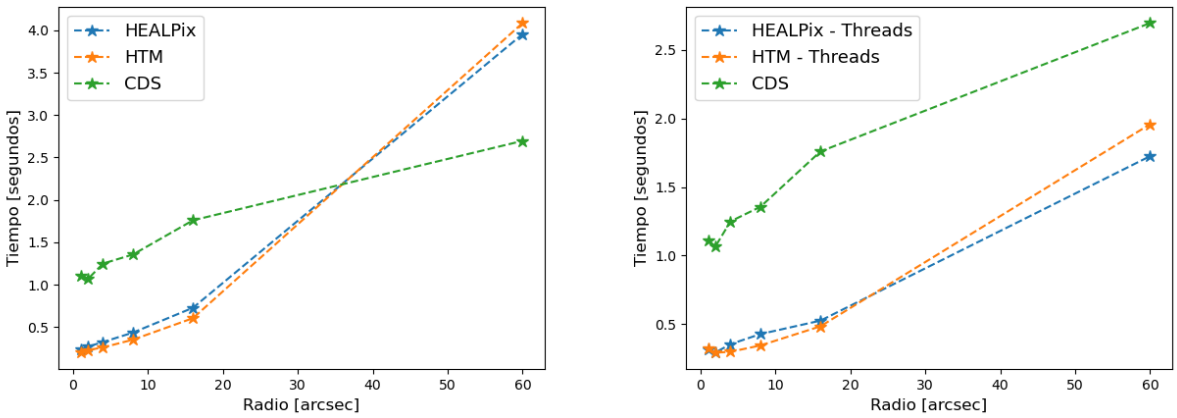

Figura 6.7: Izquierda: Plot comparación tiempos de ejecución crossmatch con el conjunto de posiciones en el plano galáctico, limit None. Derecha: Plot comparación tiempos de ejecución crossmatch con el conjunto de posiciones en el plano galáctico, limit None usando hilos.

Esto llevó a investigar qué sucedería al utilizar hilos. La teoría inicial era que el tiempo de ejecución podría volverse más lento en el caso de radios pequeños para el sistema desarrollado, ya que se consumiría más tiempo en preparar los hilos y fusionar los resultados. Utilizar hilos aceleraría los procesos de matching y filtrado, los cuales en el caso de radios pequeños con pocas coordenadas, son muy rápidos y por tanto, el aumento de tiempo requerido para preparar hilos y fusionar las respuestas sería mayor que el tiempo que se ahorra en las fases de procesamiento ya mencionadas. Sin embargo, se espera que esto se compense con una reducción en el tiempo requerido para procesar consultas de 1 arco minuto, lo que llevaría a una distribución de tiempos más cercana a la de CDS.

En el caso de utilizar hilos, en la imagen derecha de la figura 6.7, se logró obtener un mejor tiempo de ejecución en comparación con CDS, utilizando los mismos parámetros para todos

los radios utilizados. Aunque se esperaba que los tiempos aumentaran para los radios más pequeños debido al procesamiento adicional requerido, la diferencia no fue lo suficientemente significativa como para que los tiempos medidos para HTM y HEALPix superaran los tiempos alcanzados por CDS, e incluso aumentando en un tercio del tiempo requerido para procesar la información en algunos de estos casos, se mantiene siendo más rápido con un margen no menor.

Esto indica que sería posible utilizar hilos para enviar archivos divididos por partes y procesarlos de manera más eficiente y rápida que utilizando CDS como sistema de crossmatch de coordenadas, sin embargo este efecto solo ocurrió con un número menor de coordenadas que los utilizados en los primeros dos samples. Esto permitiría aprovechar el sistema desarrollado en este trabajo de manera más eficiente que utilizando CDS actualmente, para acelerar el proceso del pipeline en ALeRCE, además de asegurar que este sea capaz de procesar en vivo todas las señales provenientes de los telescopios de manera efectiva y eficaz.

Coordenadas de objetos aleatorios cerca del Ecuador

Al procesar conjuntos de coordenadas cercanas al Ecuador, correspondiente a 1.000 coordenadas, se observa que CDS es más eficiente que el sistema desarrollado para conjuntos más grandes de coordenadas cuando se utilizan radios mayores. Esto se refleja en los tiempos de crossmatch, donde CDS logra mejores tiempos, como se puede apreciar en la imagen izquierda de la figura 6.8, alcanzando mejores resultados en términos de velocidad de procesamiento para un radio de un arco minuto.

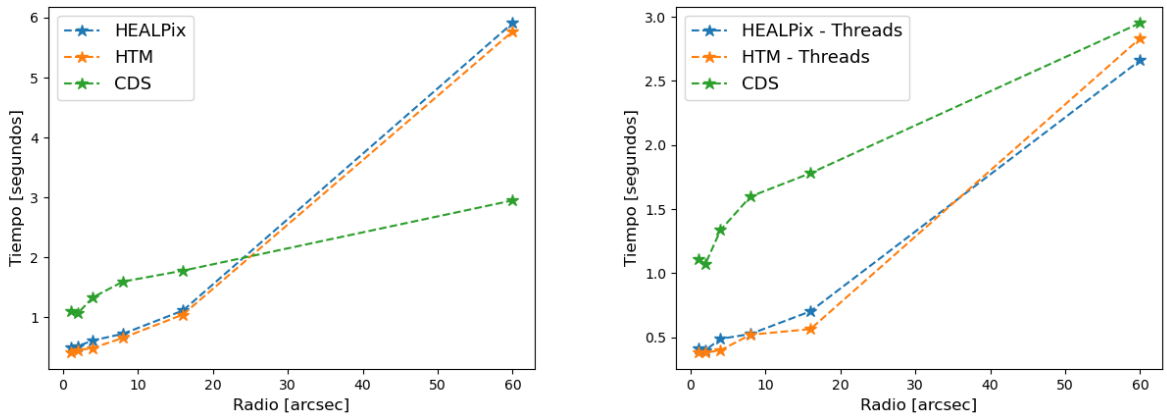

Figura 6.8: Izquierda: Plot comparación tiempos de ejecución crossmatch con el conjunto de posiciones cercanas al Ecuador, limit None. Derecha: Plot comparación tiempos de ejecución crossmatch con el conjunto de posiciones cercanas al Ecuador, limit None usando hilos.

