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

SERVICIO DE CROSSMATCHING DE OBJETOS ASTRONÓMICOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

ALEJANDRA VERÓNICA ALARCÓN VALENZUELA

PROFESOR GUÍA:
AIDAN HOGAN

PROFESOR CO-GUÍA:
FRANCISCO FÖRSTER BURÓN

MIEMBROS DE LA COMISIÓN:
EDUARDO GODOY VEGA
FELIPE BRAVO MARQUEZ

SANTIAGO DE CHILE
2022

Resumen

ALeRCE es un agente de alertas astronómicas chileno que trabaja con alertas provenientes del *Twicky Transient Facility* y próximamente del *Legacy Survey of Space and Time* del observatorio Vera C. Rubin. El objetivo principal de ALeRCE es el estudio de tres categorías amplias de objetos: transitorios, estrellas variables y núcleos galácticos activos. Para lo anterior, se utiliza una serie de procesos que incluyen la ingestión, agregación, clasificación y visualización de datos.

Para filtrar e identificar los datos provenientes de las alertas, se realiza el *crossmatching* entre un catálogo astronómico y las posiciones de los objetos entrantes. El *crossmatching* consiste en encontrar un objeto en dos o más catálogos cruzando los datos del catálogo con las del objeto en cuestión. ALeRCE utiliza un servicio externo de *crossmatching*, CDS X-Match, por lo que no tiene control sobre los catálogos disponibles ni la disponibilidad del sistema, cosa que puede afectar el rendimiento del agente de alertas.

Debido a lo anterior es que en este trabajo se propone implementar un servicio compuesto por una interfaz de programación de aplicaciones (en inglés *application programming interface*, o API) que se conecta a una base de datos en MongoDB que contiene el catálogo CatWISE2020. A través de la API se pueden realizar peticiones HTTP para obtener el *crossmatch* entre una lista de ubicaciones y el catálogo en el sistema.

Según los experimentos realizados para comprobar la correctitud, el sistema desarrollado tiene alrededor de un porcentaje de diferencia en el número de objetos entregado de un 1 % en promedio, así como también hay una diferencia de 0,25 segundos de arco en las distancias encontradas para los objetos cuando se trata de radios cercanos a 16 segundos de arco. Con respecto a la velocidad, el sistema implementado es más eficiente para muestras pequeñas de datos en radios menores a 8 segundos de arco.

En general el trabajo genera un sistema competitivo en comparación a CDS X-Match pero existen características mejorables, como una optimización en el tiempo de respuesta de la API, que se pueden implementar en el futuro.

Apathy is death. Worse than death,
because at least a rotting corpse feeds the beasts and insects.

—Darth Traya

Agradecimientos

A mi papá Hernán por su apoyo constante y anécdotas varias sobre la vida en Beauchef en tiempos pasados, a mi mamá Ana por su cariño y retos hasta sus últimos días, a mi hermano Andrés por los memes, las risas y las salidas a comer. A mi abuela Josefina, mi tía Luisa y mi prima María José por los años que estuve de allegada en la casa y las conversaciones ocasionales.

A Aidan y Francisco por su mentoría, por sus consejos sobre cosas técnicas y otras no tan técnicas, por responder a todas las dudas que tenía, por las conversaciones que a veces se daban y por confiarme este tema a pesar de que no me creía capaz de llevarlo a cabo. También a Alberto por ayudarme con la configuración de AWS y ayudarme a empezar a programar al inicio de este trabajo.

Al Chanta Team/Empresas Mangulas por los años de amistad, las conversaciones ñoñas, las risas y las partidas de rol, a pesar de la distancia física y mis desapariciones por meses hasta que me dignaba a aparecer.

Al Dani, el Alexis y Jorge por ser mis primeros amigos y apoyo en el comienzo de mi vida universitaria y hasta estos días. También a Pedro por todas esas salidas, juntas en su departamento y por contratarme como encargada de relaciones públicas a pesar de que no me pagaba (?).

A los grupos *Anime no Seishin Doukoukai*, el *Club de Rol de Injeniería* y *La Radio Integral* por haber sido mis espacios de distensión durante todos estos años y haber aportado algo más a mi formación como profesional y como persona, así como también por haberme permitido conocer grandes personas que me acompañarían en este camino. Una mención especial para el Lulo y Ñanku por su amistad, el bullying y los años que fuimos los encargados de las tardes de karaoke.

Al DCC y su comunidad por ser un lugar ameno y seguro, como un oasis después de la hostilidad de plan común (igual me eché ramos, pero esa es otra historia).

Mención mucho muy especial al Andy, Lecaros, la Fabi y el Terry por los memes, ser patitos debugger y aguantar mis momentos de demencia que me daban ocasionalmente.

En fin, a todas las personas que me han acompañado de alguna forma durante mis años como estudiante.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación

El primer catálogo de estrellas fue creado por Hiparco de Nicea en la Antigua Grecia. Los primeros catálogos modernos, que fueron escritos en papel durante los siglos XVIII y XIX, fueron los cimientos de la astronomía moderna al recolectar datos de la posición y el brillo de cientos de miles de estrellas.

En la década de 1990, James Gunn junto a otros científicos diseñaron el *Sloan Digital Sky Survey* (SDSS), un proyecto de observación del espacio que fue llevado a cabo en el observatorio Apache Point en Nuevo México, Estados Unidos. Este proyecto marcó un hito importante en la astronomía y la llevó al campo del *big data* debido a su capacidad de sondear un tercio de la esfera celeste, del cual ha obtenido información de miles de millones de objetos como estrellas, galaxias y agujeros negros supermasivos en lugares distantes del espacio. Esto se traduce en que el SDSS recolecta cerca de 200 GB de datos por noche que son agregados a una base de datos que contiene aproximadamente 50 TB [4].

Así es como surgió la astroinformática: un campo interdisciplinario de estudio que combina la astronomía con la ciencia de datos, el *big data* y el *machine learning*. En la astroinformática existen 4 desafíos relevantes que se abordan usando procesamiento masivo de datos. Estos desafíos son:

1. Volumen: Se generan muchos datos por noche en varios observatorios distribuidos alrededor del mundo.
2. Velocidad: Existen ocasiones donde se deben tomar decisiones en tiempo real, por lo que existe el desafío de procesar un alto flujo de alertas de los telescopios.
3. Variedad: Existen numerosos catálogos de objetos astronómicos que se alimentan de distintos proyectos de observatorios, y para un mismo objeto pueden existir varios registros de este a lo largo del tiempo.
4. Veracidad: Al realizar búsquedas de objetos en los catálogos astronómicos, se debe asegurar que la información relativa a este efectivamente sea del elemento y no haya

sido clasificado erróneamente.

Por lo anterior es que ha surgido la necesidad de crear agentes de alertas astronómicas. Además de esto, es conocido que Chile es una potencia mundial en cuanto al estudio de la astronomía debido a que el desierto de Atacama provee un ambiente idóneo para la observación.

Así es como surge el *Automatic Learning for the Rapid Classification of Events* (ALeRCE), que es un agente de alertas astronómicas chileno diseñado para proveer una clasificación de largos flujos de alertas de telescopios, tales como los proveídos por el *Zwicky Transient Facility* (ZTF) y, en el futuro, el *Legacy Survey of Space and Time* (LSST) del observatorio Vera C. Rubin en el norte de Chile.

Dentro de la línea de procesos que realiza ALeRCE se encuentra el *crossmatching* de cuerpos astronómicos, el cual consiste en encontrar pares de coincidencias entre dos (o más) catálogos correspondientes a un mismo objeto[1]. Para esto se realiza una búsqueda proyectando un cono con centro en la posición de cada punto y se localizan todos los puntos en el radio cercano, este proceso se le denomina *conesearch*.

Actualmente, uno de los problemas que enfrenta ALeRCE es el uso de servicios externos como parte de su cadena de procesamiento de los datos, lo cual puede presentar inconvenientes como la interrupción del servicio externo y la limitación de los datos disponibles.

1.2. Objetivos

El objetivo general de este trabajo es reemplazar el servicio utilizado actualmente por ALeRCE, CDS X-Match, con uno local que sea capaz de procesar y almacenar alrededor de 10 millones de alertas por noche, lo que correspondería a aproximadamente 350 alertas por segundo, que es el flujo esperado de datos cuando el LSST entre en operación; asimismo, este nuevo sistema debe tener una alta disponibilidad. También debe ser capaz de realizar *crossmatching* de las alertas entrantes sobre varios catálogos.

Para alcanzar el objetivo general se han propuesto los siguientes objetivos específicos:

1. Entender la estructura de los catálogos astronómicos
2. Revisar los posibles catálogos para ser utilizados en el trabajo.
3. Comparar y escoger un método de indexación espacial para realizar búsquedas eficientes.
4. Diseñar el modelo de datos para los objetos astronómicos.
5. Habilitar *endpoints* para acceder al nuevo servicio.
6. Realizar pruebas de rendimiento y correctitud de datos en el sistema.

[1] `https://ipg.fer.hr/ipg/resources/astronomical_cross-matching`

1.3. Estructura del documento

El presente documento se compone de 7 capítulos, incluyendo la introducción de éste. En el capítulo 2 se detallan los antecedentes, conceptos y tecnologías relevantes para el entendimiento de este trabajo. En el capítulo 3 se resume el diseño e implementación del servicio desarrollado como solución. En el capítulo 4 se detalla el pre procesamiento de los datos necesarios. En el capítulo 5 se explica la implementación de la interfaz desarrollada. En el capítulo 6 se indican los experimentos realizados junto con sus resultados. Por último, en el capítulo 7 se da conclusión a esta memoria señalando el cumplimiento de objetivos y posible trabajo futuro.

Capítulo 2

Estado del Arte

En este capítulo se detallan tópicos importantes para el entendimiento del problema y la solución a desarrollar en este trabajo.

2.1. Crossmatching

El *crossmatching* dentro del contexto de la astroinformática consiste en encontrar un objeto en dos o más catálogos, para lo cual cruza los datos de ambas listas según las coordenadas de la ubicación del cuerpo (con un margen de error arbitrario), lo cual es importante para llevar un registro de los objetos y analizar sus posibles variaciones en el tiempo.

Para realizar el cruce de datos, una de las técnicas más utilizadas se llama *conesearch*. Este consiste en formar la proyección de un cono en un plano, a partir de un punto y un radio [10]. Como se trabaja con objetos astronómicos, el punto se define a partir de sus coordenadas (la ascensión recta y la declinación) y el radio se define por una distancia arbitraria medida en segundos de arco.

Para realizar una búsqueda de manera eficiente se utilizan índices espaciales, los cuales se diferencian de índices normales por utilizar estructuras de datos con búsquedas eficientes como *quadtrees*, *k-d trees*, *R-tree*, entre otras.

Aquí se presentarán tres índices espaciales que trabajan sobre coordenadas esféricas, las cuales son ideales debido a que las coordenadas ecuatoriales (que son las utilizadas usualmente en astronomía) se modelan sobre una esfera. Este sistema de coordenadas incluye dos valores: la ascensión recta (RA) y la declinación (DEC); estos son equivalentes a la latitud y longitud geográficas respectivamente.

