Optimizando búsqueda de polígonos geográficos en go

Rediseñando el servicio core de áreas en PedidosYa

Luis Gabriel Gomez

@lggomez

PeYa en números

15 países

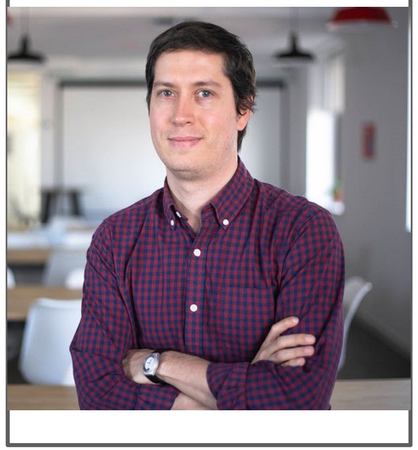
+500 ciudades

+4500 personas

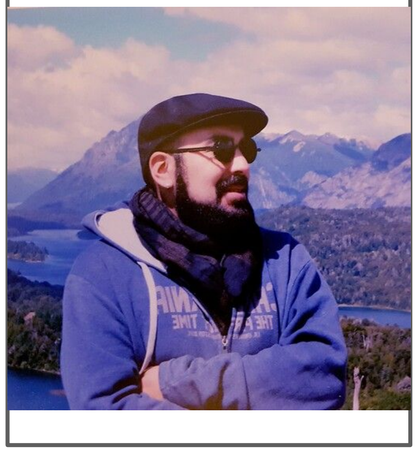
+100 PedidosYa Markets

PedidosYa

Locations backend - El equipo

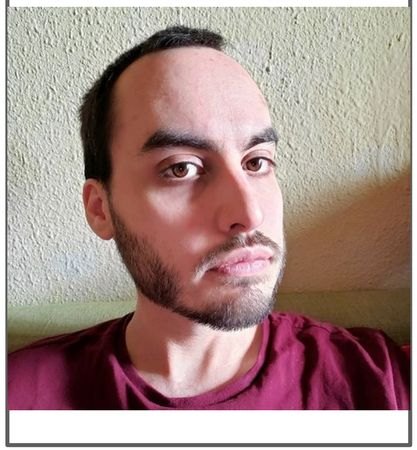


German Ubillos
Software Engineering Manager



Luis Gabriel Gomez
Principal Software Engineer

(Quien les habla en esta agradable velada)



Emanuel Mendez
Software Engineer Backend

PedidosYa

Definiendo el problema

Qué es una location?

Locations - áreas

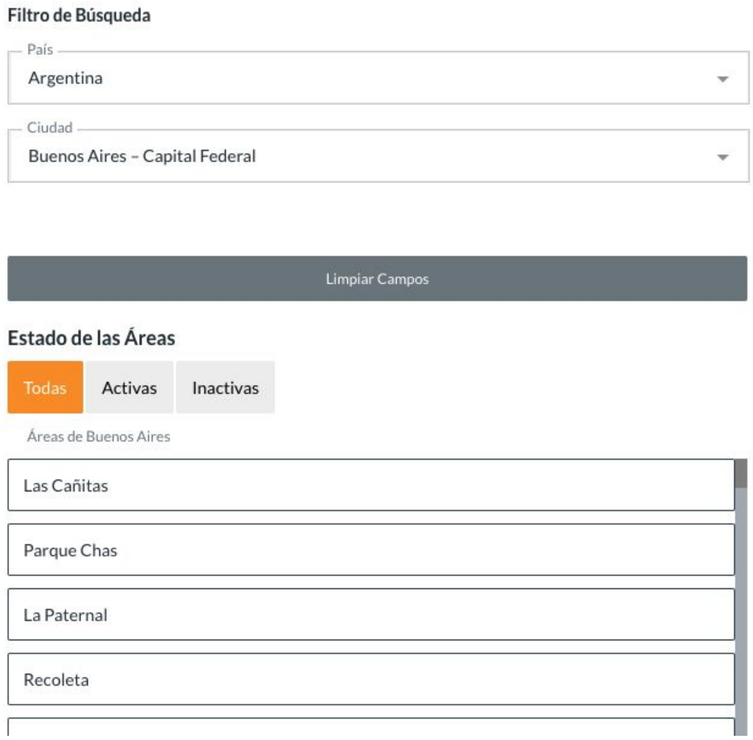
- En PedidosYa, cuando un usuario inicia la aplicación a partir de su ubicación se obtiene su identificador de área (area_id). Este flujo es crítico y cubierto por el servicio de locations

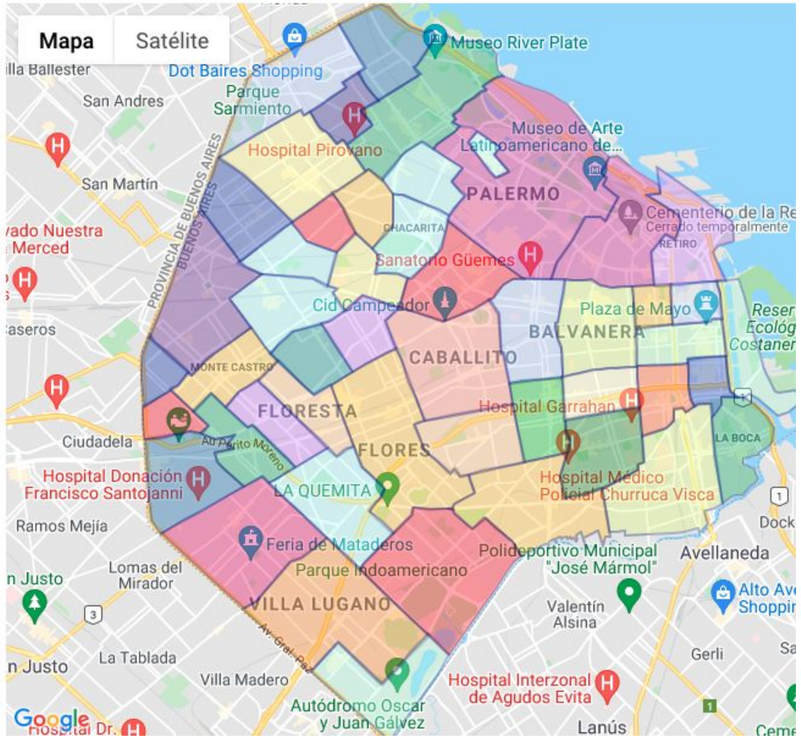
- Este dato es necesario, ya que a partir del área determinamos la ciudad y se utiliza para guardar direcciones

PedidosYa

Locations - áreas (cont.)

Como se puede ver desde el backoffice, las áreas vienen dadas por país y ciudad





PedidosYa

Locations - entidad

De lo visto recién, vamos a tener una entidad llamada location que consiste en:

- country_id
- city_id
- *área (polígono)*
- Metadatos (nombres, campos opcionales, etc)

Y el área a su vez está formada por:

- area_id
- country_id
- city_id
- *puntos cardinales (aristas)*
- Metadatos (nombres, campos opcionales, etc)

PedidosYa

Requerimiento

Es un servicio core con múltiples clientes dentro de PedidosYa, y cubre varias responsabilidades:

- Servir búsquedas de área bajo múltiples casuísticas
  - coordenadas con y sin country_id (búsqueda adicional de país)
  - get por area_id (acceso directo)
- Mantenerse al dia (eventualmente consistente) con la fuente de verdad que hasta el día de hoy es la DB legacy
- Dar metricas para negocio (penetración y conversión de usuarios)