Sin embargo, al utilizar hilos es posible lograr que los tiempos de procesamiento de la aplicación desarrollada sean nuevamente inferiores en todo el conjunto de radios utilizados, manteniendo un comportamiento lineal más consistente con lo observado en el resto de los tiempos medidos, pudiendo notar nuevamente un leve aumento en la pendiente al utilizar un radio de 1 arcominuto. El incremento de tiempo para el radio más grande se debe al

considerable aumento de objetos obtenidos como resultado de las consultas a Cassandra, lo que impacta principalmente en el tiempo necesario para realizar el crossmatch entre todas las coordenadas de entrada y los resultados obtenidos.

Comparando ambas imágenes de la figura 6.8, se puede observar que la mejora de tiempo para el radio de 1 arcominuto no es tan significativa como en el caso de los objetos en el plano galáctico. Al analizar los resultados de las tablas, donde se observa que la cantidad de objetos resultantes para el conjunto de posiciones cercanas al Ecuador con un radio de 1 arcominuto es mayor que en el conjunto del plano galáctico bajo el mismo radio de consulta, se puede inferir que el proceso de matching y filtrado realizado por CDS es más eficiente y/o presenta una menor variabilidad en sus mediciones en comparación con el sistema desarrollado.

De acuerdo con este resultado en particular, se puede concluir que un aspecto importante a mejorar en el servicio desarrollado es la optimización del proceso de matching y filtrado. Además, sería recomendable considerar la implementación de un nuevo endpoint que permita recibir archivos, dividirlos en conjuntos más pequeños y realizar el crossmatching utilizando hilos para ejecutar las solicitudes en paralelo desde el servidor, en lugar de hacerlo desde el cliente como se ha hecho hasta ahora en las pruebas realizadas. Esto implicaría realizar un estudio detallado sobre el número óptimo de hilos a utilizar, en función de la cantidad de coordenadas de entrada, así como determinar la cantidad de hilos del servidor que se pueden emplear para el procesamiento del crossmatching sin afectar el resto del pipeline de ALeRCE.

Discusión general

La información presentada a través de las distintas pruebas sugiere que utilizar 1.000 coordenadas como entrada para el proceso de crossmatch con un radio de 1 arco minuto se acerca al límite donde el sistema desarrollado deja de ser más eficiente que CDS en términos de tiempos de respuesta. Sin embargo, es importante destacar que esta conclusión es preliminar y requiere realizar más pruebas para obtener una conclusión definitiva. No obstante, estos resultados son un indicador relevante para buscar formas de maximizar los tiempos de respuesta en el proceso de crossmatch. Sería beneficioso realizar más investigaciones y optimizaciones para mejorar el rendimiento del sistema, en particular del matching de coordenadas y filtrado de resultados, en casos con conjuntos de coordenadas más grandes.

Con todos los resultados presentados a lo largo de este trabajo se pueden listar las siguientes conclusiones alcanzadas de acuerdo con los experimentos:

- Se encontró que los niveles de resolución óptimos no dependían mayormente del radio para los radios de las consultas utilizados, sino que los óptimos son similares para todo el rango.
- Se encontró que no existen diferencias mayores entre utilizar HEALPix y utilizar HTM con el sistema desarrollado, dado que las mayores diferencias de tiempo ocurren durante el proceso de filtrado y matcheo de objetos.
- Se puede concluir que para los primeros valores de radio, a medida que se aumenta el radio existe una tendencia lineal con respecto al aumento de los tiempos de ejecución

aún si el crecimiento de la mayoría de los radios utilizados fue exponencial. No obstante, la pendiente de esta recta aumenta considerablemente en el paso de 16 arcosegundos a 1 arcominuto. Se puede esperar una tendencia cuadrática, dado que al aumentar el radio, el área del cono aumenta de acuerdo a $\pi \times radio^2$, por lo que se puede suponer que el número de coordenadas aumenta de manera análoga, con lo que el tiempo para procesar las peticiones debería seguir una tendencia similar. Esto se puede corroborar usando un rango más largo y continuo de radios de input, no obstante, el foco de interés en este trabajo fue lograr el mínimo requerido de 350 consultas por segundo para un radio de arcosegundo, y lograr procesar consultas de 1 arcominuto.

- Se corroboró que utilizar la variable para limitar la cantidad de objetos aumenta los tiempos en el caso de que limit se utilicé con un valor distinto a 1. El caso de limit 1 corresponde a los mejores tiempos de respuesta.
- Se descubrió que CDS presenta mejores tiempos para todos los radios consultados con archivos de tamaño más grande, en este caso se usaron 20 mil coordenadas y 50 mil coordenadas.
- Se encontró que con tamaños pequeños de archivos (menos de 1.000 coordenadas) el sistema desarrollado es capaz de mejorar los tiempos frente a los de CDS con la excepción del radio de 1 arcominuto, donde se tiene que CDS supera ampliamente los alcanzados por HTM o HEALPix.
- Se pudo mejorar los tiempos utilizando hilos para todas las configuraciones del sistema, logrando superar los tiempos de CDS con pocas coordenadas de entrada.
- Se pudo concluir que la configuración más rápida para lograr resultados sería utilizar hilos, con limit = 1 y una cantidad de coordenadas de no más de mil por archivo de entrada para las peticiones.

Capítulo 7

Conclusión

7.1. Trabajo Realizado

El resultado final de esta memoria fue la creación de una API utilizando FastAPI que permite realizar consultas de crossmatching, especialmente enfocada en el catálogo CatWISE2020 y su integración con el servidor de ALeRCE. Se llevó a cabo un estudio exhaustivo de los tiempos utilizando diferentes parámetros de resolución de píxeles en las técnicas de HEALPix y Hierarchical Triangular Mesh.