Primero, se encuentra la malla triangular jerárquica (*Hierarchical Triangular Mesh*, o HTM), que consiste en una descomposición recursiva de una esfera a partir de un octaedro para modelar una esfera y definir áreas de píxeles que representan coordenadas en el espacio. Para la descomposición, se toma una cara del poliedro (que es un triángulo) y se trazan líneas entre los puntos medios de cada arista de la cara, dividiendo al triángulo en 4 más pequeños;

y esa operación se repite en cada cara del octaedro.

A cada triángulo que conforma el octaedro se le denomina *trixel* de nivel 0, y cada uno puede ser descompuesto en 4 *trixels* más pequeños introduciendo nuevos vértices en el punto medio de cada lado. En la figura 2.1 primero se presentan tres cuerpos con *trixels* de nivel 0, 1 y 2 en la fila superior, mientras que en la fila inferior los niveles son más altos y se asemejan más a una esfera.

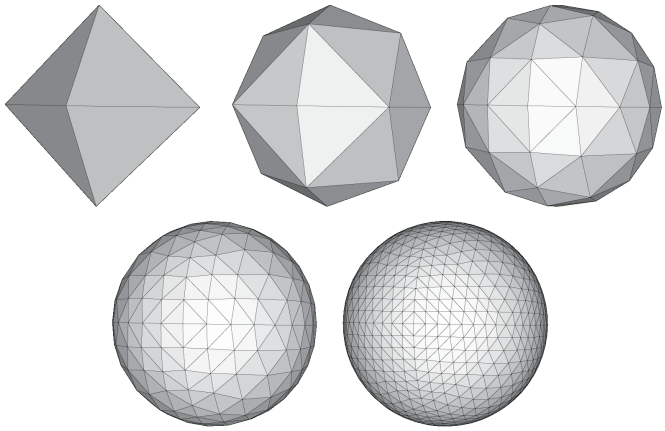


Figura 2.1: Representación visual de la descomposición en HTM [13].

Sin embargo, pueden haber inconvenientes con usar HTM, como el hecho de que solo tiene implementaciones para sistemas de código cerrado como Microsoft SQL Server y MySQL, además de problemas de rendimiento cuando la segmentación es muy alta; por lo que surgió Quad Tree Cube (Q3C) como una opción open-source que puede ser utilizada en PostgreSQL. En Q3C se usa la idea de inscribir un cubo en una esfera y crear un *quadtree* en cada cara de dicho cubo, de tal forma se crea una descomposición recursiva en la superficie de la esfera.

En la figura 2.2 se observa una esfera con las divisiones representadas en la frase anterior: se toma un cubo y se "infla" hasta lograr una esfera, y cada cara del cubo se segmenta con líneas paralelas a las aristas de cada cara.

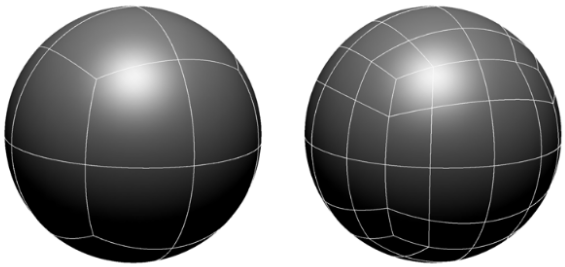


Figura 2.2: Representación visual de la segmentación en Q3C [8].

Una cosa que tienen en común HTM y Q3C es que la esfera no se divide necesariamente en áreas iguales. Para poder realizar una división donde cada segmento represente un área de igual magnitud existe HEALPix (acrónimo para *Hierarchical Equal Area iso-Latitude Pixelation* [7]).

HEALPix realiza divisiones con cuadriláteros curvilíneos sobre una esfera que se pueden seguir dividiendo recursivamente en 4 secciones. Estas divisiones se pueden ver representadas en la figura 2.3. Los pixeles están distribuidos en líneas de latitud constante, es decir que estas líneas son paralelas entre sí. Este método de indexación espacial está disponible para ser utilizada con distintos lenguajes de programación como C, C++, FORTRAN90, Java y Python.

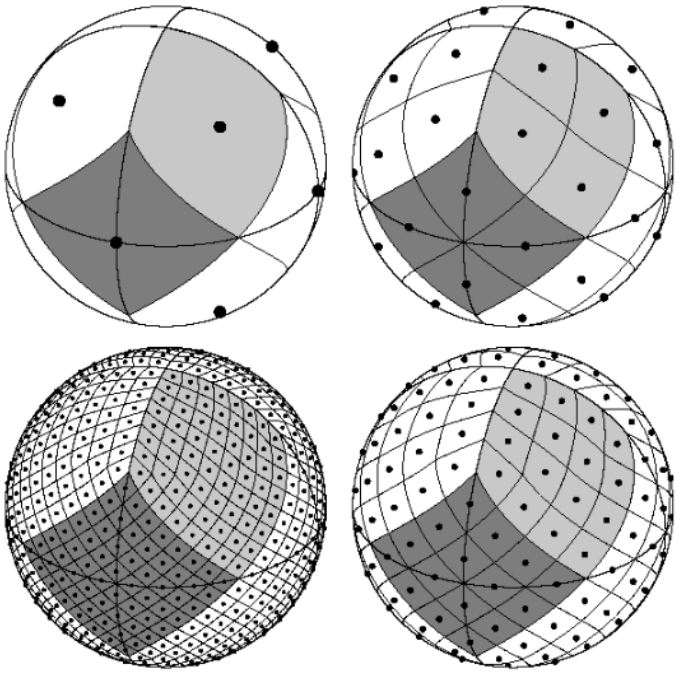


Figura 2.3: Representación de la segmentación en HEALPix [7]

2.2. Catálogos Astronómicos

Un catálogo astronómico consiste en una lista o tabulación de objetos astronómicos que son agrupados típicamente por compartir un tipo en común, morfología, origen, medio de detección o método de descubrimiento. Los catálogos más comunes son los de estrellas, pero también existen de otros tipos de objetos como nebulosas, galaxias, cuasares, cúmulos de estrellas, entre otros.

En la antigüedad, los catálogos se confeccionaban a mano en papel, pero ahora se componen por bases de datos compartidas en internet a través de aplicaciones web. Actualmente los catálogos más populares son aquellos generados a partir de sondeos astronómicas.

Un sondeo astronómico (o *astronomical survey* en inglés) es un mapa general del cielo, o de un sector de éste, de propósito general. Generalmente cada sondeo está restringido a una banda de frecuencia específica (ya sea infrarrojo, ultravioleta, rayos X, entre otros).

Algunos ejemplos de relevancia para este trabajo son:

- Zwicky Transient Facility (ZTF) [1]: Es un sondeo tomado por una cámara del telescopio Samuel Oschin del observatorio Palomar en Estados Unidos. Se encarga de observar fenómenos de corta duración que ocurren en días o hasta años.

  Los fenómenos u objetos detectados pueden ser novas, supernovas o tránsitos de asteroides y cometas delante de estrellas.

  En su última versión *Data Release 8* tiene más de 6 mil millones de objetos.

- Wide-field Infrared Survey Explorer (WISE)[1]: Es un telescopio que pertenece a la Administración Nacional de Aeronáutica y el Espacio de Estados Unidos (NASA).

  Entre 2009 y 2011 realizó un sondeo de objetos en el espectro del infrarrojo, el cual volvió a entrar en 2013 bajo el nombre de NEOWISE (por *Near-Earth Objects*) con la misión de identificar objetos cercanos a la Tierra, así como también objetos más distantes tales como asteroides y cometas.

  Actualmente existe un catálogo llamado AllWISE que contiene los registros de WISE y NEOWISE. Este contiene alrededor de 750 millones de objetos.

- Legacy Survey of Space and Time (LSST)[2]: Es un sondeo que entrará en funcionamiento a través del observatorio Vera C. Rubin, en el norte de Chile. Se estima que generará 500 Petabytes de imágenes y datos del cielo y estará en funcionamiento por 10 años.

2.3. ALeRCE

Una alerta astronómica es un paquete de datos que reporta un cambio en el cielo, observado por un telescopio durante un sondeo. Todas las cosas que cambien en la zona observada gatillan una alerta que contiene información sobre lo que ocurre ahí.

Automatic Learning for the Rapid Classification of Events (ALeRCE) es un agente de alertas chileno diseñado para proveer una clasificación de largos flujos de alertas de telescopios, tales como los proveídos por el *Zwicky Transient Facility* (ZTF) y, en el futuro, el *Legacy Survey of Space and Time* (LSST) del observatorio Vera C. Rubin en el norte de Chile [5].

El objetivo principal de ALeRCE es el estudio de tres categorías amplias de objetos: transitorios, estrellas variables y núcleos galácticos activos; también provee clasificación de objetos del sistema solar como un objetivo científico secundario.

El agente de alertas ha procesado 97 millones de alertas provenientes del *stream* público del ZTF a un ritmo de 300 mil alertas por noche, lo que equivale a procesar 11 por segundo asumiendo que una noche dura 8 horas. Cada alerta es registrada según un esquema de Apache Avro[3] que sigue lo descrito en la tabla 2.1.

ALeRCE usa una serie de procesos (graficados en la figura 2.4[4]) que incluye ingestión en tiempo real y operaciones de agregación, *crossmatching*, clasificación con *machine learning*

[1] `https://www.nasa.gov/mission_pages/WISE/mission/index.html`
[2] `https://www.lsst.org/about`
[3] `https://avro.apache.org/`
[4] `http://alerce.science/alerce-pipeline/`

Tabla 2.1: Campos de una alerta astronómica según ztf-avro-alert

Campo	Tipo	Descripción
schemavsn	string	versión del esquema usado
publisher	string	origen del paquete de alerta
objectId	long	identificador único para el objeto
candid	long	identificador único para el subtraction candidate
candidate	ztf.alert.candidate	registro del candidato
prv_candidates	lista de ztf.alert.prv_candidate or null	historial de candidatos de los últimos 30 días
cutoutScience	ztf.alert.cutout or null	recorte de "science image"
cutoutTemplate	ztf.alert.cutout or null	recorte de la referencia "coadded"
cutoutDifference	ztf.alert.cutout or null	recorte de la imagen resultante de la diferencia

y visualización de los flujos de alertas del ZTF. Mientras que para el almacenamiento, se guardan las alertas en un contenedor de Amazon Simple Storage Service (S3) y en una base de datos local. Para este trabajo se hará énfasis en la etapa de *crossmatching*.

2.4. CDS X-Match

Actualmente ALeRCE depende de un servicio externo, llamado CDS X-Match, que es dependiente del Centro de datos astronómicos de Estrasburgo (*Centre de Données astronomiques de Strasbourg*, o CDS), para el proceso de *crossmatch*. Este servicio permite que los usuarios puedan utilizar algoritmos para cruzar datos astronómicos de manera eficiente con catálogos, o listas de objetos, muy grandes (con cerca de mil millones de filas) a través de una interfaz web. Este servicio dispone de 3 sub servicios:

- VizieR[5]: Servicio de catálogos de objetos astronómicos obtenidos de distintos *surveys*. El agente de alertas trabaja específicamente con WISE.
- SIMBAD[6]: Una base de datos que provee datos básicos, identificaciones cruzadas, bibliografía y medidas para objetos astronómicos.
- Datos subidos por los usuarios[7]: A través de la interfaz web de X-Match se pueden subir tablas como VOTable, FITS o CSV.