PedidosYa

Requerimiento parte II - La venganza

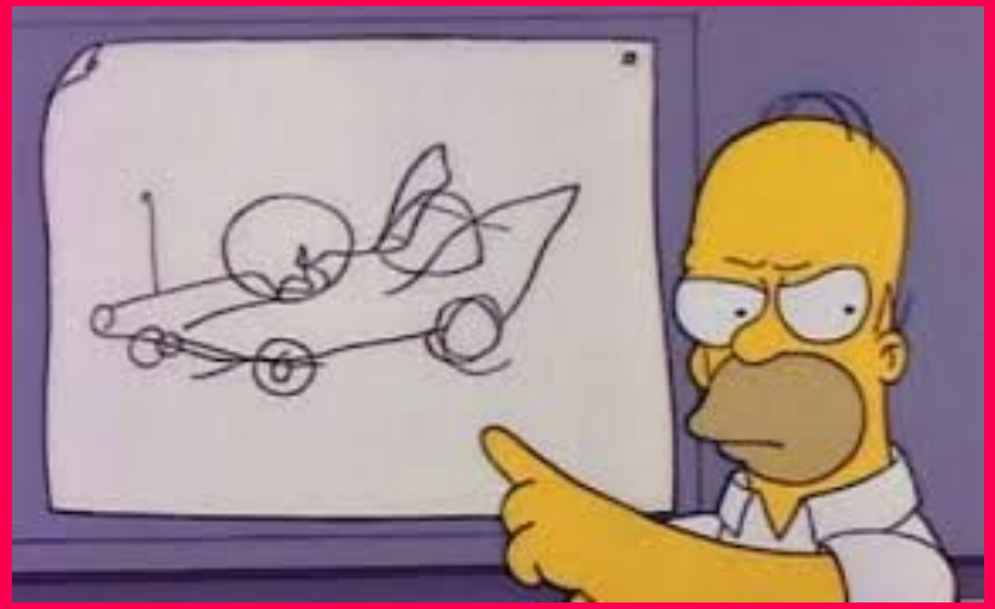
Sumado a lo anterior:

- Latencia mínima y consistente, tener worst cases negligibles
- Minimizar el margen de inconsistencia en lo posible
- Manejo de cache-control
- *Posibilidad de obtener áreas cercanas (en caso de que el punto buscado no pertenezca a una)*

PedidosYa

Requerimientos de la solución

Qué hay? Que podemos y estamos haciendo?



Locations v1 - Actual & Legacy

DEPRECADOSE - Microservicio en grails que, realiza las búsquedas y sirve los desde memoria. Dichos datos se cargan desde una DB MySQL:

- Se almacenan los puntos en memoria, realizando una búsqueda por raycasting
- Se recorren las áreas de forma lineal si no hay country_id, creando un *worst case incremental* en base a las áreas nuevas y cantidad de elementos
- En su forma general, se realiza el recorrido de un mapa (el cual está indexado por country_id cuando se dispone)
- Dicha base en memoria se refresca cada 60 minutos hardcodeados en código

PedidosYa

Locations v1 - Problemas a resolver en v2

- Performance: La estructura de indexado y disposición de las áreas es suboptima, los worst cases generan percentiles p95 superiores a 4 milisegundos y p99 de hasta 15-20 milisegundos
- Resiliencia: Grails no es una plataforma muy amistosa en este escenario con containers (heap size vs working memory)
- Extensibilidad: Dado el diseño actual no se pueden aplicar diferentes lógicas de búsqueda o extender la actual sin degradar aún más la performance
- Consistencia: Es deseable un diseño más reactivo y autónomo de actualización de los datos en memoria, configurable

PedidosYa

Un poco de geometría

Puntos? Polígonos?

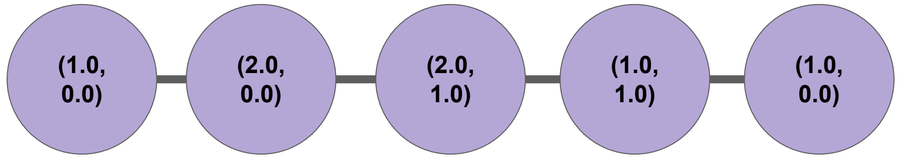
Definiendo un polígono

Cómo llegamos desde un conjunto de coordenadas geográficas hasta la figura vista en backoffice (polígono)??? en 3 pasos (y abstracciones geométricas):

1-

Este es un **punto** único (par de coordenadas)

2-



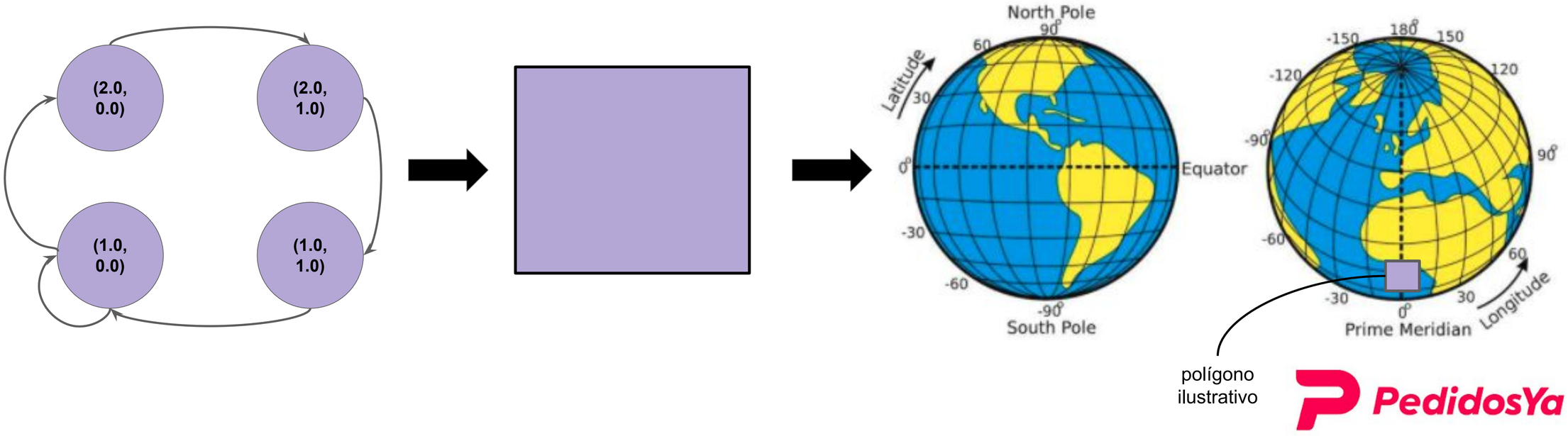
Este es un **anillo de puntos** (array). Podemos ver que el último punto se corresponde con el primero, cerrando el anillo

PedidosYa

Definiendo un polígono en el globo

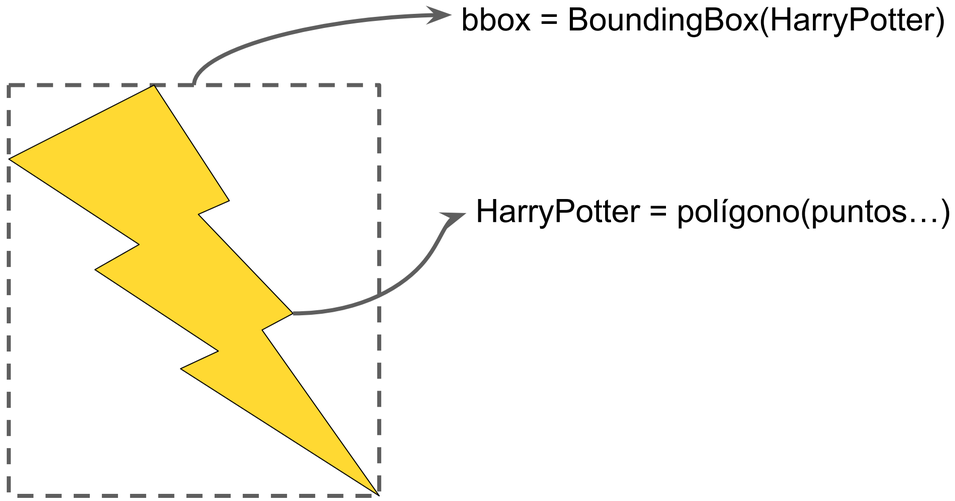
Paso 3: Teniendo como referencia el sistema de coordenadas globales, podemos visualizar el anillo como polígono de la siguiente manera:

3-



Polígonos - Bounding boxes

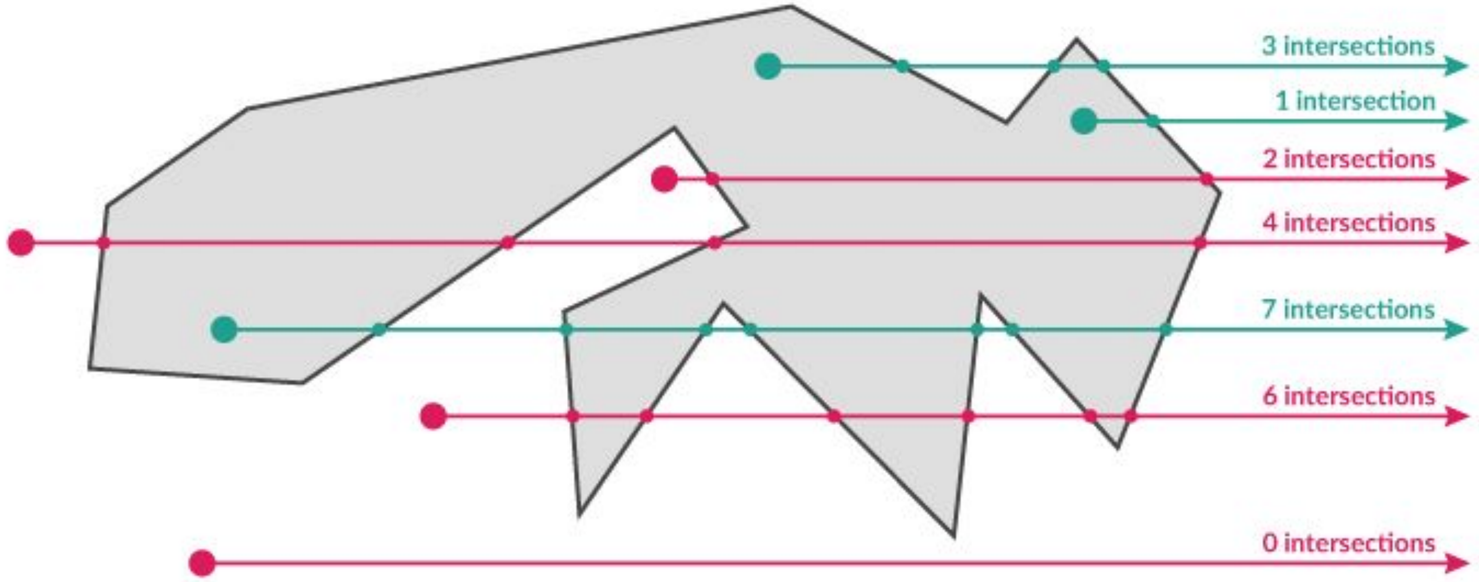
Dado un polígono, su bounding box es el rectángulo mínimo que encierra al mismo:



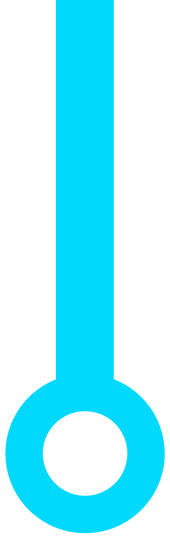
PedidosYa

Polígonos - Búsqueda de puntos: Raycasting

Dado un punto, para saber si se encuentra en un polígono se utiliza el algoritmo denominado *raycasting*: se proyecta el punto y dada la paridad de intersecciones con aristas del polígono se determina la inclusión o no del mismo



PedidosYa



Geometría en go - packages

Tuvimos 2 opciones relevadas:

1. github.com/golang/geo: Package dentro de la org. oficial, orientado a geometrías esféricas y cartesianas, con API orientada a primitivas lineares, planares y de mayores dimensiones
2. github.com/paulmach/orb: Opción con buen soporte de la comunidad y variedad de packages (incluyendo geometría planar), soporte para geoJSON e implementación nativa de Raycasting bajo una API simplificada

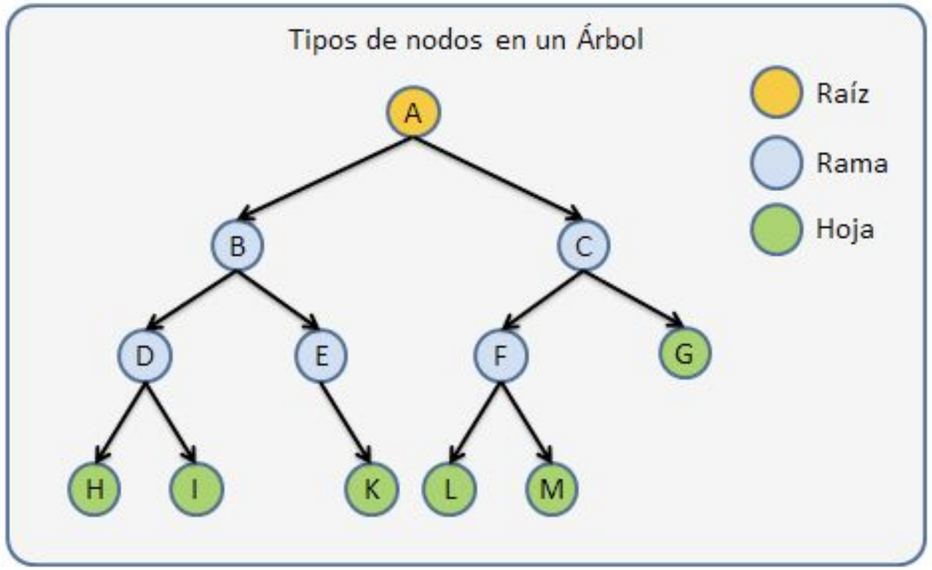
PedidosYa

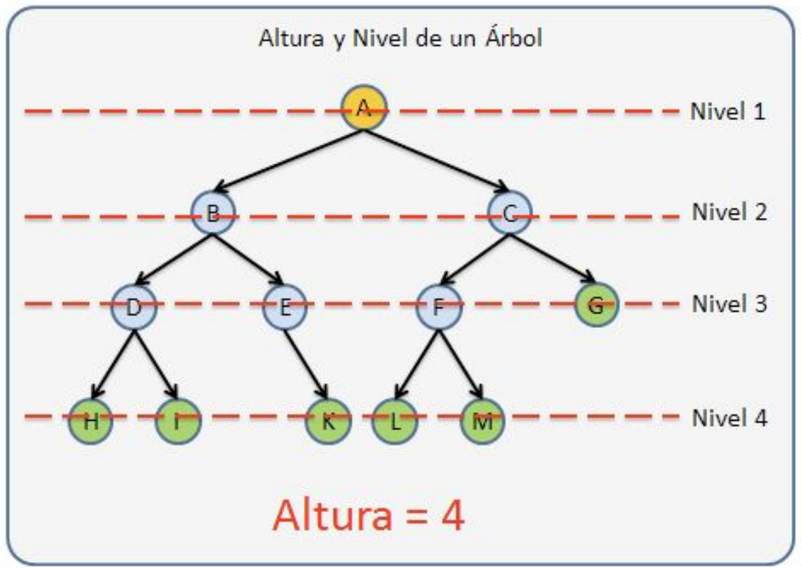
Un poco de computación

Una vuelta por estructuras y algoritmos relevantes al problema

Árboles (trees)

Los árboles son una familia de estructuras de datos orientadas a almacenar datos de forma jerárquica y para (en nuestro caso) otorgar eficiencia en su búsqueda (también tienen otros usos como compresión o deduplicación de datos)





edidosYa

Árboles - Tipos

Como vimos anteriormente, siempre vamos a tener nodos y aristas, aunque dicha representación siempre puede estar sujeta al tipo de árbol y especialmente a la implementación del mismo (ejemplo: muchas implementaciones no tienen representadas las aristas)