A lo largo del proyecto, se encontró necesario implementar ajustes inesperados en la planificación inicial. Estos cambios incluyeron la transición de la librería healpix de astropy a la librería healpy para realizar los cálculos de HEALPix, la corrección de desviaciones en los valores de Ra/Dec de los objetos en el catálogo mediante el uso de valores específicos para cada tile del cielo, y una reestructuración de las tablas de la base de datos tras un análisis de los tiempos de ejecución en función de las variables nside y htm depth para HEALPix y HTM, respectivamente. A pesar de estos desafíos, se lograron abordar sin contratiempos significativos ni demoras sustanciales en relación con el plan establecido.

Los objetivos planteados en el primer capítulo se cumplieron en su totalidad y sirvieron como base para el desarrollo paso a paso de la aplicación, lo que permitió comprender y superar las complejidades y dificultades inherentes al problema. En términos de la cantidad mínima de consultas procesadas por segundo, se superó ampliamente el requisito mínimo de 350 consultas por segundo para un arco segundo, llegando a procesar más de 1.000 consultas por segundo. En el caso de un arco minuto, aunque no existía un mínimo establecido, se logró un promedio de 320 consultas por segundo utilizando threads, acercándose considerablemente al objetivo sin la restricción.

Es importante destacar que en todos los casos de prueba se logró un 100 % de coincidencia con los resultados de CDS, lo cual es un indicador positivo de la precisión de los resultados obtenidos. En cuanto a los resultados de las pruebas de velocidad de ejecución, se observó que la rapidez de ejecución no se acerca a la capacidad de CDS para conjuntos de datos grandes. Sin embargo, se demostró que con muestras pequeñas de hasta 1.000 coordenadas es posible paralelizar la ejecución dividiendo los archivos en partes y lograr una mayor eficiencia en la

resolución de los matcheos para todos los radios estudiados.

7.2. Trabajo futuro y optimizaciones posibles

Dentro de las posibles mejoras y opciones de desarrollo para fortalecer el sistema, se consideran las siguientes alternativas:

- Investigar y optimizar el proceso de matching de coordenadas resultantes de las consultas a la base de datos de Cassandra con las coordenadas de entrada, así como mejorar el filtrado de los resultados. Este paso en particular ha mostrado un cambio en la pendiente de los gráficos para el radio de 1 arco minuto.
- Realizar un estudio exhaustivo del equilibrio entre la cantidad de datos y la cantidad de hilos disponibles para el proceso de crossmatch. Esto permitiría implementar una funcionalidad en el pipeline del crossmatch para dividir los archivos de entrada y procesar las consultas en paralelo desde el lado del servidor. Esto facilita el envío de solicitudes y evita que el cliente tenga que gestionarlas.
- Implementar el escalamiento horizontal en Cassandra, lo cual mejoraría la disponibilidad de los datos y aumentaría el rendimiento de manera escalable con el número de nodos. Además, proporciona mayor tolerancia a fallos y permitiría la distribución de datos en diferentes zonas geográficas, garantizando una mayor disponibilidad.
- Una posible mejora sería explorar alternativas para automatizar el proceso de estudio de los niveles de resolución, generación de tablas e inserción de datos para otros catálogos. Sin embargo, se debe tener en cuenta que los catálogos astronómicos no siguen un estándar completamente uniforme en cuanto al manejo de columnas o los nombres de encabezados de las columnas, lo que dificulta la automatización completa del proceso. No obstante, se pueden desarrollar enfoques flexibles que permitan adaptarse a diferentes formatos de catálogos y establecer reglas específicas para el manejo de columnas. Esto podría incluir técnicas de análisis y extracción de datos, así como configuraciones personalizadas para cada catálogo en particular. De esta manera, se podría lograr un grado significativo de automatización y facilitar la integración de nuevos catálogos en el sistema.
- Una mejora propuesta es unificar los endpoints de acuerdo al formato de entrada y salida, permitiendo así una mayor flexibilidad para recibir nuevos formatos de entrada y proporcionando más opciones de formatos de salida para las tablas resultantes. Para lograr esto, se puede agregar un nivel de abstracción en la API que sea capaz de identificar el formato de entrada y adaptándose de manera de poder manejarlo adecuadamente, además de entregar como opción de formatos de salida aquellos formatos de catálogos más comunes en la astronomía.
- Desarrollar una interfaz web que facilite a astrónomos e investigadores que trabajen con los datos de ALeRCE obtener resultados de crossmatch de forma intuitiva y eficiente. Esta representaría una herramienta valiosa para agilizar el procesamiento de información necesaria en sus investigaciones. Esta interfaz les brindará la capacidad de realizar

consultas de crossmatch de manera sencilla, permitiéndoles acceder rápidamente a los resultados deseados, optimizando su flujo de trabajo.

Bibliografía

[1] A Alarcón Valenzuela. Servicio de crossmatching de objetos astronómicos. `https://repositorio.uchile.cl/handle/2250/188808`, 2022. [Online].