CDS tiene disponible implementaciones para HEALPix y un buscador de catálogos para VizieR en su cuenta de Github `https://github.com/cds-astro`. Sin embargo, no se encontró un algoritmo de *crossmatching* ya que HEALPix lo utiliza para consultas por secciones y no para el cielo completo, según lo descrito en la documentación del servicio[8].

El problema de utilizar CDS X-Match está relacionado con tres factores principales:

- Restricción de tiempo: El sistema arroja un error si la operación excede un tiempo de 7.200 segundos en ejecutarse.

[5] `https://vizier.inasan.ru/viz-bin/VizieR`
[6] `https://simbad.u-strasbg.fr/simbad/`
[7] `http://cdsxmatch.u-strasbg.fr/xmatch/doc/table-upload.html`
[8] `http://cdsxmatch.u-strasbg.fr/xmatch/doc/xmatch-area.html`

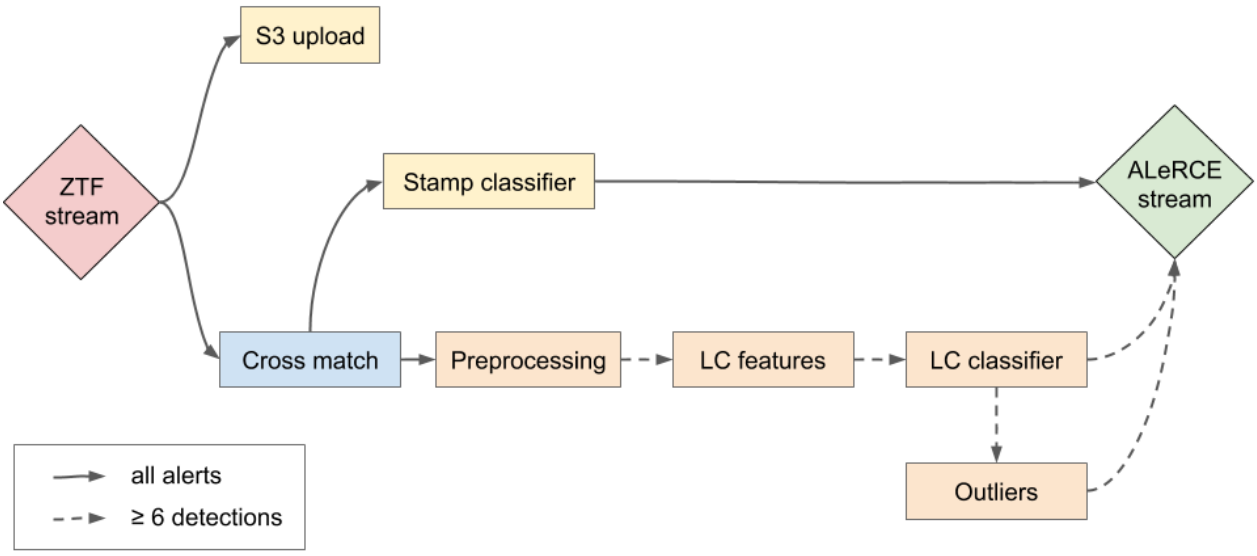


Figura 2.4: Flujo de datos utilizado por ALeRCE.

- Actualización de datos: Se desconoce la frecuencia de actualización de los datos almacenados en VizieR, el cual depende de CDS.
- Disponibilidad: al ser un servicio alojado por una organización externa a ALeRCE, no se puede manejar eficazmente si existe una caída del sistema o algún tipo de inconveniente, lo cual paraliza el funcionamiento del *pipeline*.

2.5. NoSQL

Las bases de datos NoSQL son aquellas que pueden manejar un volumen masivo de datos no estructurados que cambian de formas diferentes con respecto a una base de datos relacional (SQL).

Hay distintas formas de representar datos como pares llave-valor, documentos, columnas de tablas y grafos. Algunos ejemplos de administradores de datos NoSQL son Amazon DynamoDB, MongoDB, Apache Cassandra, Neo4j, entre otros.

A diferencia de las bases de datos relacionales, NoSQL no sigue el estándar ACID. En su lugar, cumple el teorema de Brewer [6] que también se le conoce como el teorema CAP, el cual indica que hay 3 características deseables en servicios de datos distribuidos:

- Consistencia (*Consistency*): Si se realiza una escritura en los datos, se debe garantizar que se mantienen los datos en el mismo estado para operaciones posteriores.
- Disponibilidad (*Availability*): Cada petición recibida debe tener éxito y entregar un resultado.
- Tolerancia de partición (*Partition-tolerance*): El sistema debe seguir funcionando aunque se presenten fallas en la red.

Sin embargo, ningún sistema puede satisfacer más de 2 de las 3 propiedades de forma simultánea.

Tabla 2.2: Comparación entre SQL y NoSQL

SQL	NoSQL
Modelo de datos relacional	Modelo de datos no relacional
Tiene un esquema fijo	Tiene un esquema dinámico
Escala verticalmente	Escala horizontalmente
Soporta consultas complejas	No soporta consultas complejas

Cabe destacar que, según las comparaciones en la tabla 2.2, los sistemas NoSQL tienen una mayor flexibilidad en cuanto a la forma de organización de la información, pero a su vez sacrifica la complejidad de las operaciones que se pueden realizar sobre los datos.

Por otro lado, los sistemas NoSQL escalan horizontalmente mientras que los sistemas SQL escalan verticalmente. El escalamiento horizontal implica que puede mejorar su rendimiento agregando más computadores al sistema entre los cuales se distribuye la carga de procesamiento, mientras que en el escalamiento vertical solo se puede agregar más recursos a un solo computador que realiza todas las operaciones sobre los datos[9].

2.6. Apache Spark

Apache Spark es un *framework* de computación distribuida. La principal abstracción es el *resilient distributed dataset* (RDD) que es una colección de objetos de lectura que están particionados a lo largo de un conjunto de máquinas y que pueden ser reconstruidos si una partición se pierde [14].

Spark provee muchas interfaces de alto nivel para distintos lenguajes como Scala, Java, Python y R. Entre las herramientas que ofrece se destaca Spark SQL: un módulo para el procesamiento de datos estructurados. Este módulo provee información sobre la estructura de los datos y los procesos que se están ejecutando sobre ellos, a su vez que los utiliza para realizar optimizaciones adicionales. Hay muchas formas de interactuar con Spark SQL pero para este trabajo se destacarán los *Datasets* y *DataFrames*.

Un *Dataset*[10] es una colección distribuida de datos que usa los beneficios de los RDD (como el uso de funciones lambda) en conjunto con la ejecución optimizada que tiene Spark SQL. La API de *Dataset* está disponible en Java y Scala. Sin embargo, esta API no está disponible para Python pero las ventajas de este tipo de estructura se pueden ver reflejadas en otras estructuras como los *DataFrames*.

Un *DataFrame*[10] es un *Dataset* organizado en columnas con nombres. Conceptualmente es el equivalente a una tabla en una base de datos relacional o a un *dataframe*[11] de Python/R.

[9] `https://www.techopedia.com/definition/7594/horizontal-scaling`

[10] `https://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes`

[11] `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html`

Capítulo 3

Solución

En este capítulo se explica de forma general el problema que motiva este trabajo y la solución propuesta junto con las tecnologías utilizadas.

3.1. Descripción del problema

ALeRCE actualmente utiliza una API (*Application Program Interface*, o interfaz de programación de aplicaciones) llamada CDS Xmatch. Esta API depende de un centro de investigación ubicado en Francia, por lo que no se tiene control sobre los datos almacenados y la disponibilidad de la plataforma, tal como fue mencionado en la sección 2.4.

3.2. Solución propuesta

La solución propuesta consiste en un sistema hospedado en *Amazon Web Services* a través de una instancia de EC2. En la figura 3.1 se muestra que el sistema recibe los datos de entrada a partir de las alertas provenientes de ZTF y LSST. para luego generar resultados que son utilizados por el resto del *pipeline*.

La implementación de la solución se divide en dos etapas:

- El trabajo *offline*, que consiste en preparar el ambiente de desarrollo y los datos necesarios para poder disponibilizarlos. En el capítulo 4 se habla en mayor detalle sobre este proceso.
- El trabajo *online* en el cual se dejan disponibles los datos y las consultas de *crossmatching* a través de una API. En el capítulo 5 se profundiza sobre este tema.

Las herramientas utilizadas en el trabajo fueron las siguientes:

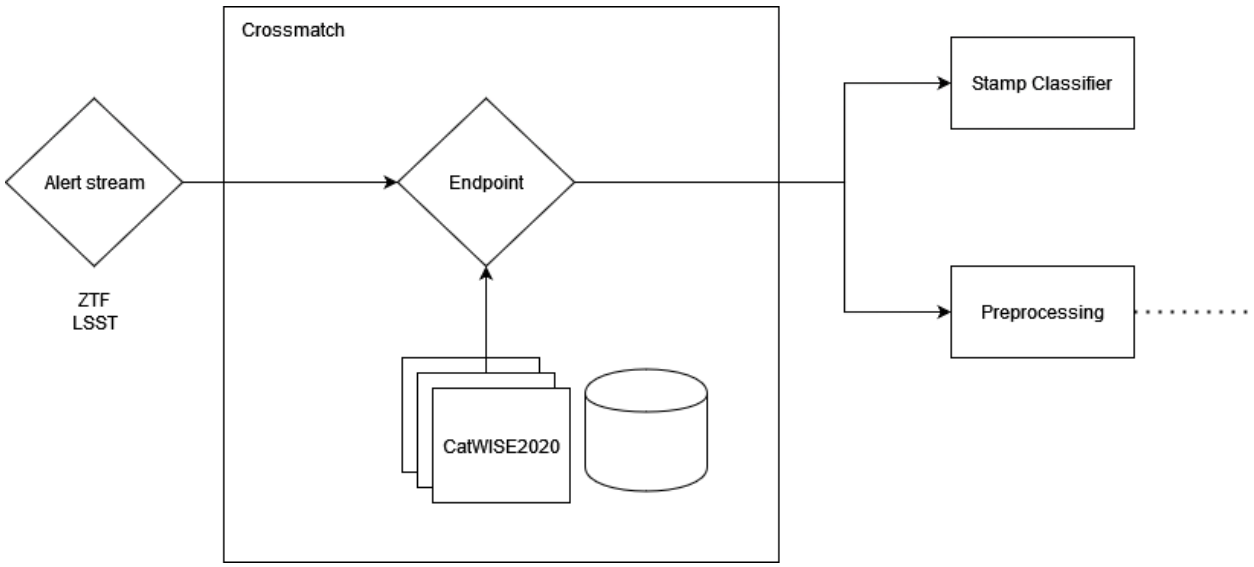


Figura 3.1: Diagrama de arquitectura del sistema propuesto unido al *pipeline* de ALeRCE.