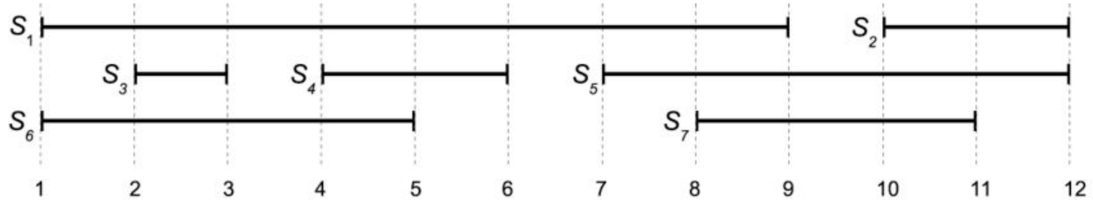
Para la solución planteada en locations v2 se usan 2 tipos árboles especializados para indexado y almacenamiento de datos respectivamente:

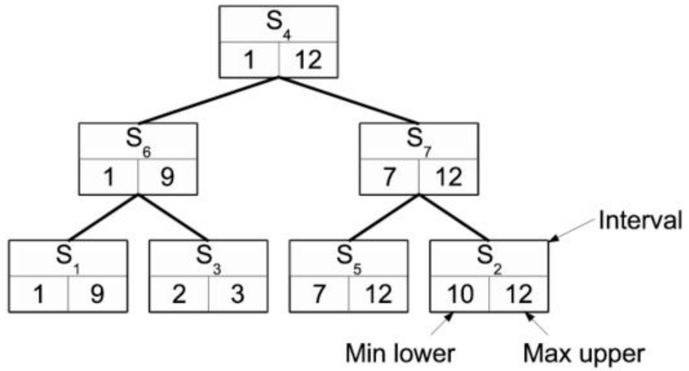
- Interval trees
- R*-trees

PedidosYa

Árboles - Interval trees

Los Interval trees (arboles de intervalos) son una variante de árbol binario que almacena intervalos, y, dado un valor de búsqueda se devuelven los intervalos en los que está incluido ese número:

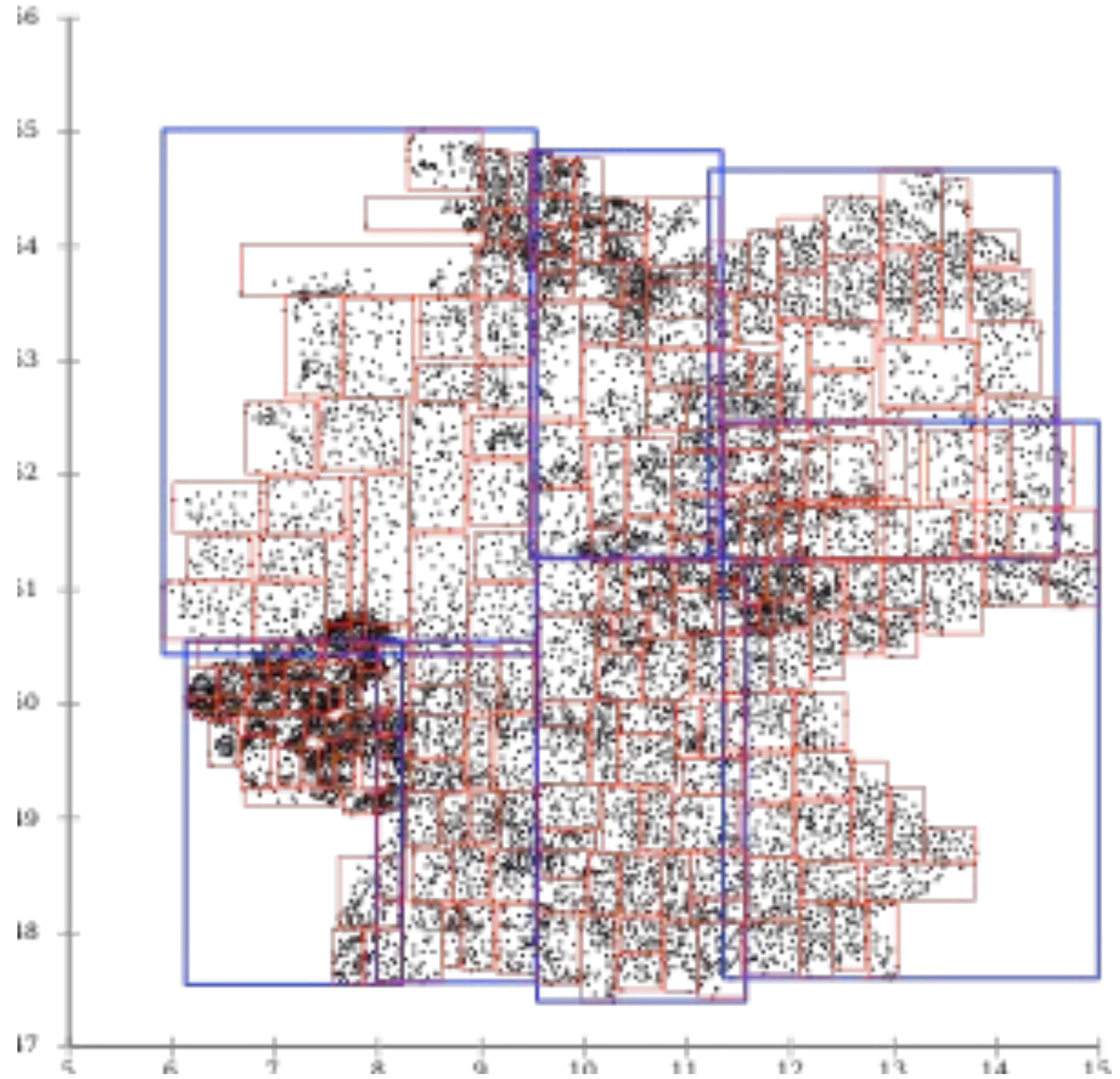




PedidosYa

Árboles - R*-trees

Los R*-trees son un tipo de árbol especializado en almacenar agrupaciones de puntos (a través de bounding boxes como nodos). Hay muchas variantes del mismo, pero generalmente la búsqueda se da por un punto o bounding box



Árboles - golang

Interval tree:

- github.com/lggomez/intree: Fork de geozelot/intree (implementación de interval tree estático) con mayor test coverage y soporte de hojas con elementos custom

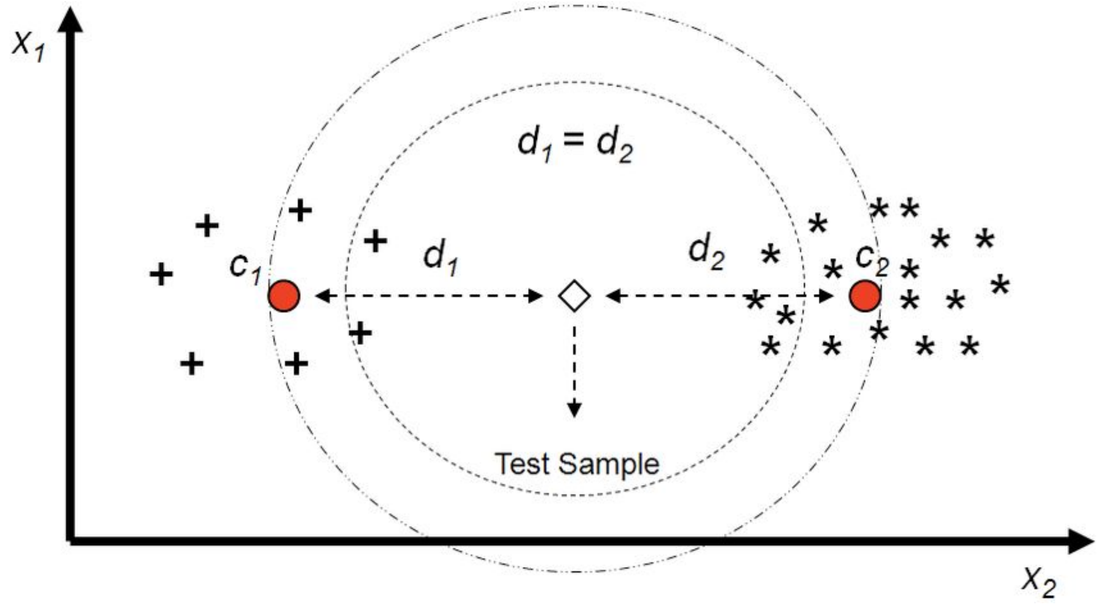
R*-tree:

- github.com/dhconnelly/rtreego: Implementación de R*-tree n-dimensional eficiente, de API flexible y basada en los papers originales de dichas estructuras y algoritmos de búsqueda

PedidosYa

Búsqueda por distancia (o cercanía)

Lo importante de los R*-trees es que nos permite implementar una búsqueda de los *k* resultados más cercanos dado un radio de distancia *d*. Esta familia de algoritmos se llama kNN (k Nearest Neighbors)



kNN - golang

- TODO (continuará)

PedidosYa

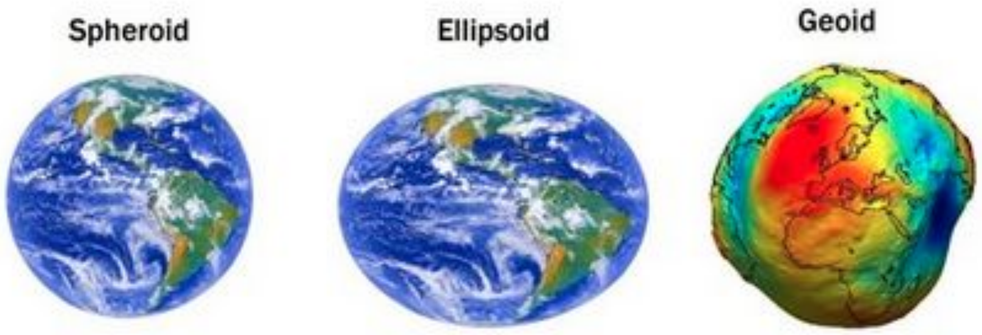
Un poco de ciencia

Pensando en grande (el globo terraqueo)

Distancias en el globo terráqueo

Calcular distancias entre coordenadas geográficas es un poco más difícil que en el plano:

1. La distancia tradicional (euclídea) no aplica dada la curvatura del globo
2. La tierra aproximada es un esfera imperfecta (elipsoide)
3. La tierra perfecta es un tipo especial de elipsoide llamado geoide. En la práctica se modelan elipsoides con constantes ya conocidas y precalculadas como el estándar WGS-84



PedidosYa

Distancias entre coordenadas

Para calcular distancias entre coordenadas vamos a utilizar, sobre WGS-84, las fórmulas de Vincenty, un método numérico iterativo de convergencia rápida y precisión arbitraria:

$$\sin\sigma = \sqrt{(\cos U_2 \sin\lambda)^2 + (\cos U_1 \sin U_2 - \sin U_1 \cos U_2 \cos\lambda)^2}$$

$$\cos\sigma = \sin U_1 \sin U_2 + \cos U_1 \cos U_2 \cos\lambda$$

$$\sigma = \operatorname{arctan2}(\sin\sigma, \cos\sigma)\text{[1]}$$

$$\sin\alpha = \frac{\cos U_1 \cos U_2 \sin\lambda}{\sin\sigma}\text{[2]}$$

$$\cos(2\sigma_\mathrm{m}) = \cos\sigma - \frac{2\sin U_1 \sin U_2}{\cos^2\alpha} = \cos\sigma - \frac{2\sin U_1 \sin U_2}{1-\sin^2\alpha}\text{[3]}$$

$$C = \frac{f}{16}\cos^2\alpha\left[4 + f\left(4 - 3\cos^2\alpha\right)\right]$$

$$\lambda = L + (1-C) f \sin\alpha \left\{\sigma + C\sin\sigma\left[\cos(2\sigma_\mathrm{m}) + C\cos\sigma\left(-1 + 2\cos^2(2\sigma_\mathrm{m})\right)\right]\right\}$$

Cuando λ converge a la precisión deseada se calculan:

$$u^2 = \cos^2\alpha\left(\frac{a^2 - b^2}{b^2}\right)$$

$$A = 1 + \frac{u^2}{16384}\left(4096 + u^2\left[-768 + u^2\left(320 - 175u^2\right)\right]\right)$$

$$B = \frac{u^2}{1024}\left(256 + u^2\left[-128 + u^2\left(74 - 47u^2\right)\right]\right)$$

$$\Delta\sigma = B\sin\sigma\left\{\cos(2\sigma_\mathrm{m}) + \frac{1}{4}B\left(\cos\sigma\left[-1 + 2\cos^2(2\sigma_\mathrm{m})\right] - \frac{B}{6}\cos[2\sigma_\mathrm{m}]\left[-3 + 4\sin^2\sigma\right]\left[-3 + 4\cos^2(2\sigma_\mathrm{m})\right]\right)\right\}$$

$$s = bA(\sigma - \Delta\sigma)$$

$$\alpha_1 = \operatorname{arctan2}(\cos U_2 \sin\lambda, \cos U_1 \sin U_2 - \sin U_1 \cos U_2 \cos\lambda)$$

$$\alpha_2 = \operatorname{arctan2}(\cos U_1 \sin\lambda, -\sin U_1 \cos U_2 + \cos U_1 \sin U_2 \cos\lambda)$$

PedidosYa

(ejercicio al/la lector@: implementarlo en su lenguaje de programación favorit@)

Golang como lenguaje para código matemático

El soporte de caracteres extendidos en los identificadores de variables en go permite expresar expresiones complejas...

$$\operatorname{hav}(\theta) = \operatorname{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\operatorname{hav}(\lambda_2 - \lambda_1)$$

$$\operatorname{hav}(\theta) = \operatorname{hav}(\varphi_2 - \varphi_1) + (1 - \operatorname{hav}(\varphi_1 - \varphi_2) - \operatorname{hav}(\varphi_1 + \varphi_2)) \cdot \operatorname{hav}(\lambda_2 - \lambda_1)$$

$$d = 2r \arcsin\left(\sqrt{\operatorname{hav}(\varphi_2 - \varphi_1) + (1 - \operatorname{hav}(\varphi_1 - \varphi_2) - \operatorname{hav}(\varphi_1 + \varphi_2)) \cdot \operatorname{hav}(\lambda_2 - \lambda_1)}\right)$$

$$= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \left(1 - \sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) - \sin^2\left(\frac{\varphi_2 + \varphi_1}{2}\right)\right) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Distancia de Haversine

PedidosYa

Golang como lenguaje para código matemático

...en código equivalente más versátil

```
φ1 := p1.LatRadians()
φ2 := p2.LatRadians()

λ1 := p1.LonRadians()
λ2 := p2.LonRadians()

latHalfVersine := math.Sin((φ2 - φ1) / 2)
lonHalfVersine := math.Sin((λ2 - λ1) / 2)

h := math.Sqrt((latHalfVersine * latHalfVersine) +
        (lonHalfVersine * lonHalfVersine) *
                math.Cos(φ1)*math.Cos(φ2))

if h > 1 {
        // d is only real for 0<=h<=1
        return math.NaN()
}

// Main inverse haversine formula
return 2 * ellipsoids.WGS84_MEAN_RADIUS * math.Asin(h)
```

PedidosYa

Geodesia - golang

github.com/lggomez/go-geodesy

Después de relevar no se encontraron packages con implementaciones open source de utilidades de geodesia en golang, por lo que surge este package:

- Soporte nativo, simple de puntos geográficos y antípodas exactas (grados y radianes)
- Distancias de Vincenty inversa y Haversine (basadas en WGS-84)
- Constantes de elipsoides GRS-80 y WGS-84 listas para usar

PedidosYa

Repasando

Después de la fase de investigación previa, ya tenemos todas las herramientas que vamos a utilizar:

- Algoritmos
  - Packages 👍
- Estructuras de datos
  - Packages 👍

Y ahora podemos continuar con la implementación principal

PedidosYa

Locations v2

Armando y presentando lo nuevo

Arquitectura

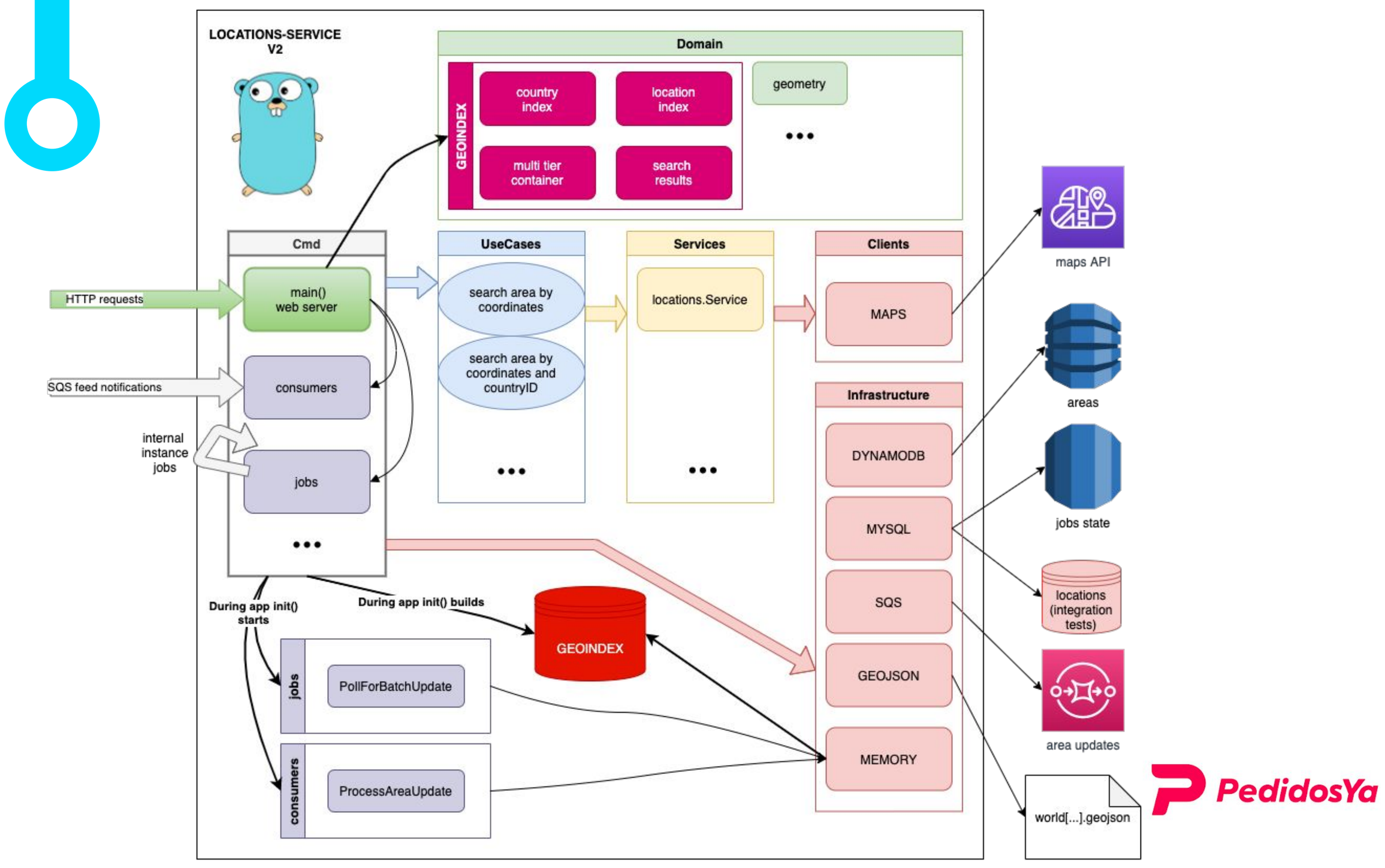
Aplicativa:

- Base datos en memoria estática expuesta vía API
- Múltiples índices inversos para optimizar las búsquedas de área
- Carga de polígonos de país en tiempo de inicialización

Datos:

- Migración de datos y formato de coordenadas a Dynamodb, con escucha de novedades de SQS
- Motor de datos basado en scheduler interno que invalida y refresca la base de datos en memoria via hot swap (contencion negligible)
- Política reactiva mixta de tiempo y cantidad de cambios entrantes, configurable

PedidosYa


LOCATIONS-SERVICE
V2
Domain
GEOINDEX
country index
location index
multi tier container
search results
geometry
Cmd
main()
web server
HTTP requests
consumers
SQS feed notifications
internal instance jobs
jobs
UseCases
search area by coordinates
search area by coordinates and countryID
Services
locations.Service
Clients
MAPS
Infrastructure
DYNAMODB
MYSQL
SQS
GEOJSON
MEMORY
maps API
areas
jobs state
locations (integration tests)
area updates
world[...].geojson
During app init() starts
During app init() builds
GEOINDEX
jobs
PollForBatchUpdate
consumers
ProcessAreaUpdate
PedidosYa

Indexado - Implementación

Geoindice por indices inversos; cambian los puntos de entrada, las instancias no se duplican (solo se copian punteros de Location):

```
// Tier #0 - Country polygons
// This tier is used to validate the country and obtain its ID given a coordinate
countryTier *treeIndex

// Inverted index #1
// Nested tiers - country interval tree (by bbox latitudes)
//              - leaf nodes constitute sub interval trees (by area polygon bbox latitudes)
locationAreaTier map[int]*treeIndex

// Inverted index #2
// Nested tiers - r*-trees of area polygon bboxes by countryID
locationRTier map[int]*rtreego.Rtree

// Inverted index #3
// Separate tier - by location/area ID
locationIDTier map[int64]*location.Location
```

PedidosYa

Indexado - Países (countryTier)

- Se indexan los polígonos de país donde opera PedidosYa para la búsqueda de country_id
- Se arma un interval tree a partir de las latitudes de las bounding boxes de cada país

PedidosYa

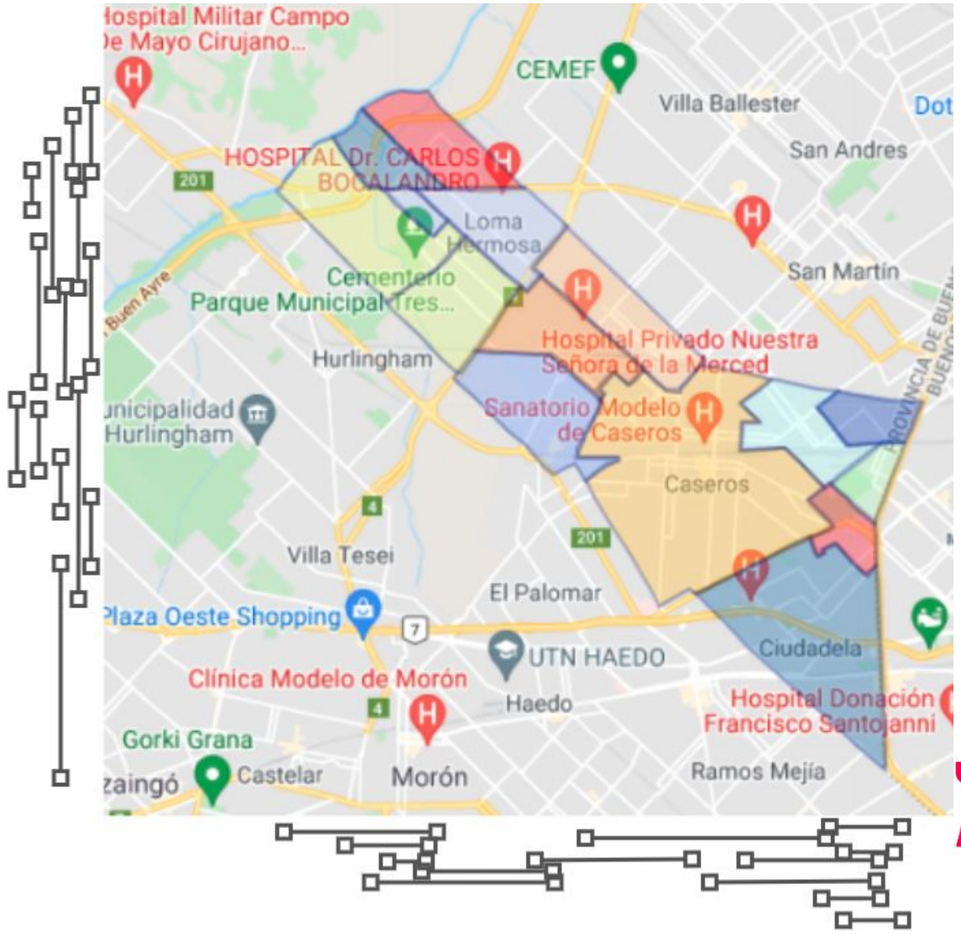
Indexado - Ventajas en go

- Localidad en memoria: las coordenadas son [][2]float64, un slice de arrays (flexibilidad + performance)
- Overhead bajo de GC: en pruebas de stress, las pausas STW afectan en p99 de los requests manteniendo estables el resto de los tiempos
- Estabilidad: Los tiempos de acceso y respuesta de la aplicación son muy estables (SD extremadamente baja)

PedidosYa

Indexado - Por qué latitudes?