[2] Eric C. Bellm, Shrinivas R. Kulkarni, Matthew J. Graham, Richard Dekany, Roger M. Smith, Reed Riddle, Frank J. Masci, George Helou, Thomas A. Prince, Scott M. Adams, C. Barbarino, Tom Barlow, James Bauer, Ron Beck, Justin Belicki, Rahul Biswas, Nadejda Blagorodnova, Dennis Bodewits, Bryce Bolin, Valery Brinnel, Tim Brooke, Brian Bue, Mattia Bulla, Rick Burruss, S. Bradley Cenko, Chan-Kao Chang, Andrew Connolly, Michael Coughlin, John Cromer, Virginia Cunningham, Kishalay De, Alex Delacroix, Vandana Desai, Dmitry A. Duev, Gwendolyn Eadie, Tony L. Farnham, Michael Feeney, Ulrich Feindt, David Flynn, Anna Franckowiak, S. Frederick, C. Fremling, Avishay Gal-Yam, Suvi Gezari, Matteo Giomi, Daniel A. Goldstein, V. Zach Golkhou, Ariel Goobar, Steven Groom, Eugean Hacopians, David Hale, John Henning, Anna Y. Q. Ho, David Hover, Justin Howell, Tiara Hung, Daniela Huppenkothen, David Imel, Wing-Huen Ip, Željko Ivezić, Edward Jackson, Lynne Jones, Mario Juric, Mansi M. Kasliwal, S. Kaspi, Stephen Kaye, Michael S. P. Kelley, Marek Kowalski, Emily Kramer, Thomas Kupfer, Walter Landry, Russ R. Laher, Chien-De Lee, Hsing Wen Lin, Zhong-Yi Lin, Ragnhild Lunnan, Matteo Giomi, Ashish Mahabal, Peter Mao, Adam A. Miller, Serge Monkewitz, Patrick Murphy, Chow-Choong Ngeow, Jakob Nordin, Peter Nugent, Eran Ofek, Maria T. Patterson, Bryan Penprase, Michael Porter, Ludwig Rauch, Umaa Rebbapragada, Dan Reiley, Mickael Rigault, Hector Rodriguez, Jan van Roestel, Ben Rusholme, Jakob van Santen, S. Schulze, David L. Shupe, Leo P. Singer, Maayane T. Soumagnac, Robert Stein, Jason Surace, Jesper Sollerman, Paula Szkody, F. Taddia, Scott Terek, Angela Van Sistine, Sjoert van Velzen, W. Thomas Vestrand, Richard Walters, Charlotte Ward, Quan-Zhi Ye, Po-Chieh Yu, Lin Yan, and Jeffry Zolkower. The Zwicky Transient Facility: System Overview, Performance, and First Results. , 131(995):018002, January 2019.

[3] Burga. Equatorial galactic coordinates transformation. `https://commons.wikimedia.org/wiki/File:Equatorial_galactic_coordinates_transformation.svg`, 2012.

[4] K. C. Chambers, E. A. Magnier, N. Metcalfe, H. A. Flewelling, M. E. Huber, C. Z. Waters, L. Denneau, P. W. Draper, D. Farrow, D. P. Finkbeiner, C. Holmberg, J. Koppenhoefer, P. A. Price, A. Rest, R. P. Saglia, E. F. Schlafly, S. J. Smartt, W. Sweeney, R. J. Wainscoat, W. S. Burgett, S. Chastel, T. Grav, J. N. Heasley, K. W. Hodapp, R. Jedicke, N. Kaiser, R. P. Kudritzki, G. A. Luppino, R. H. Lupton, D. G. Monet, J. S. Morgan, P. M. Onaka, B. Shiao, C. W. Stubbs, J. L. Tonry, R. White, E. Bañados, E. F.

Bell, R. Bender, E. J. Bernard, M. Boegner, F. Boffi, M. T. Botticella, A. Calamida, S. Casertano, W. P. Chen, X. Chen, S. Cole, N. Deacon, C. Frenk, A. Fitzsimmons, S. Gezari, V. Gibbs, C. Goessl, T. Goggia, R. Gourgue, B. Goldman, P. Grant, E. K. Grebel, N. C. Hambly, G. Hasinger, A. F. Heavens, T. M. Heckman, R. Henderson, T. Henning, M. Holman, U. Hopp, W. H. Ip, S. Isani, M. Jackson, C. D. Keyes, A. M. Koekemoer, R. Kotak, D. Le, D. Liska, K. S. Long, J. R. Lucey, M. Liu, N. F. Martin, G. Masci, B. McLean, E. Mindel, P. Misra, E. Morganson, D. N. A. Murphy, A. Obaika, G. Narayan, M. A. Nieto-Santisteban, P. Norberg, J. A. Peacock, E. A. Pier, M. Postman, N. Primak, C. Rae, A. Rai, A. Riess, A. Riffeser, H. W. Rix, S. Röser, R. Russel, L. Rutz, E. Schilbach, A. S. B. Schultz, D. Scolnic, L. Strolger, A. Szalay, S. Seitz, E. Small, K. W. Smith, D. R. Soderblom, P. Taylor, R. Thomson, A. N. Taylor, A. R. Thakar, J. Thiel, D. Thilker, D. Unger, Y. Urata, J. Valenti, J. Wagner, T. Walder, F. Walter, S. P. Watters, S. Werner, W. M. Wood-Vasey, and R. Wyse. The pan-starrs1 surveys, 2019.

[5] Peter R. M. Eisenhardt, Federico Marocco, John W. Fowler, Aaron M. Meisner, J. Davy Kirkpatrick, Nelson Garcia, Thomas H. Jarrett, Renata Koontz, Elijah J. Marchese, S. Adam Stanford, Dan Caselden, Michael C. Cushing, Roc M. Cutri, Jacqueline K. Faherty, Christopher R. Gelino, Anthony H. Gonzalez, Amanda Mainzer, Bahram Mobasher, David J. Schlegel, Daniel Stern, Harry I. Teplitz, and Edward L. Wright. The CatWISE preliminary catalog: Motions from WISE and NEOWISE data. *The Astrophysical Journal Supplement Series*, 247(2):69, apr 2020.

[6] F Förster, G Cabrera-Vives, E Castillo-Navarrete, PA Estévez, P Sánchez-Sáez, J Arredondo, FE Bauer, R Carrasco-Davis, M Catelan, F Elorrieta, et al. The automatic learning for the rapid classification of events (alerce) alert broker. *The Astronomical Journal*, 161(5):242, 2021.

[7] Françoise Genova, Daniel Egret, Olivier Bienaymé, François Bonnarel, Pascal Dubois, Pierre Fernique, Gérard Jasniewicz, Soizick Lesteven, Richard Monier, François Ochsenbein, et al. The cds information hub-on–line services and links at the centre de données astronomiques de strasbourg. *Astronomy and astrophysics supplement series*, 143(1):1–7, 2000.

[8] K. M. Gorski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. HEALPix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622(2):759–771, apr 2005.