- Apache Spark: Motor de computación distribuida. Se escogió por su capacidad de paralelizar los procesos lo cual permitió optimizar el preprocesamiento de datos, además de tener plugins para poder utilizarse en conjunto con MongoDB y Python.
- MongoDB: Motor de bases de datos distribuido orientado a documentos. Fue escogido para el trabajo debido al soporte para consultas geográficas, la escalabilidad horizontal y su popularidad, que implica que está en constante desarrollo.
- Python: Lenguaje de programación de alto nivel. Fue escogido por su facilidad para la comprensión y ejecución del código. Las siguientes librerías fueron utilizadas durante el desarrollo del trabajo:
  - PySpark: Conector de Apache Spark para Python.
  - PyMongo: Conector de MongoDB para Python.
  - Astropy: Conjunto de utilitarios de uso astronómico. Se utilizó para encontrar los puntos en el plano galáctico y en el perpendicular a este a partir del sistema galáctico de coordenadas.
  - FastAPI: *Framework* web para desarrollo de APIs REST. Se optó por este en lugar de otras librerías como Django REST por su facilidad y rapidez para implementar los endpoints.
  - Uvicorn: Una implementación de servidor ASGI[1]. La ventaja de usarlo es que permite llamadas asíncronas.
- Postman: Interfaz gráfica que permite realizar peticiones a APIs. Se utilizó para probar la API desarrollada y poder compararla con la de CDS.
- Docker: Plataforma de software que permite desplegar aplicaciones con facilidad a través de contenedores. Se utilizó para disponibilizar la API.

[1] *Asynchronous Server Gateway Interface*: `https://asgi.readthedocs.io/en/latest/`

Capítulo 4

Preprocesamiento

En este capítulo se describe con mayor detalle la primera etapa del trabajo que consiste en el preprocesamiento de los datos que componen al catálogo a utilizar en este trabajo, así como también la construcción de la base de datos que lo almacena.

4.1. Catálogo

CatWISE es un catálogo de todo el cielo que recolecta los datos de WISE y NEOWISE en las longitudes de onda de 3,4 y 4,6 μm [2]. Este catálogo tiene 2 versiones: una preliminar que usa datos entre 2010 y 2016, y CatWISE2020 que agrega dos años más de objetos observados.

CatWISE2020 contiene 2.232.515.025 objetos entre *catalog* y *reject*, cuya diferencia radica en que el primero debe cumplir las siguientes condiciones[1]:

1. Ser "primario" (es decir, estar en la sección donde esa fuente está más lejos de los bordes) en su sección de la grilla[2].
2. Tener una relación señal/ruido (o SNR, por *signal to noise ratio* en inglés) igual o mayor a 5 para W1[3] sin artefactos identificados (un valor de 0 para "ab_flags" en la parte de W1 (caracter izquierdo) de la columna 178[4]).

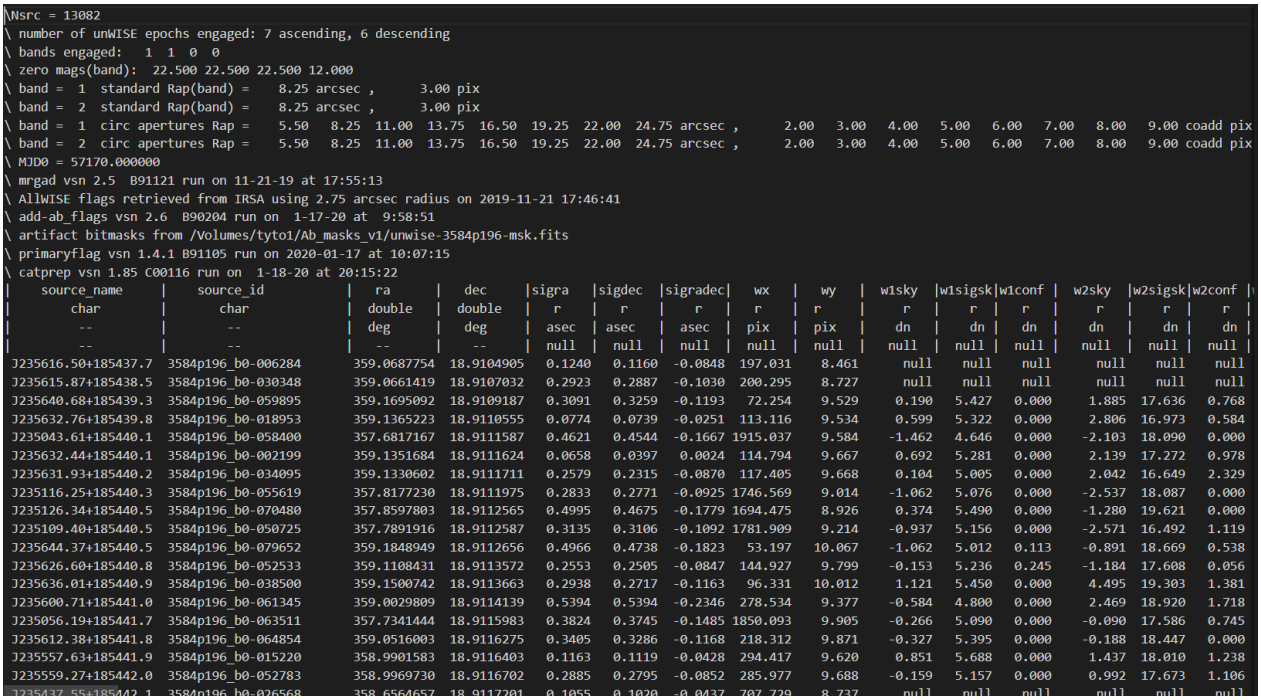
Si no ocurre lo anterior, también se puede incluir si cumple que:

1. Tener $SNR >= 5$ para W2[5] sin artefactos identificados (un valor de 0 para "ab_flags"' en la parte de W2 (caracter derecho) de la columna 178).

[1] `https://portal.nersc.gov/project/cosmo/data/CatWISE/2020README.txt`
[2] Según el sistema de coordenadas ecuatorial.
[3] La banda 1 de 3.4 μm.
[4] `https://portal.nersc.gov/project/cosmo/data/CatWISE/2020cwcat.sis20200318.txt`
[5] La banda 2 de 4.6 μm.

Sin embargo, para poder garantizar la correctitud de los resultados descritos en el Capítulo 6, solo se utilizaron los archivos etiquetados como *catalog* que representan a 1.890.715.640 objetos observados por WISE y NEOWISE.

El conjunto de datos original, al cual se le llamará *raw*, está subdividido en 36.481 archivos de los cuales la mitad son *catalog* y la otra mitad son *reject*. Cada archivo *raw* tiene formato `tbl.gz`, el cual es un archivo ASCII comprimido con formato de tabla como el que se muestra en la figura 4.1. Las tablas con *catalog* tienen 187 columnas mientras que las con *reject* tienen 188.



```
\Nsrc = 13082
\ number of unWISE epochs engaged: 7 ascending, 6 descending
\ bands engaged:   1  1  0  0
\ zero mags(band):  22.500 22.500 22.500 12.000
\ band =  1  standard Rap(band) =    8.25 arcsec ,      3.00 pix
\ band =  2  standard Rap(band) =    8.25 arcsec ,      3.00 pix
\ band =  1  circ apertures Rap =    5.50   8.25  11.00  13.75  16.50  19.25  22.00  24.75 arcsec ,      2.00   3.00   4.00   5.00   6.00   7.00   8.00   9.00 coadd pix
\ band =  2  circ apertures Rap =    5.50   8.25  11.00  13.75  16.50  19.25  22.00  24.75 arcsec ,      2.00   3.00   4.00   5.00   6.00   7.00   8.00   9.00 coadd pix
\ MJD0 = 57170.000000
\ mrgad vsn 2.5  B91121 run on 11-21-19 at 17:55:13
\ AllWISE flags retrieved from IRSA using 2.75 arcsec radius on 2019-11-21 17:46:41
\ add-ab_flags vsn 2.6  B90204 run on  1-17-20 at  9:58:51
\ artifact bitmasks from /Volumes/tyto1/Ab_masks_v1/unwise-3584p196-msk.fits
\ primaryflag vsn 1.4.1 B91105 run on 2020-01-17 at 10:07:15
\ catprep vsn 1.85 C00116 run on  1-18-20 at 20:15:22
|    source_name     |    source_id           |    ra     |    dec    |sigra  |sigdec  |sigradec|   wx    |   wy    |  w1sky  |w1sigsk|w1conf |  w2sky  |w2sigsk|w2conf |
|        char        |        char            |  double   |  double   |   r   |   r    |   r    |   r     |   r     |    r    |   r   |   r   |    r    |   r   |   r   |
|         --         |         --             |    deg    |    deg    |  asec |  asec  |  asec  |   pix   |   pix   |    dn   |   dn  |   dn  |    dn   |   dn  |   dn  |
|         --         |         --             |    --     |    --     |  null |  null  |  null  |   null  |   null  |   null  |  null |  null |   null  |  null |  null |
 J235616.50+185437.7  3584p196_b0-006284        359.0687754  18.9104905  0.1240   0.1160  -0.0848  197.031    8.461     null    null    null     null    null    null
 J235615.87+185438.5  3584p196_b0-030348        359.0661419  18.9107032  0.2923   0.2887  -0.1030  200.295    8.727     null    null    null     null    null    null
 J235640.68+185439.3  3584p196_b0-059895        359.1695092  18.9109187  0.3091   0.3259  -0.1193   72.254    9.529    0.190   5.427   0.000    1.885  17.636   0.768
 J235632.76+185439.8  3584p196_b0-018953        359.1365223  18.9110555  0.0774   0.0739  -0.0251  113.116    9.534    0.599   5.322   0.000    2.806  16.973   0.584
 J235043.61+185440.1  3584p196_b0-058400        357.6817167  18.9111587  0.4621   0.4544  -0.1667 1915.037    9.584   -1.462   4.646   0.000   -2.103  18.090   0.000
 J235632.44+185440.1  3584p196_b0-002199        359.1351684  18.9111624  0.0658   0.0397   0.0024  114.794    9.667    0.692   5.281   0.000    2.139  17.272   0.978
 J235631.93+185440.2  3584p196_b0-034095        359.1330602  18.9111711  0.2579   0.2315  -0.0870  117.405    9.668    0.104   5.005   0.000    2.042  16.649   2.329
 J235116.25+185440.3  3584p196_b0-055619        357.8177230  18.9111975  0.2833   0.2771  -0.0925 1746.569    9.014   -1.062   5.076   0.000   -2.537  18.087   0.000
 J235126.34+185440.5  3584p196_b0-070480        357.8597803  18.9112565  0.4995   0.4675  -0.1779 1694.475    8.926    0.374   5.490   0.000   -1.280  19.621   0.000
 J235109.40+185440.5  3584p196_b0-050725        357.7891916  18.9112587  0.3135   0.3106  -0.1092 1781.909    9.214   -0.937   5.156   0.000   -2.571  16.492   1.119
 J235644.37+185440.5  3584p196_b0-079652        359.1848949  18.9112656  0.4966   0.4738  -0.1823   53.197   10.067   -1.062   5.012   0.113   -0.891  18.669   0.538
 J235626.60+185440.8  3584p196_b0-052533        359.1108431  18.9113572  0.2553   0.2505  -0.0847  144.927    9.799   -0.153   5.236   0.245   -1.184  17.608   0.056
 J235636.01+185440.9  3584p196_b0-038500        359.1500742  18.9113663  0.2938   0.2717  -0.1163   96.331   10.012    1.121   5.450   0.000    4.495  19.303   1.381
 J235600.71+185441.0  3584p196_b0-061345        359.0029809  18.9114139  0.5394   0.5394  -0.2346  278.534    9.377   -0.584   4.800   0.000    2.469  18.920   1.718
 J235056.19+185441.7  3584p196_b0-063511        357.7341444  18.9115983  0.3824   0.3745  -0.1485 1850.093    9.905   -0.266   5.090   0.000   -0.090  17.586   0.745
 J235612.38+185441.8  3584p196_b0-064854        359.0516003  18.9116275  0.3405   0.3286  -0.1168  218.312    9.871   -0.327   5.395   0.000   -0.188  18.447   0.000
 J235557.63+185441.9  3584p196_b0-015220        358.9901583  18.9116403  0.1163   0.1119  -0.0428  294.417    9.620    0.851   5.688   0.000    1.437  18.010   1.238
 J235559.27+185442.0  3584p196_b0-052783        358.9969730  18.9116702  0.2885   0.2795  -0.0852  285.977    9.688   -0.159   5.157   0.000    0.992  17.673   1.106
 J235437.55+185442.1  3584p196_b0-026568        358.6564657  18.9117201  0.1055   0.1020  -0.0437  707.729    8.737     null    null    null     null    null    null
```