Experimentalmente notamos mayor agrupación de intervalos de bounding boxes por longitud, por lo cual indexar por latitud incrementa la performance de acceso al índice en un 20% (mayor ganancia en Argentina)



PedidosYa

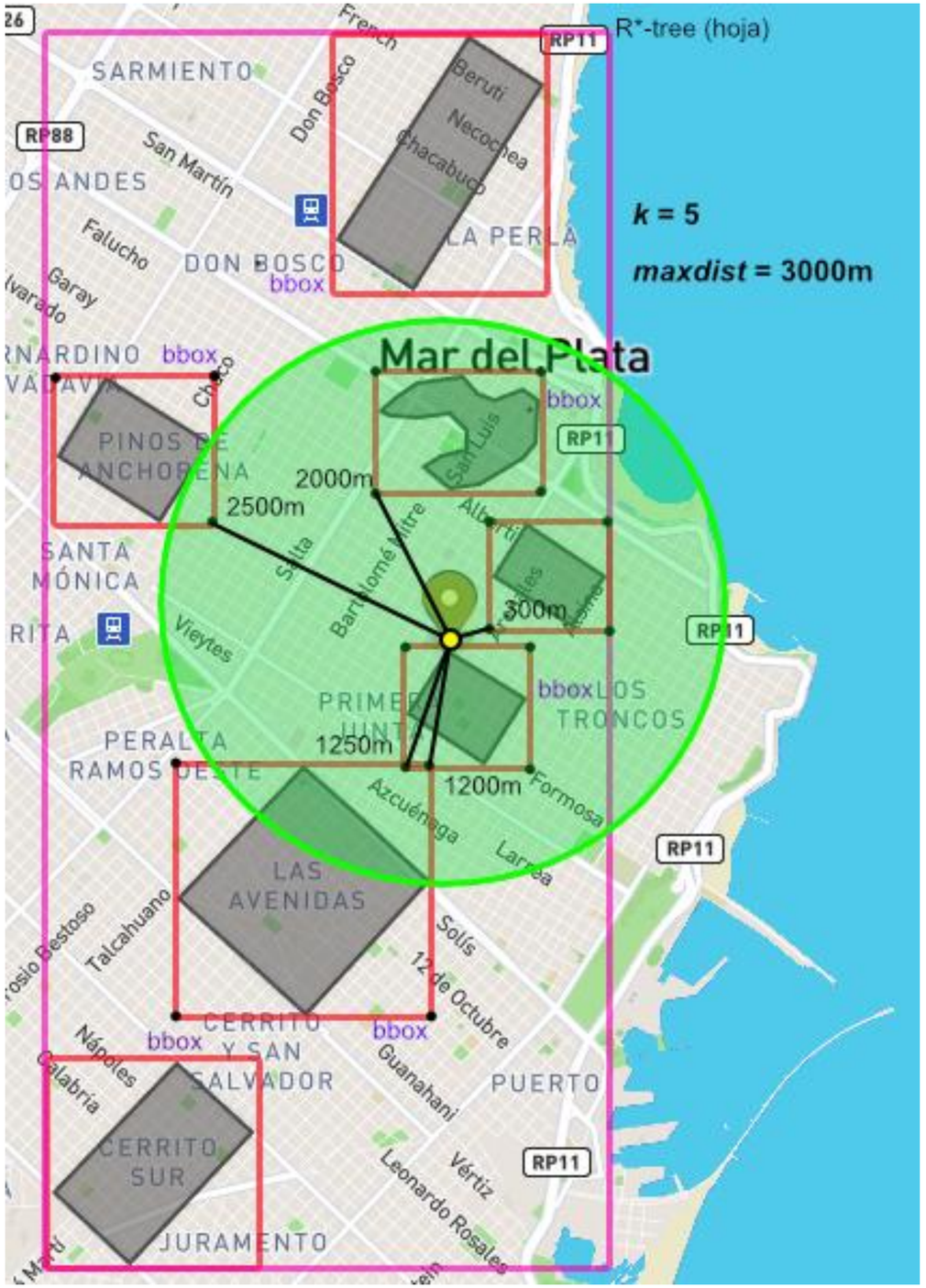
Búsqueda - Algoritmo kNN híbrido

1. Se busca en el tier 0 (si no hay country_id, sino el paso inicial es *2*)
2. Se busca en el tier 1 (interval tree)
   a. Si hay resultado(s), se devuelve el match, *sino (o si no hay match)*
3. Se realiza la búsqueda por cercanía (KNN):
   a. Búsqueda en tier 2 (r*-tree) de *k* resultados más cercanos
   b. Mediante una heurística + un distancia máxima *d* (x Vincenty), se filtran los polígonos y se devuelve el resultado final

PedidosYa

Búsqueda

Heurística (usando bounding boxes)

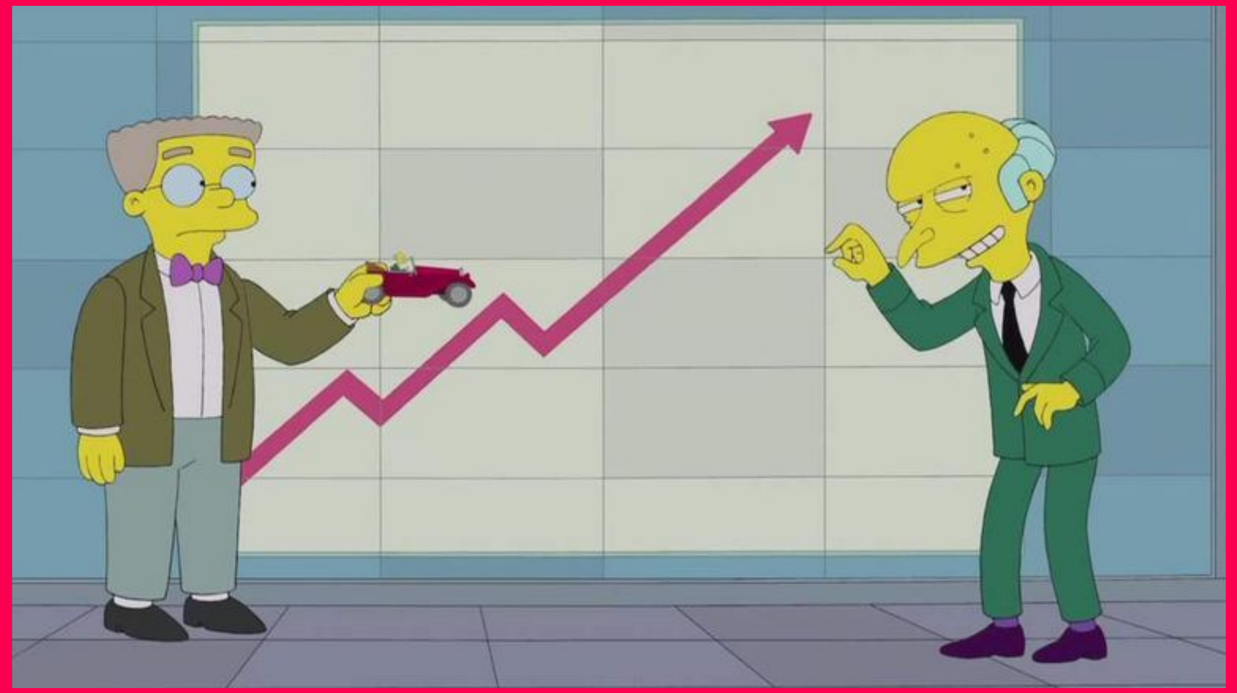


Búsqueda

Heurística (usando puntos de polígono)

Resultados

Qué logramos con todo esto?