[9] Krzysztof M. Gorski, Benjamin D. Wandelt, Frode K. Hansen, Eric Hivon, and Anthony J. Banday. The healpix primer, 1999.

[10] Z Ivezic, SM Kahn, JA Tyson, B Abel, E Acosta, R Allsman, D Alonso, Y Alsayyad, SF Anderson, J Andrew, JRP Angel, GZ Angeli, R Ansari, P Antilogus, C Araujo, R Armstrong, KT Arndt, P Astier, E Aubourg, N Auza, TS Axelrod, DJ Bard, JD Barr, A Barrau, JG Bartlett, AE Bauer, BJ Bauman, S Baumont, E Bechtol, K Bechtol, AC Becker, J Becla, C Beldica, S Bellavia, FB Bianco, R Biswas, G Blanc, J Blazek, RD Blandford, JS Bloom, J Bogart, TW Bond, MT Booth, AW Borgland, K Borne,

JF Bosch, D Boutigny, CA Brackett, A Bradshaw, WN Brandt, ME Brown, JS Bullock, P Burchat, DL Burke, G Cagnoli, D Calabrese, S Callahan, AL Callen, JL Carlin, EL Carlson, S Chandrasekharan, G Charles-Emerson, S Chesley, EC Cheu, H-F Chiang, J Chiang, C Chirino, D Chow, DR Ciardi, CF Claver, J Cohen-Tanugi, JJ Cockrum, R Coles, AJ Connolly, KH Cook, A Cooray, KR Covey, C Cribbs, W Cui, R Cutri, PN Daly, SF Daniel, F Daruich, G Daubard, G Daues, W Dawson, F Delgado, A Dellapenna, R De Peyster, M De Val-Borro, SW Digel, P Doherty, R Dubois, GP Dubois-Felsmann, J Durech, F Economou, T Eifler, M Eracleous, BL Emmons, AF Neto, H Ferguson, E Figueroa, M Fisher-Levine, W Focke, MD Foss, J Frank, MD Freemon, E Gangler, E Gawiser, JC Geary, P Gee, M Geha, CJB Gessner, RR Gibson, DK Gilmore, T Glanzman, W Glick, T Goldina, DA Goldstein, I Goodenow, ML Graham, WJ Gressler, P Gris, LP Guy, A Guyonnet, G Haller, R Harris, PA Hascall, J Haupt, F Hernandez, S Herrmann, E Hileman, J Hoblitt, JA Hodgson, C Hogan, JD Howard, D Huang, ME Huffer, P Ingraham, WR Innes, SH Jacoby, B Jain, F Jammes, MJ Jee, T Jenness, G Jernigan, D Jevremovic, K Johns, AS Johnson, MWG Johnson, RL Jones, C Juramy-Gilles, M Juric, JS Kalirai, NJ Kallivayalil, B Kalmbach, JP Kantor, P Karst, MM Kasliwal, H Kelly, R Kessler, V Kinnison, D Kirkby, L Knox, IV Kotov, VL Krabbendam, KS Krughoff, P Kubanek, J Kuczewski, S Kulkarni, J Ku, NR Kurita, CS Lage, R Lambert, T Lange, JB Langton, L Le Guillou, D Levine, M Liang, K-T Lim, CJ Lintott, KE Long, M Lopez, PJ Lotz, RH Lupton, NB Lust, LA Macarthur, A Mahabal, R Mandelbaum, TW Markiewicz, DS Marsh, PJ Marshall, S Marshall, M May, R McKercher, Michellemcqueen, J Meyers, M Migliore, M Miller, DJ Mills, C Miraval, J Moeyens, FE Moolekamp, DG Monet, M Moniez, S Monkewitz, C Montgomery, CB Morrison, F Mueller, GP Muller, FM Arancibia, DR Neill, SP Newbry, J-Y Nief, A Nomerotski, M Nordby, P O'Connor, J Oliver, SS Olivier, K Olsen, W O'Mullane, S Ortiz, S Osier, RE Owen, R Pain, PE Palecek, JK Parejko, JB Parsons, NM Pease, JM Peterson, JR Peterson, DL Petravick, MEL Petrick, CE Petry, F Pierfederici, S Pietrowicz, R Pike, PA Pinto, R Plante, S Plate, JP Plutchak, PA Price, M Prouza, V Radeka, J Rajagopal, AP Rasmussen, N Regnault, KA Reil, DJ Reiss, MA Reuter, ST Ridgway, VJ Riot, S Ritz, S Robinson, W Roby, A Roodman, W Rosing, C Roucelle, MR Rumore, S Russo, A Saha, B Sassolas, TL Schalk, P Schellart, RH Schindler, S Schmidt, DP Schneider, MD Schneider, W Schoening, G Schumacher, ME Schwamb, J Sebag, B Selvy, GH Sembroski, LG Seppala, A Serio, E Serrano, RA Shaw, I Shipsey, J Sick, N Silvestri, T Colin, JA Smith, RC Smith, Shahram, C Soldahl, L Storrie-Lombardi, E Stover, MA Strauss, RA Street, CW Stubbs, IS Sullivan, D Sweeney, JD Swinbank, A Szalay, P Takacs, SA Tether, JJ Thaler, JG Thayer, S Thomas, AJ Thornton, V Thukral, J Tice, DE Trilling, M Turri, R Van Berg, D Berk, K Vetter, F Virieux, T Vucina, W Wahl, L Walkowicz, B Walsh, CW Walter, DL Wang, S-Y Wang, M Warner, O Wiecha, B Willman, SE Winters, D Wittman, SC Wolff, WM Wood-Vasey, X Wu, B Xin, P Yoachim, and H Zhan. Lsst: From science drivers to reference design and anticipated data products. *Astrophysical Journal*, 873(2), 2019.