Figura 4.1: Ejemplo de archivo TBL

Inicialmente se tuvo que analizar el formato de las tablas para poder traspasarlo a una estructura de datos manejable. Para esto, se realizó un programa en Python que leyera un archivo línea por línea para separar los datos en cada fila de la tabla e insertarlos en un arreglo de *strings*.

Luego de procesar cada fila de un archivo, se construye un Dataframe a partir de los arreglos para transformar los archivos *raw* en JSON que van a ser insertados posteriormente en la base de datos. Como el volumen de datos es grande se prefirió usar Apache Spark a través de PySpark.

Spark es un plataforma de procesamiento paralelo de código abierto orientado a la velocidad de cómputo y a la tolerancia a fallos, mientras que PySpark es un conector de esta plataforma para Python que incorpora su propia implementación de Dataframe basado en el de Pandas, pero adaptado para trabajar con el *Resilient Distributed Dataset* de Spark.

Cabe destacar que se consultó a astrónomos del equipo de ALeRCE cuáles columnas eran las más importantes para incorporarlas en el conjunto de datos a trabajar, reduciendo el número de columnas de 186 a 20 cuyas descripciones se detallan en la tabla 4.1.

Tabla 4.1: Nombres de columnas y descripciones de los datos de CatWISE2020 incluidos en el trabajo.

Nombre de la columna	Descripción
source_name	designación hexasegimal de la fuente
source_id	tile name + código de procesamiento + índice wphot
ra	ascensión recta (ICRS)
dec	declinación (ICRS)
sigra	incertidumbre en ra (arcsec)
sigdec	incertidumbre en dec (arcsec)
w1mag	magnitud de apertura estándar con corrección aplicada
w1sigm	incertidumbre de la magnitud de apertura estándar, banda 1
w1flg	apertura estándar flag, banda 1
w2mag	magnitud de apertura estándar con corrección aplicada
w2sigm	incertidumbre de la magnitud de apertura estándar, banda 2
w2flg	standard aperture flag, banda 2
w1k	índice k de Stetson para variabilidad [12]; banda 1
w1mlq	-log(Q), Q = 1-P(chi-square); banda 1
w2k	índice k de Stetson para variabilidad [12]; banda 2
w2mlq	-log(Q), Q = 1-P(chi-square); banda 2
PMRA	movimiento en ra
PMDec	movimiento propio en dec
sigPMRA	incertidumbre en PMRA
sigPMDec	incertidumbre en PMDec

4.2. Base de datos

Tomando en cuenta la cantidad de objetos contenidos en el catálogo, según lo indicado en la sección anterior, se optó por trabajar con un sistema de base de datos distribuida. Esto se debe a su capacidad para manejar grandes volúmenes de datos sin comprometer la estabilidad del sistema.

En este trabajo se escogió como sistema de base de datos a MongoDB, el cual es orientado a documentos. Lo que destaca a MongoDB es la eficiencia en el almacenamiento de los datos a través de BSON (*Binary JSON*): un tipo de JSON que se almacena como binario que hace que sea más rápido para su lectura y ocupe menos espacio en disco.

4.2.1. Inserción de datos

En la sección 4.1 se mencionó la creación de archivos JSON que son utilizados para facilitar la inserción de datos en MongoDB ya que el formato es el apropiado para su manejo en la base de datos. Para automatizar la inserción de datos primero se guardaron los archivos en un bucket de S3 de Amazon Web Services (AWS).

4.2.2. Indexación espacial

En una base de datos, un índice es una estructura de datos que mejora la velocidad de las operaciones de búsqueda y recuperación de información a cambio de una mayor cantidad de escrituras y mayor uso de almacenamiento. Generalmente se usan estructuras con tiempos de búsqueda eficientes (menor a $O(n)$ en promedio) como *B+ trees*, árboles balanceados y *hashes*.

MongoDB tiene implementado dos índices geoespaciales para tener búsquedas eficientes en consultas con coordenadas geográficas: `2d` que usa geometría plana y `2dsphere` que usa geometría esférica. Se prefirió el uso de este último debido a su parecido a la esfera celeste aunque se asemeja más el modelo a la forma del planeta Tierra. Debido a esa diferencia, se tuvo que hacer una conversión de medidas para modelar de forma adecuada el cielo, las cuales se detallan mejor en la sección 5.1.

Para poder crear un índice espacial es necesario que el campo a indexar siga uno de los siguientes formatos:

- GeoJSON: Según la documentación[6] de MongoDB, es preferible usar este formato si se desea hacer cálculos en una geometría parecida a la Tierra. MongoDB usa el sistema de referencia WGS 84 para realizar la conversión.

```
<location field> : {
    type: <GeoJSON type>,
    coordinates: <coordinates array>
}
```

- Par de coordenadas de legado: La documentación[7] indica que es apropiado para cálculos de distancias en un plano euclidiano.

```
<field>: [ <longitude>, <latitude>]
<field>: { <field1>: <longitude>, <field2>: <latitude>}
```

En ambos casos los valores de `longitude` van entre -180 y 180 y los de `latitude` entre -90 y 90.

[6] `https://www.mongodb.com/docs/manual/geospatial-queries/#geojson-objects`
[7] `https://www.mongodb.com/docs/manual/geospatial-queries/#legacy-coordinate-pairs`

Capítulo 5

Implementación de la interfaz

En este capítulo se presenta la implementación de la interfaz a la cual se conectará la *pipeline* de ALeRCE para realizar las consultas de *crossmatching*.

Para la implementación se tomó en cuenta las funciones realizadas para las pruebas de crossmatching que se mencionan en el capítulo 6 y que ahora se describen en mayor detalle. En general todo el código se desarrolló en Python.

5.1. Crossmatching

Esta parte del código se puede separar en dos procesos principales: la conversión de unidades y la consulta de búsqueda.

5.1.1. Conversión de unidades

Para la conversión de unidades hay que tomar en cuenta que, tal como se mencionó en la sección 4.2.2, MongoDB usa el sistema *World Geodesic System 1984* (o WGS 84) para modelar la esfera de la Tierra, la cual considera los parámetros indicados en la tabla 5.1[1]:

Tabla 5.1: Parámetros de WGS 84 para calcular medidas sobre la Tierra.

Semieje mayor (a)	6.378.137,0000
Semieje menor (b)	6.356.752,31414
Excentricidad (e)	0,081819190842600

Estos parámetros son necesarios ya que se conoce que la forma de la Tierra no es una esfera perfecta, sino que está "achatada" en los polos, lo cual se puede interpretar que la Tierra tiene forma de un elipsoide.

[1]Ver Example 1 en "Sample Outputs From Programs INVERSE and FORWARD" en `https://geodesy.noaa.gov/PC_PROD/Inv_Fwd/readme.htm`.

Para cada posición que se quiere localizar, se calcula el radio de curvatura meridional [11]:

$$RM = \frac{a(1-e^2)}{(1-e^2 \sin^2 lat)^{\frac{3}{2}}}$$

Donde a es el semieje mayor, e es la excentricidad y lat es la latitud de la posición que se quiere consultar. A partir del resultado de RM se calcula el largo del arco que corresponde al ángulo de 1° en radianes:

$$Arco = RM \cdot 0{,}01745329$$

En general, para este trabajo se realiza un desplazamiento en 180 grados para la ascensión recta debido al rango de valores que admite la longitud según lo mencionado en la sección 4.2.2. Debido a lo anterior, los valores para la longitud lon y latitud lat utilizados son los siguientes:

$$lon = ra - 180$$
$$lat = dec$$

5.1.2. Consultas de búsqueda

Para la consulta de *conesearch* se aprovecha la presencia de los índices espaciales para optimizar la búsqueda. Además, MongoDB permite fijar un valor para la distancia máxima entre el centro del cono y los resultados a mostrar.

Se implementó una función que primero aplica la conversión de unidades explicada en la sección anterior, la cual se ocupa para obtener el radio de búsqueda en metros. Lo anterior se debe a que el operador *maxDistance* utiliza metros como unidad de medida[2] cuando se entregan las coordenadas como GeoJSON.

Después se procede a realizar la consulta de búsqueda a través de la función `find` que está incluido en PyMongo y que es equivalente a la función homónima en el shell de MongoDB, mongosh.

El siguiente extracto de código muestra la consulta que se realiza a la base de datos:

```
cursor = coll.find({'loc': {
    '$nearSphere': {
        '$geometry': {'type': 'Point',
                      'coordinates': [lon, lat]},
        '$maxDistance': meter_radius*0.0002778
        }}})
```

Según la documentación[3,4], `find` recibe un diccionario con las condiciones con las que se desea filtrar la búsqueda.

[2] `https://www.mongodb.com/docs/v5.3/reference/operator/query/maxDistance/`

[3] `https://pymongo.readthedocs.io/en/stable/api/pymongo/collection.html#pymongo.collection.Collection.find`

[4] `https://www.mongodb.com/docs/manual/reference/method/db.collection.find/`

5.2. API

Para implementar la API con el endpoint que realiza las consultas se utilizó la librería *FastAPI*[5]. Para esto se utilizó como guía la página web de la documentación de X-Match, más específicamente la sección 3.2 que trata sobre las llamadas a la API a través del protocolo HTTP[6].

Los endpoints habilitados son los siguientes:

Lista de Catálogos (método GET)

Muestra una lista de los nombres de los catálogos existentes en el sistema. Se accede a través de la URL "/catalog". Este *endpoint* no recibe ningún parámetro.