Algunos números - Grails vs Golang & Arq v2

Performance

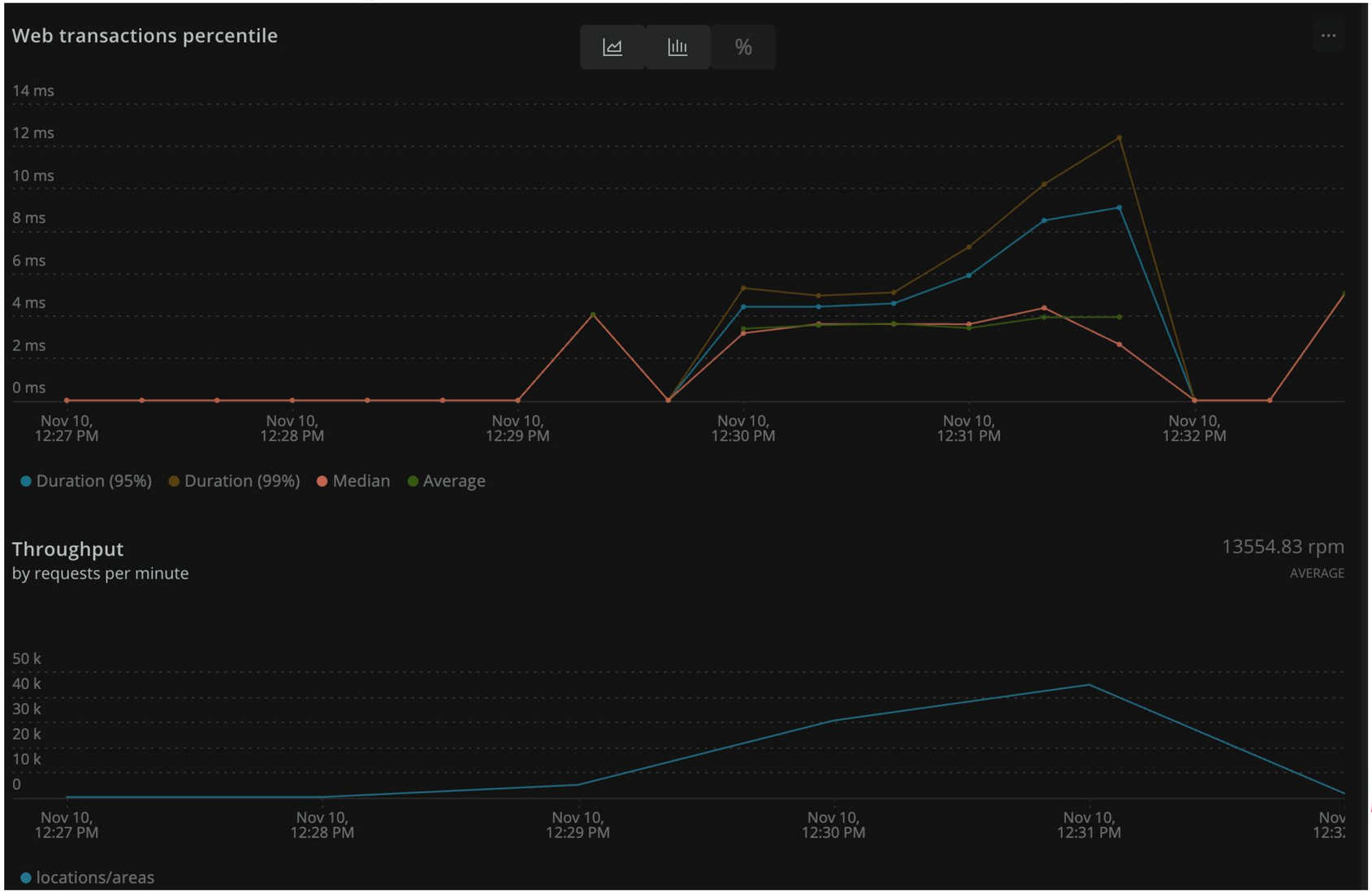
- 75-80% de reducción en tiempos promedio de respuesta
- 85% (cota inferior) de reducción en tiempos en percentiles
- 99,5% de reducción de uso de memoria RAM

Negocio

- 90% de reducción en casos de área no encontrada

PedidosYa

Pruebas de carga - Antes (v1, con country_id siempre)



PedidosYa

Pruebas de carga - Ahora (v2, escenarios mixtos)

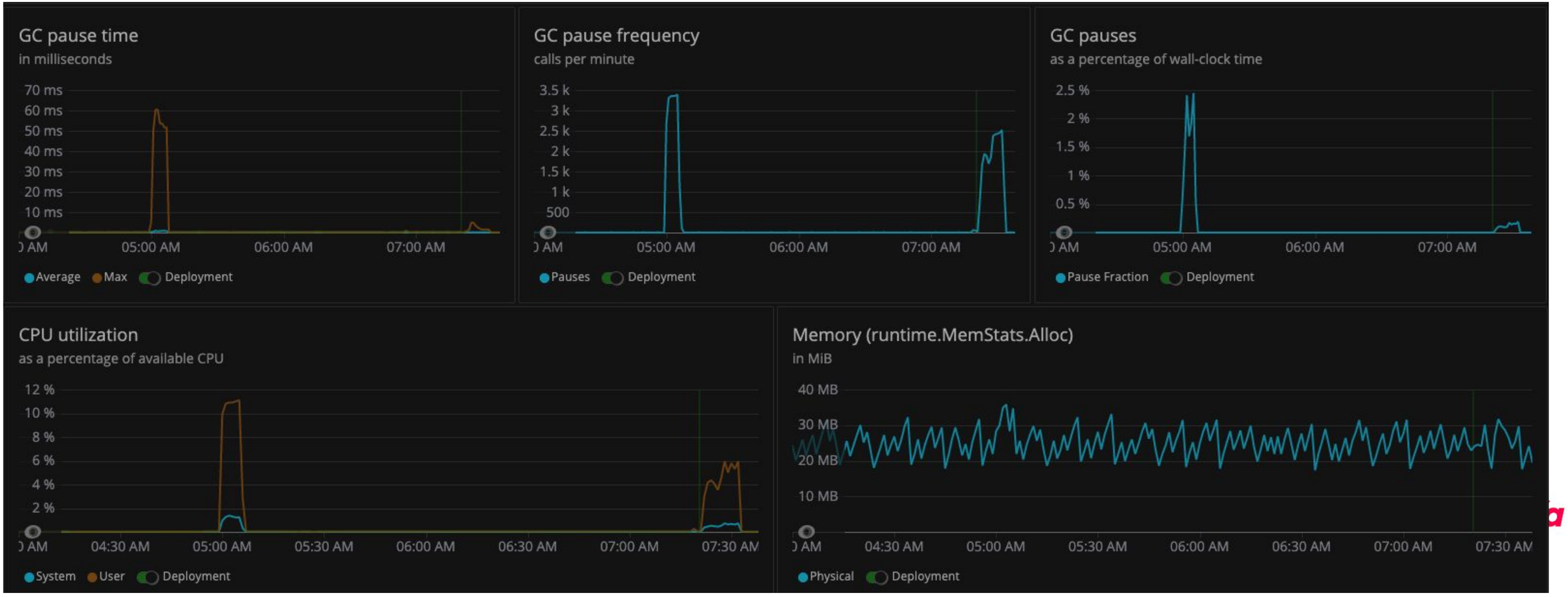
p99: requests entrantes en pausa de GC (en instancia tipo tiny)

PedidosYa

Pruebas de carga - Geoíndice (2 instancias tiny, go 1.16)

Pruebas de carga - Geoíndice (2 instancias tiny, go 1.16)

Pruebas de carga - Geoíndice (cont.)



Preguntas?

¡Muchas gracias!

luisgg.me

@PeYaTech

PedidosYa Tech