[11] S. Koposov and O. Bartunov. Q3C, Quad Tree Cube – The new Sky-indexing Concept for Huge Astronomical Catalogues and its Realization for Main Astronomical Queries (Cone Search and Xmatch) in Open Source Database PostgreSQL. In C. Gabriel, C. Arviset, D. Ponz, and S. Enrique, editors, *Astronomical Data Analysis Software and Systems XV*, volume 351 of *Astronomical Society of the Pacific Conference Series*, page 735,

July 2006.

[12] Amrith Kumar and Kenneth Rugg. Brewers conjecture and a characterization of the limits, and relationships between consistency, availability and partition tolerance in a distributed service. *CoRR*, abs/1904.12636, 2019.

[13] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, apr 2010.

[14] Federico Marocco, Peter R. M. Eisenhardt, John W. Fowler, J. Davy Kirkpatrick, Aaron M. Meisner, Edward F. Schlafly, S. A. Stanford, Nelson Garcia, Dan Caselden, Michael C. Cushing, Roc M. Cutri, Jacqueline K. Faherty, Christopher R. Gelino, Anthony H. Gonzalez, Thomas H. Jarrett, Renata Koontz, Amanda Mainzer, Elijah J. Marchese, Bahram Mobasher, David J. Schlegel, Daniel Stern, Harry I. Teplitz, and Edward L. Wright. The CatWISE2020 catalog. *The Astrophysical Journal Supplement Series*, 253(1):8, feb 2021.

[15] Federico Marocco, Peter RM Eisenhardt, John W Fowler, J Davy Kirkpatrick, Aaron M Meisner, Edward F Schlafly, S Adam Stanford, Nelson Garcia, Dan Caselden, and Michael C et al. Cushing. The CatWISE2020 Catalog. *The Astrophysical Journal Supplement Series*, 253(1):8, 2021.

[16] Peter B. Stetson. On the automatic determination of light-curve parameters for cepheid variables. *Publications of the Astronomical Society of the Pacific*, 108:851 – 851, 1996.

[17] Alexander S. Szalay, Jim Gray, George Fekete, Peter Z. Kunszt, Peter Kukol, and Ani Thakar. Indexing the sphere with the hierarchical triangular mesh. *CoRR*, abs/cs/0701164, 2007.

[18] Yanxia Zhang and Yongheng Zhao. Astronomy in the big data era. *Data Science Journal*, May 2015.

ANEXOS

Anexo

50 mil posiciones aleatorias

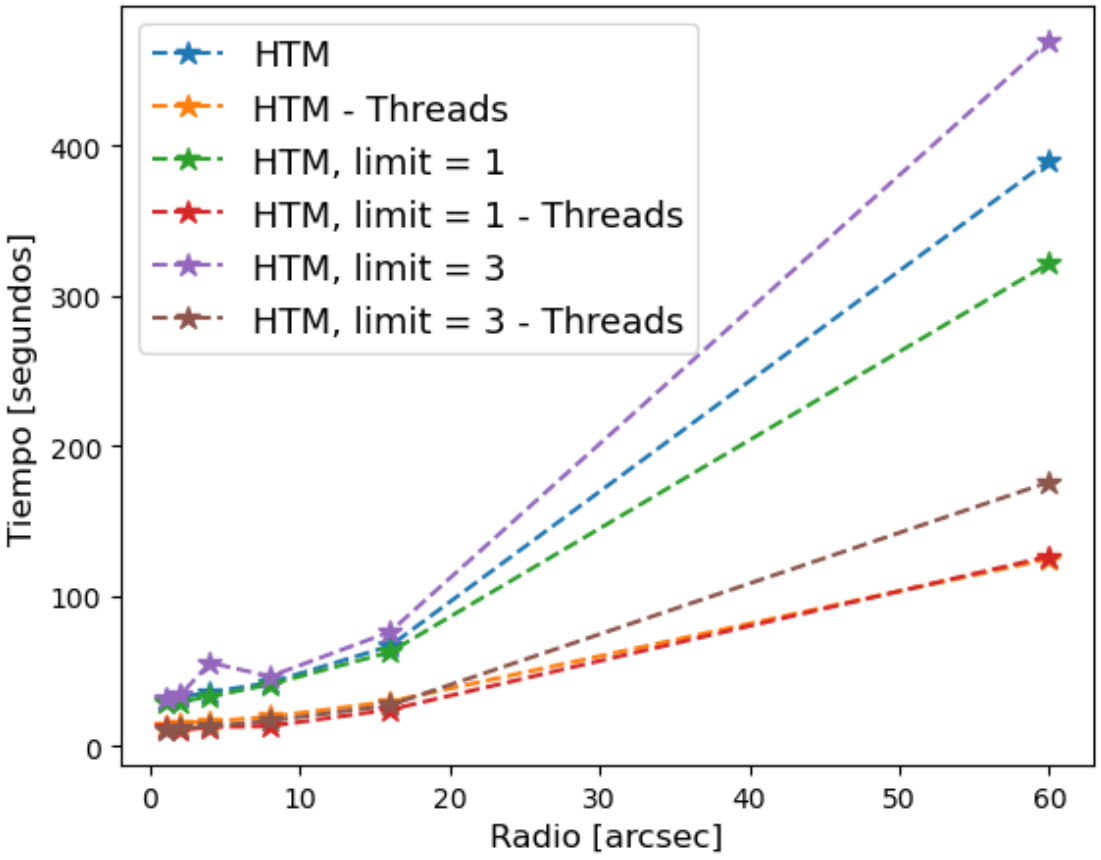

Figura A.1: Plot comparación tiempos de ejecución crossmatch con el conjunto de 50 mil posiciones aleatorias, utilizando HTM variando limit y usando threads.

El comportamiento observado al trabajar con HTM y HEALPix variando los limits y utilizando o no utilizando threads es similar. Estos resultados se presentan en la figura A.1, y se puede contrastar con los resultados obtenidos en la figura 6.5, la cuál utiliza HEALPix, observando la misma tendencia de que utilizar threads es más rápido independiente del valor

de limit. Además, se mantiene la ventaja de utilizar limit = 1 como la configuración más rápida, y limit = 3 la más lenta.