Un ejemplo de respuesta se ilustra de la siguiente manera:

```
[
  "test_objs",
  "catwise2020"
]
```

Crossmatch (método POST)

Este endpoint entrega una lista de los objetos encontrados en el *crossmatch* entre un determinado catálogo y una lista de posiciones. Se accede a través de la URL "/crossmatch". Los parámetros que recibe este endpoint son los siguientes:

- *radius*: Radio de búsqueda para realizar el *conesearch* en segundos de arco.
- *input*: Archivo de texto con el conjunto de ubicaciones sobre las cuales se realiza el *crossmatch*. Se realizó como una forma de optimizar el número de peticiones que se realizan al servicio, en lugar de mandar una posición por petición HTTP. Un ejemplo del archivo a subir es el siguiente:

```
ra,dec
86.40498828654472,-28.936177761791473
89.26862915603925,-24.63840462743747
91.94042231894309,-20.290029497760873
94.46552057758339,-15.903357618473493
96.88229207387474,-11.48861973015191
99.22424989029605,-7.054602780278212
101.52164247516362,-2.6091415241310076
```

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

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

- *limit* (opcional): Número máximo de resultados por *conesearch* realizado en cada punto. El valor por defecto de este parámetro es *None*.
- *catalog* (opcional): Catálogo astrónomico dentro del sistema con el cual se realiza el *crossmatch*. Por defecto de este parámetro es *"catwise2020"*.

A continuación se muestra un ejemplo de una consulta a través de la librería *requests*[7]:

```
request_params = {'radius': 2, 'limit': 2}
input_file = {'input': open("galactic_plane.txt", 'rb')}
results = requests.post("http://23.23.87.67:8000/conesearch/", params=
    request_params, files=input_file)
```

Y la respuesta a la petición entrega los siguientes campos:

- *total_search_time*: Tiempo total que tarda en realizar la consulta a la base de datos. Es la suma de los tiempos en que se generan los cursores para cada posición entregada. Se muestra en formato [H]H:MM:SS[.UUUUUU].
- *generate_file_time*: Tiempo total que tarda en generar la lista entregada en el campo *results*. Es la suma de los tiempos en que se itera sobre el cursor para obtener los resultados como JSON.
- *num_objs*: Número de objetos encontrados en el *crossmatch*.
- *results*: Una lista de objetos en formato JSON con los datos mencionados en la tabla 4.1.

Un ejemplo de la respuesta se muestra en el siguiente texto:


```
{
    "total_search_time": "0:00:00.007898",
    "generate_file_time": "0:00:00.417975",
    "num_objs": 2,
    "results": [
        {
            "_id": "3164m409_b0-036691",
            "PMDec": 0.21327,
            "PMRA": 0.08757,
            "loc": {
                "coordinates": [
                    136.28622639999998,
                    -40.9142604
                ],
                "type": "Point"
            },
            "sigPMDEC": 0.0929,
            "sigPMRA": 0.0907,
```


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

```
            "sigdec": 0.2345,
            "sigra": 0.2304,
            "source_name": "J210508.69-405451.3",
            "w1flg": 3,
            "w1k": 0.64565,
            "w1mag": 15.245,
            "w1mlq": 3.62,
            "w1sigm": 0.019,
            "w2flg": 3,
            "w2k": 0.84591,
            "w2mag": 14.967,
            "w2mlq": 0.72,
            "w2sigm": 0.048
        },
        {
            "_id": "2141p136_b0-017938",
            "PMDec": 0.04471,
            "PMRA": -0.02622,
            "loc": {
                "coordinates": [
                    33.60863939999999,
                    13.0087861
                ],
                "type": "Point"
            },
            "sigPMDEC": 0.05,
            "sigPMRA": 0.0435,
            "sigdec": 0.1467,
            "sigra": 0.1263,
            "source_name": "J141426.07+130031.6",
            "w1flg": 3,
            "w1k": 0.91693,
            "w1mag": 16.557,
            "w1mlq": 5.6,
            "w1sigm": 0.052,
            "w2flg": 3,
            "w2k": 0.9179,
            "w2mag": 17.141,
            "w2mlq": 0.18,
            "w2sigm": 0.381
        }
    ]
}
```

Capítulo 6

Experimentos

Para poder verificar que el nuevo servicio es útil a los objetivos que tiene ALeRCE, se tuvo que hacer pruebas para ver si podía asemejarse o mejorar el tiempo de ejecución de las consultas con respecto a CDS. También es relevante verificar que los resultados sean correctos, es decir que los objetos detectados efectivamente estén en el radio de búsqueda al momento de realizar la consulta.

La máquina en la cual se cargó el catálogo y se construyó el sistema es una instancia m5.xlarge[1] de Amazon *Elastic Compute Cloud* (EC2) que tiene un procesador Intel(R) Xeon(R) Platinum 8259CL de 4 núcleos y 16 GB de memoria RAM. Para el almacenamiento se incorporaron dos dispositivos de estado sólido (SSD), uno de 30 GB para instalar el software necesario y otro de 1 TB para almacenar el catálogo y sus índices de la base de datos, a través de Amazon Elastic Block Store[2].

6.1. Explicación de los experimentos de consulta

Para ejecutar los experimentos se tomaron distintas muestras de posiciones en la esfera celeste:

1. Posiciones de objetos escogidos aleatoriamente de CatWISE.
2. 450 puntos tomados en el plano galáctico de la Vía Láctea.
3. 180 puntos de plano perpendicular al de la Vía Láctea.

Para el conjunto 1 se escogieron objetos aleatorios extraídos de CatWISE2020 de los cuales solo se conservó su posición. Además se generó otro conjunto aleatorio a partir de la función *uniform* de la librería *random* de Python.

[1] `https://aws.amazon.com/es/ec2/instance-types/m5/`
[2] `https://aws.amazon.com/es/ebs/`

Para los conjuntos 2 y 3 se utilizó la librería *astropy* para obtener las posiciones a partir del sistema de coordenadas galáctico. En la figura 6.1 se puede observar que el sistema ocupa dos coordenadas: la longitud ℓ y la latitud b, cuyos valores van entre los 0 y 360 grados.

Para la muestra 2 se fijó b en cero y las posiciones en ℓ se separaron en 0.8 grados, mientras que para 3 se fijó ℓ en cero y b estuvo en el rango entre -90 y 90 grados con una separación de 1 grado.

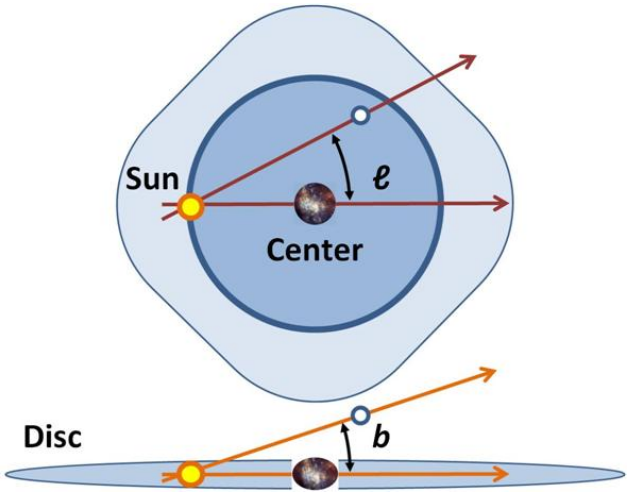


Figura 6.1: Representación de las coordenadas del sistema galáctico [9].

A partir de lo anterior se realizaron consultas a CDS y el servicio desarrollado en este trabajo para radios de 1, 2, 4, 8 y 16 segundos de arco[3].

En el caso de CDS la consulta se realizó a través de su interfaz web utilizando el catálogo CatWISE2020 (el mismo que se trabajó en la sección 4.1). Para el sistema propio, se ejecutó un programa de Python desde la instancia que contiene la base de datos en AWS. Posteriormente también se realizaron pruebas con consultas a la API descrita en la sección 5.2.

6.2. Resultados

6.2.1. Correctitud

Para verificar la correctitud se comparó los resultados que entregan cada uno de los sistemas contando la cantidad de objetos que entrega cada uno y contabilizando las coincidencias entre los resultados de ambos sistemas.

[3]Equivale a $\frac{1}{3600}$ grados sexagesimales.

Posiciones de objetos seleccionados aleatoriamente

Tabla 6.1: Conteo de objetos encontrados en el *crossmatch* en CDS, el sistema implementado y las coincidencias entre ambos con la muestra aleatoria.

Radio (arcsec)	CDS	Sistema propio	Ambos
1	18.185	18.179	18.177
2	18.300	18.296	18.293
4	19.440	19.430	19.416
8	29.445	29.365	29.314
16	81.797	81.293	81.199

Como se puede observar en la tabla 6.1, el sistema desarrollado en este trabajo encuentra menos coincidencias que CDS. Una hipótesis sobre el motivo de esta diferencia es que pudo deberse a que CDS encontró objetos que se encuentran en el borde del radio de búsqueda.

Para comprobar la hipótesis se realizaron consultas de radios entre $16 - 2^m$, con m entre -5 y 3. Solo se realiza con estos valores de radio para determinar el margen de error en el radio de detección y para descartar que el sistema desarrollado tenga un error muy grande.

Tabla 6.2: Conteo de objetos encontrados en los dos sistemas y las coincidencias entre ambos con la muestra aleatoria entre 8 y 16 segundos de arco comparando con resultados del CDS para 16 arcsec.

Radio	Sistema Propio	Coincidencias de CDS	Diferencia
16	81.439	81.345	94
15,96875	81.162	81.128	34
15,9375	80.900	80.886	14
15,875	80.357	80.356	1
15,75	79.307	79.307	0
15,5	77.227	77.227	0
15	73.294	73.294	0
14	65.691	65.691	0
12	51.710	51.710	0
8	29.366	29.366	0

En la tabla 6.2 se reflejan las coincidencias para un resultado que contiene 81.439 objetos obtenidos con CDS para un radio de 16 segundos de arco para la muestra de 20 mil posiciones, mientras que en las pruebas reflejadas en la tabla 6.3 se usó como base el resultado, para la misma muestra y el mismo radio, entregado por el sistema propio.

Tabla 6.3: Conteo de objetos encontrados en los dos sistemas y las coincidencias entre ambos con la muestra aleatoria entre 8 y 16 segundos de arco comparando con resultados del sistema propio para 16 arcsec.

Radio	CDS	Matches	Diff
16	81.796	81.345	451
15,96875	81.532	81.249	283
15,9375	81.246	81.104	142
15,875	80.718	80.698	20
15,75	79.648	79.648	0
15,5	77.591	77.591	0
15	73.595	73.595	0
14	65.972	65.972	0
12	51.943	51.943	0
8	29.444	29.444	0

Se puede observar que la diferencia entre las coincidencias de los resultados de ambos sistemas, y el resultado de CDS, disminuye cuando el radio se acerca a 15,75.

Posiciones en el plano galáctico

Al tratarse de un conjunto de objetos más pequeño para comparar, las diferencias pueden verse amplificadas.

Tabla 6.4: Conteo de objetos encontrados en el *crossmatch* en CDS, el sistema implementado y las coincidencias entre ambos con la muestra del plano galáctico.

Radio (arcsec)	CDS	Sistema propio	Ambos
1	8	7	7
2	23	23	23
4	118	115	115
8	435	431	431
16	1.892	1.883	1.880

Según los datos mostrados en la tabla 6.4, hay una diferencia entre el 0,6 % y 2,6 % entre el número de objetos mostrados por ambos sistemas y para los que muestra CDS.

Posiciones en el plano perpendicular a la galaxia

Tabla 6.5: Conteo de objetos encontrados en el *crossmatch* en CDS, el sistema implementado y las coincidencias entre ambos con la muestra del plano perpendicular.

Radio (arcsec)	CDS	Sistema propio	Ambos
1	2	2	2
2	8	7	7
4	35	34	34
8	125	122	122
16	511	504	504

Con respecto a los resultados relacionados a esta muestra, se infiere de la tabla 6.5 que la diferencia oscila entre un 1,3 % y 2,9 %.

6.2.2. Velocidad

Para poder comparar la velocidad se midió el tiempo promedio para 20 consultas en la API implementada en este trabajo y en la API de CDS X-Match cuya URL base es `http://cdsxmatch.u-strasbg.fr/xmatch/api/v1/sync` debido a que la interfaz web no entregaba tiempos precisos cuando estos eran menores a 1 segundo. Las consultas se realizaron a través de Postman.

Posiciones de objetos seleccionados aleatoriamente

Para este conjunto de datos se realizaron distintas pruebas considerando que tiene la muestra más grande de los tres conjuntos.

Primero se realizó, a través de un programa, una serie de llamadas a las API de CDS y del sistema propio de manera secuencial y se calculó el promedio de ejecución de las consultas realizadas con la librería *requests*. El comportamiento del tiempo con respecto al radio de búsqueda se ve reflejado en la figura 6.2.

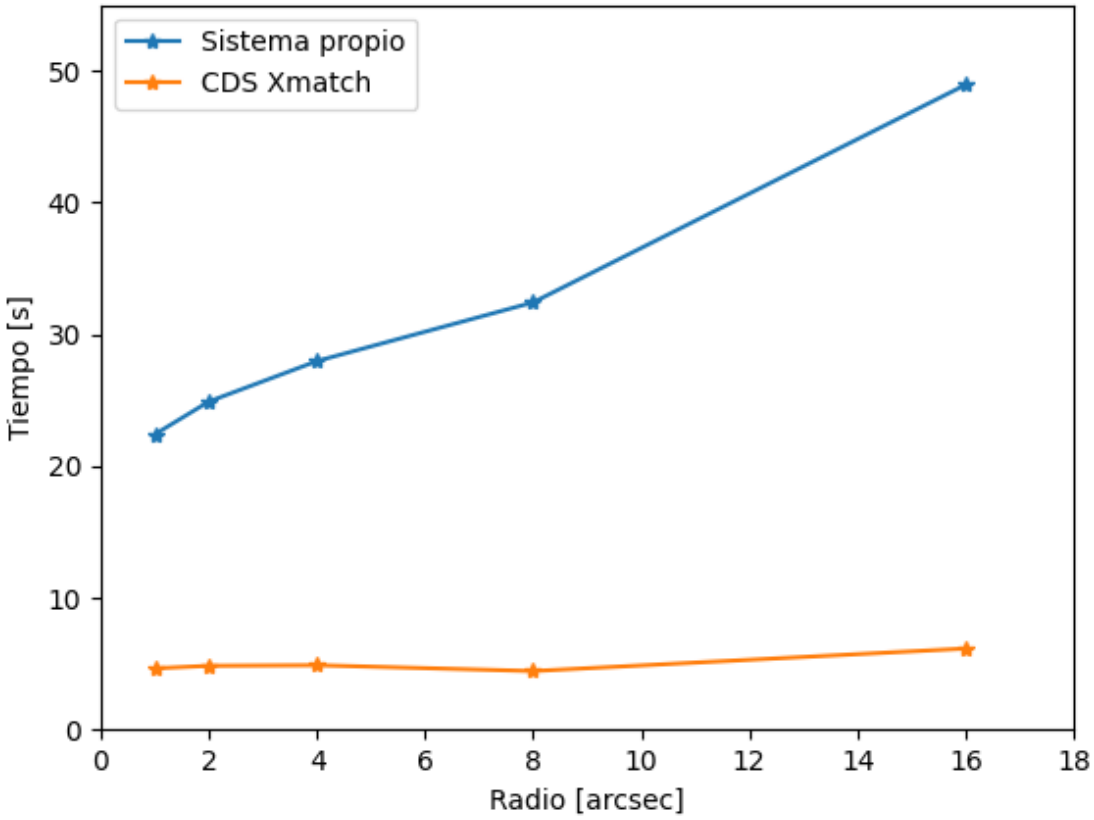


Figura 6.2: Tiempo de ejecución en CDS y la API para la muestra de 20 mil puntos aleatorios

Posteriomente se generó un nuevo archivo de muestra con 50 mil posiciones, donde 20 mil son aquellas utilizadas anteriormente y las restantes fueron generadas aleatoriamente. Primero se realizó una prueba con la muestra entera en una sola consulta a las APIs a través de Postman, cuyos tiempos se pueden ver graficados en la figura 6.3.

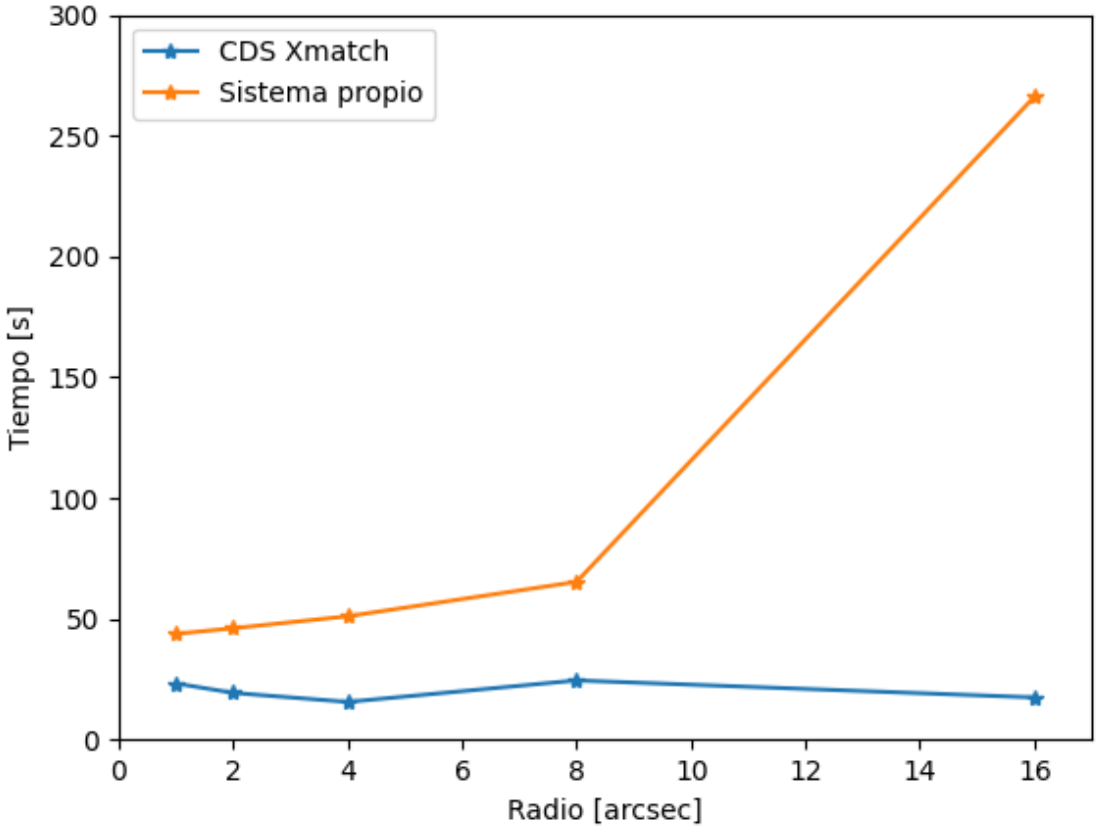


Figura 6.3: Tiempo de ejecución en CDS y la API para la muestra de 50 mil puntos aleatorios

Para probar otras maneras de hacer las consultas, se dividió la muestra en 5 partes de 10 mil posiciones cada una y se enviaron de forma paralela a través de *threads* de Python. La imagen en la figura 6.4 muestra una mejoría con respecto al caso en que se envían los 50 mil puntos en una sola petición.

Sin embargo, ese no es un cambio que se realizó desde el servidor que hospeda el servicio, sino que fue un esfuerzo adicional del lado del cliente.

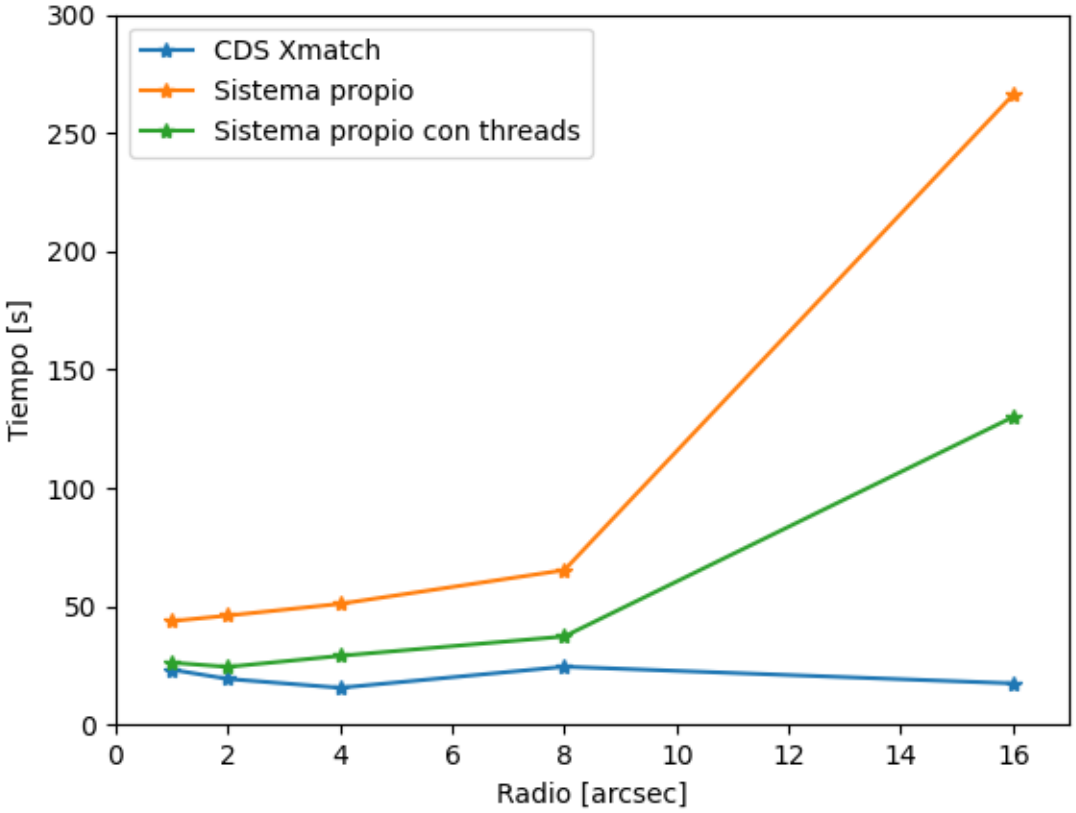


Figura 6.4: Tiempo de ejecución en CDS y la API con y sin paralelismo para la muestra de 50 mil puntos aleatorios.