Posiciones en el plano perpendicular al plano galáctico

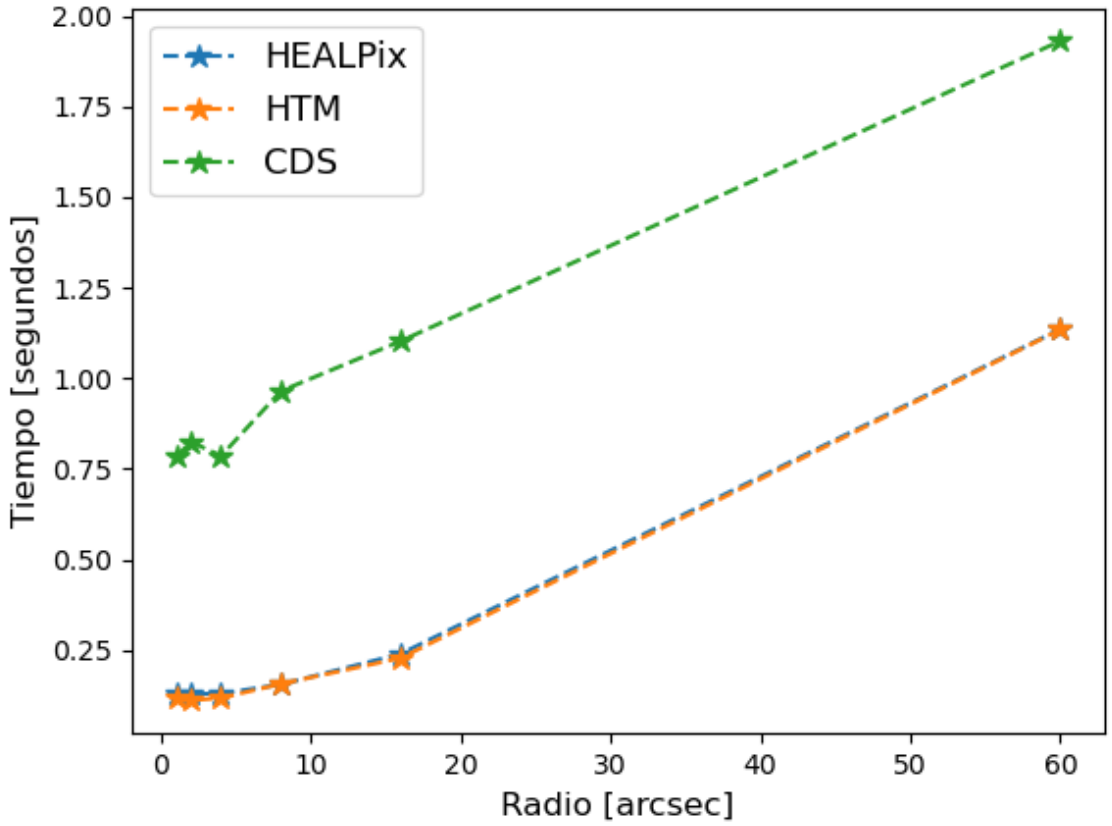

Figura A.2: Plot comparación tiempos de ejecución crossmatch con el conjunto de posiciones en el plano perpendicular al galáctico, limit None.

En el caso del conjunto de posiciones en el plano perpendicular al plano galáctico, que consta de solo 180 coordenadas, se observa un comportamiento similar al caso del plano galáctico con hilos y limit None bajo los mismos parámetros. Según la figura A.2, los tiempos de HTM y HEALPix se mantienen siempre por debajo de los tiempos de CDS, siendo aproximadamente 6 veces más rápidos en el caso de 1 arco segundo.

Este comportamiento se mantiene al utilizar hilos, aunque nuevamente se observa un aumento en el tiempo de procesamiento para los radios más pequeños. A pesar de este aumento, el sistema propio sigue siendo considerablemente más rápido que CDS en estas circunstancias.

Los resultados obtenidos demuestran que el sistema desarrollado durante este trabajo es efectivo y eficiente para procesar conjuntos pequeños de coordenadas con pocos matches. Se

logró superar el rendimiento de CDS en términos de velocidad en estos casos. Sin embargo, para conjuntos más grandes con más objetos resultantes, especialmente con radios más grandes, CDS demostró ser más eficiente en términos de tiempo de procesamiento.