Posiciones en el plano galáctico

En la figura 6.5 se puede observar dos cosas. La primera es que el tiempo de respuesta de la API escala de forma lineal a pesar de que el radio de búsqueda escala exponencialmente. La segunda es que para radios pequeños CDS es menos eficiente para entregar resultados.

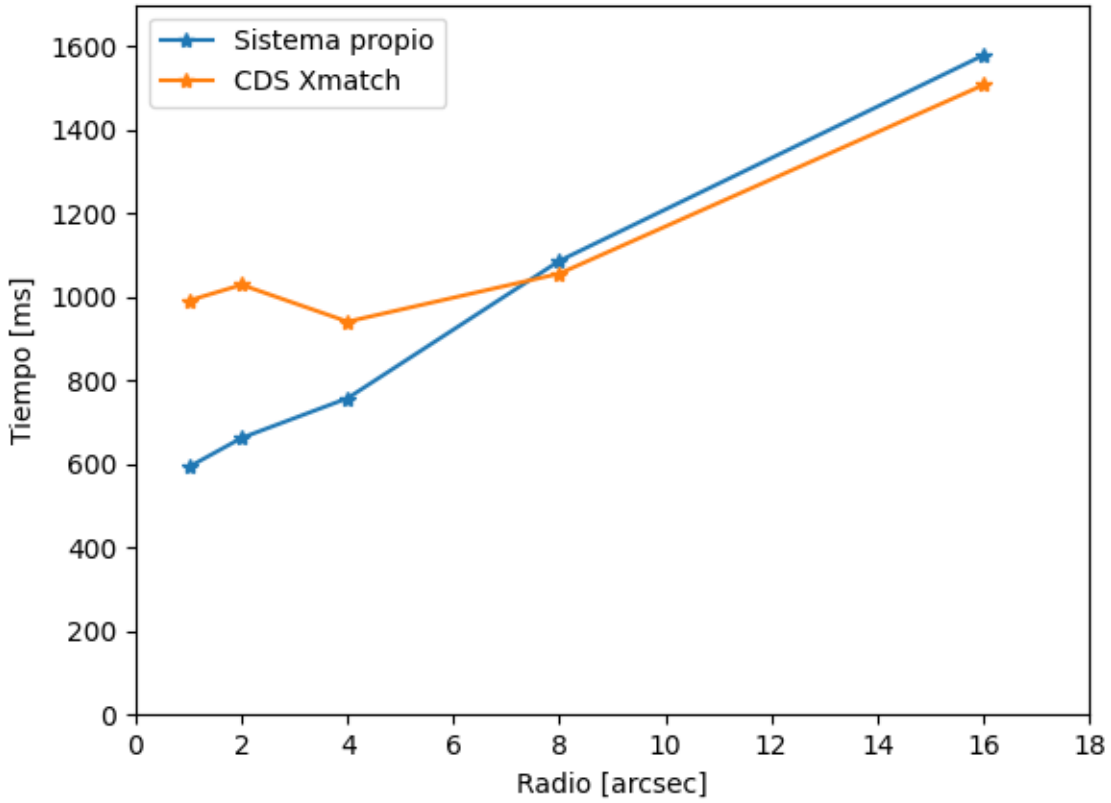


Figura 6.5: Tiempo de ejecución en CDS y la API para la muestra de puntos en el plano galáctico.

Posiciones en el plano perpendicular a la galaxia

En la figura 6.6 se muestra que el sistema propio tiene mejor rendimiento que CDS, lo cual tiene sentido considerando que es la muestra con menos posiciones de todas, por lo que va a tomar menos tiempo procesar.

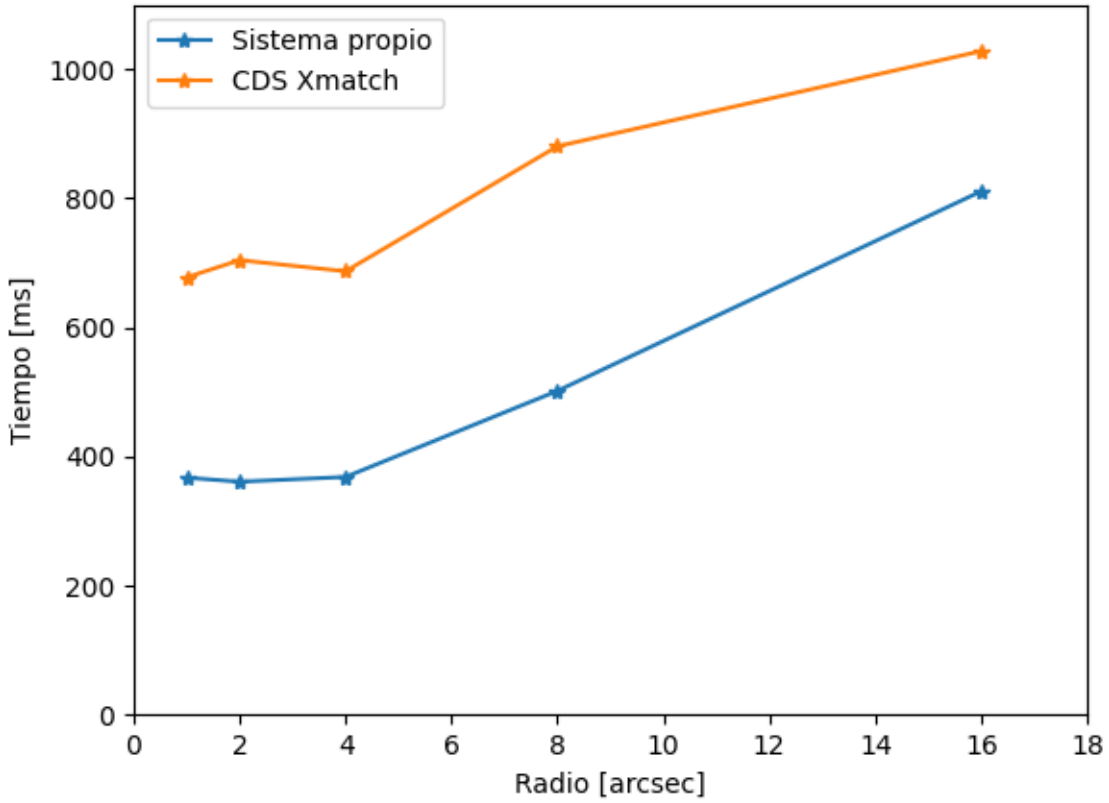


Figura 6.6: Tiempo de ejecución en CDS y la API para la muestra de puntos en el plano perpendicular a la galaxia.

A partir de las figuras mostradas en esta sección, se puede observar un incremento importante en el tiempo de ejecución para el sistema implementado a partir del radio de 8 segundos de arco. Además, se puede deducir que se evitó el sobrecosto en la implementación de CDS para obtener tiempos de ejecución de orden relativamente constante, cosa que no se logró en este trabajo. Lo anterior puede deberse tanto a la implementación como a las tecnologías utilizadas para crear el software.

Capítulo 7

Conclusión

7.1. Trabajo Realizado

El resultado de este trabajo fue un sistema que permite realizar *crossmatching* a través de una API que se puede conectar a cualquier sistema, ya sea a través de una aplicación web u otros métodos de realizar peticiones HTTP.

Con respecto a los objetivos específicos planteados en la sección 1.2, se puede decir que sí se cumplieron en su totalidad además de que sirvieron como una hoja de ruta para conseguir el producto final. Los objetivos 1, 2, 3 y 4 fueron explicados en el capítulo 4, el objetivo 5 en el capítulo 5 y finalmente el objetivo 6 se vio reflejado en lo detallado en el capítulo 6.

Sobre la correctitud de los resultados, se puede concluir que el error ronda el 1 %, lo cual puede ser significativo o no dependiendo del caso de uso. Sin embargo, faltó evaluar la aparición de objetos en los resultados del sistema propio que no se presentaban en los de CDS.

Con respecto a los tiempos de consulta, este trabajo permite búsquedas más rápidas que CDS X-Match para lotes pequeños de posiciones. Sin embargo, para consultas más grandes el sistema tiende a empeorar el tiempo en que realiza las consultas.

En resumen, se logró generar un sistema funcional que puede ser mejorado para lograr tiempos de procesamiento más cortos.

7.2. Trabajo Futuro

A pesar de haber logrado los objetivos en su gran mayoría, hay algunos detalles que se pueden perfeccionar para mejorar el rendimiento del sistema tales como:

- Usar *sharding*: MongoDB tiene la capacidad de escalar horizontalmente y lo puede hacer a través del *sharding*, el cual consiste en agregar más máquinas que comparten

los datos y el procesamiento para optimizar las operaciones de lectura y escritura.

- Paralelizar la recuperación de los resultados a partir del cursor: Para obtener los resultados de una consulta en MongoDB se debe iterar sobre el cursor que entrega, lo cual puede ser lento si la consulta entrega muchos resultados.

También existen formas de agregar valor al sistema implementado. Algunos de ejemplos de esto pueden ser:

- Agregando catálogos a la base de datos (como Gaia).
- Implementando una interfaz web para realizar consultas de *crossmatching*.
- Agregando más endpoints que cumplan otro propósito con el mismo catálogo.
- Agregar más formatos de archivo de la respuesta como CSV y VOTable.

Bibliografía

[1] 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. *Publications of the Astronomical Society of the Pacific*, 131(995):018002, dec 2018.

[2] 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.

[3] Ian N. Evans, Francis A. Primini, Kenny J. Glotfelty, Craig S. Anderson, Nina R. Bonaventura, Judy C. Chen, John E. Davis, Stephen M. Doe, Janet D. Evans, Giuseppina Fabbiano, and et al. The chandra source catalog. *The Astrophysical Journal Supplement Series*, 189(1):37–82, Jun 2010.

[4] Eric D. Feigelson and G. Jogesh Babu. Big data in astronomy. *Significance*, 9(4):22–25, 2012.

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

[6] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, jun 2002.

[7] 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.

[8] 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.

[9] Brews ohare. Galactic coordinates, 2008. `https://commons.wikimedia.org/wiki/File:Galactic_coordinates.JPG`.

[10] Raymond Plante, Roy Williams, Robert Hanisch, and Alex Szalay. Simple cone search version 1.03.

[11] Richard H Rapp. *Geometric Geodesy Part I.* Ohio State University Department of Geodetic Science and Surveying, 1958 Neil Avenue, Columbus, Ohio 43210, 1991.

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

[13] Alexander S. Szalay, Jim Gray, George Fekete, Peter Z. Kunszt, Peter Kukol, and Ani Thakar. Indexing the sphere with the hierarchical triangular mesh, 2007.

[14] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, page 10, USA, 2010. USENIX Association.