Coordenadas de objetos aleatorios cerca del ecuador

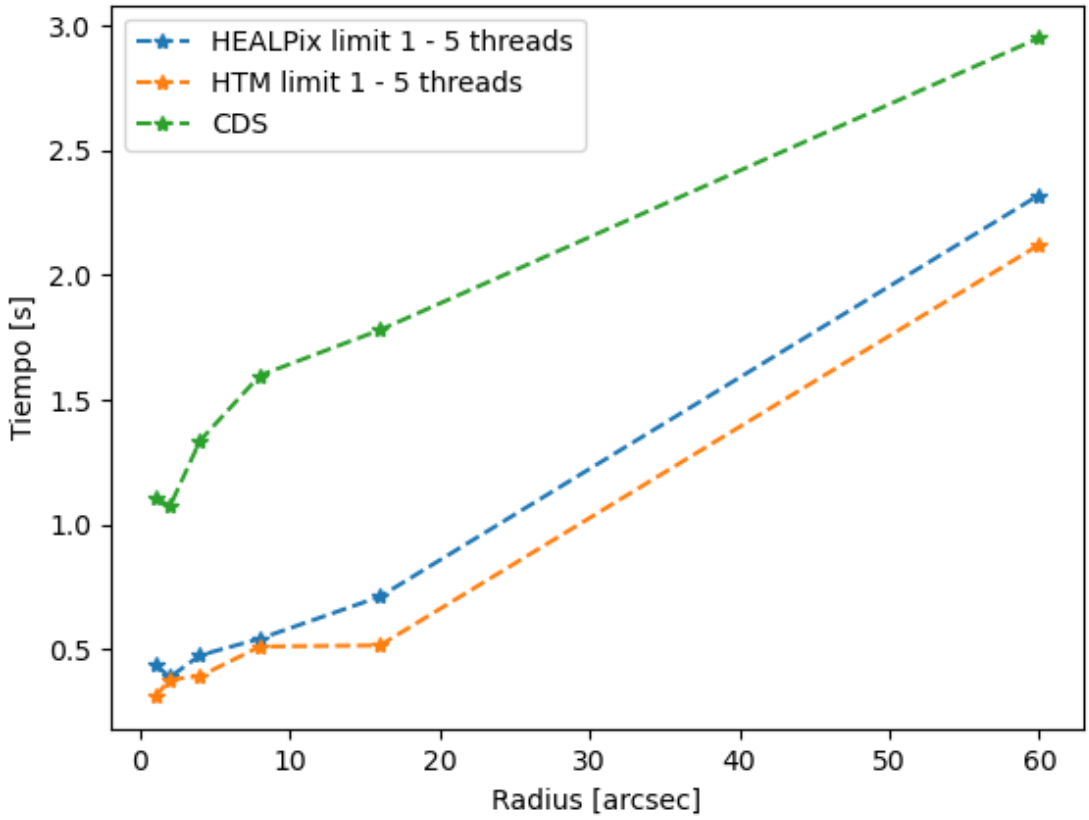

Figura A.3: Plot comparación tiempos de ejecución crossmatch con el conjunto de posiciones cercanas al ecuador, limit = 1 usando threads.

Los resultados para el sistema desarrollado en el caso de utilizar limit = 1 corresponden a la mejor configuración posible sin implementar threads. Utilizando threads es posible lograr una mejora considerable en los radios más altos, no obstante, los tiempos en los menores suele disminuir un poco, dado el tiempo de overhead necesario para construir e iniciar los hilos, además de la posterior unión de los resultados, lo que se puede observar en la figura A.3. Contrastando este resultado con los de la imagen derecha de la figura 6.8, la cuál utiliza el mismo conjunto de datos e implementa hilos, pero con limit = None, es posible notar que se alcanzan tiempos similares, mejorando en particular en el caso de 1 arcominuto.

Coordenadas de objetos aleatorios cerca de los polos

El último conjunto de pruebas tiene la misma cantidad de objetos que el conjunto de puntos cerca del ecuador, cuyos comportamientos se encuentran graficados en la figura 6.8, por lo que se espera que se obtenga un comportamiento similar para los tiempos de ejecución de las consultas. El resultado es análogo, incluso si se compara el caso donde se realizan las requests para HTM y HEALPix utilizando threads, como se puede observar en la figura A.4. No obstante, en este último caso se tiene que si bien HEALPix se mantiene levemente mejor que CDS para el radio de 1 arco minuto, alcanzando un tiempo promedio de 2,738 frente a 2,958 segundos alcanzado por CDS, HTM tiene un tiempo levemente mayor, logrando un promedio de 3,093 segundos, aunque presentando una desviación estándar de 0,489 segundos en esta última medición.

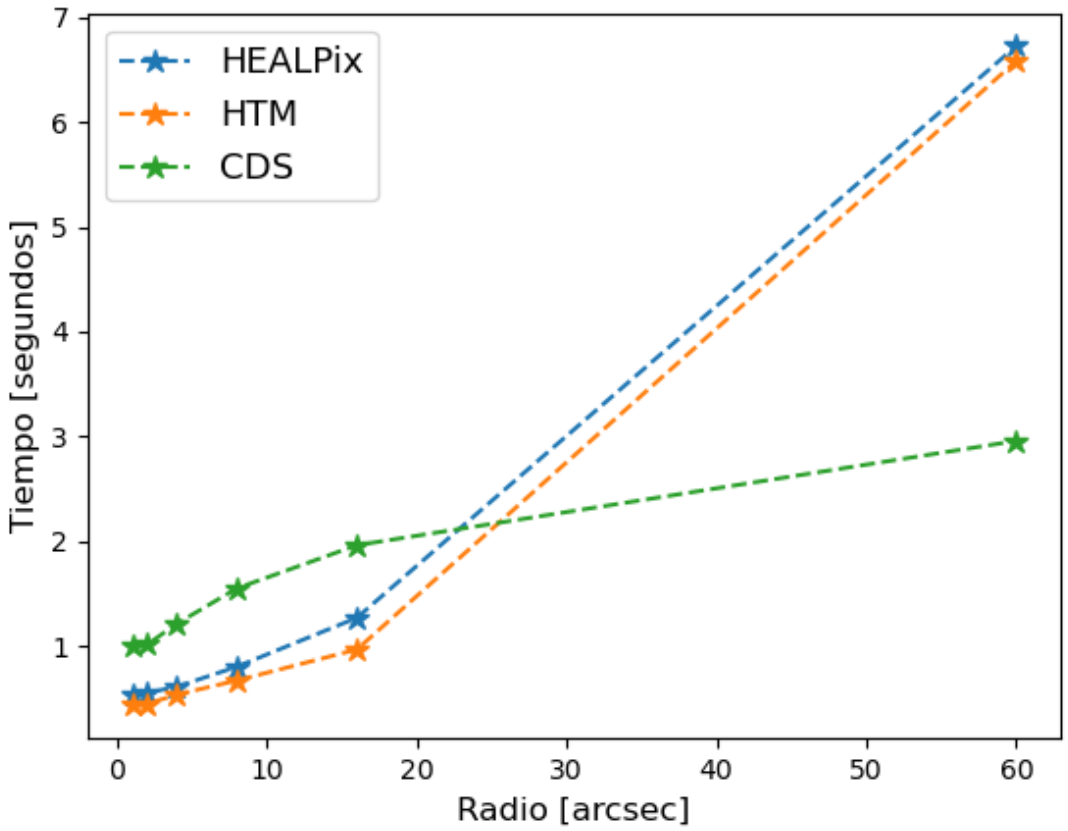

Figura A.4: Plot comparación tiempos de ejecución crossmatch con el conjunto de posiciones cercanas a los polos, limit None.