Efficient Geospatial Queries with Go

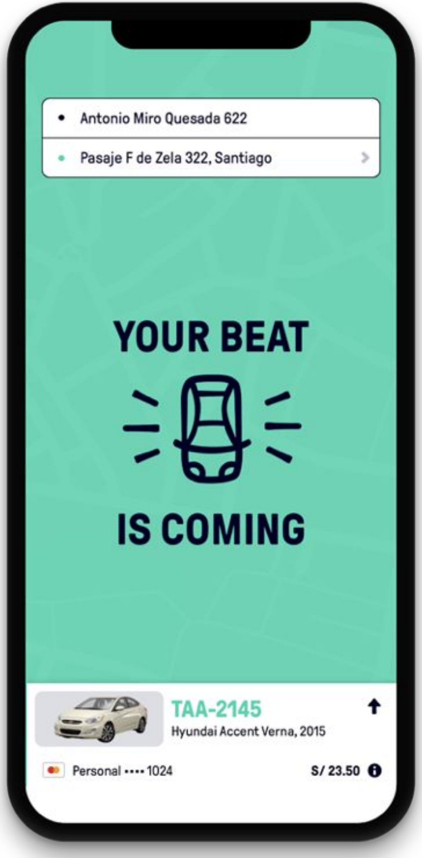
Fotis Papadopoulos
Senior Backend Engineer

BEAT

WHAT IS BEAT

The fastest growing ride-hailing app in Latin America

- Connects passengers & drivers, real-time, 24/7
- Our Mission: Affordable, Fast, Safe and Reliable Transportation
- Part of the FREE NOW group, the ride-hailing joint venture of BMW and Daimler
- Operates in 23 cities (Peru, Chile, Colombia, Mexico, Argentina, Greece)
- HQs in Athens
- Beat Engineering Hub in Amsterdam



BEAT

14.000.000

Active Passengers

487.000

Active Drivers

Spatial Indexing

Optimizing location queries

BEAT

Spatial Indexing

Spatial Indexing allows for fast and efficient location queries on large datasets without sequential scan.

Typical spatial operations:

- **Spatial Range Query**: find spatial objects within a specified range from a given object
- **Spatial Join**: find objects that spatially interact with each other (intersection, containment, etc)

Containment is a common spatial index operation

BEAT

Spatial Indexing

The puzzle Beat has to solve

Given a passenger's location:

- How many drivers are there in an X km radius around the passenger?
- How can the search be cost & resource effective?

Given a driver's location:

- How can we have the driver's latest position during passengers' requests?

Scale the above to **~45K connected drivers** and **passengers** during high-demand hours.

BEAT

Initial Approach

MongoDB & Lessons Learned

Initial Approach

MongoDB and why it’s good for geospatial queries

Reasons to use MongoDB:

- `Quick setup`
- `Geospatial indexing with built-in` **`2dsphere`** `index`
- `Client libraries for most mainstream programming languages`

BEAT

Initial Approach

Why MongoDB failed our expectations

Reasons not to use MongoDB:

- 2dsphere index does not support **sharding**
- Geospatial indexing is **expensive**
- Scales only vertically
- Beat's PHP client lack of connection "pooling" support
- Replica set lag & network noise
- 2dsphere index can be slow on dense datasets

BEAT

A Better Approach

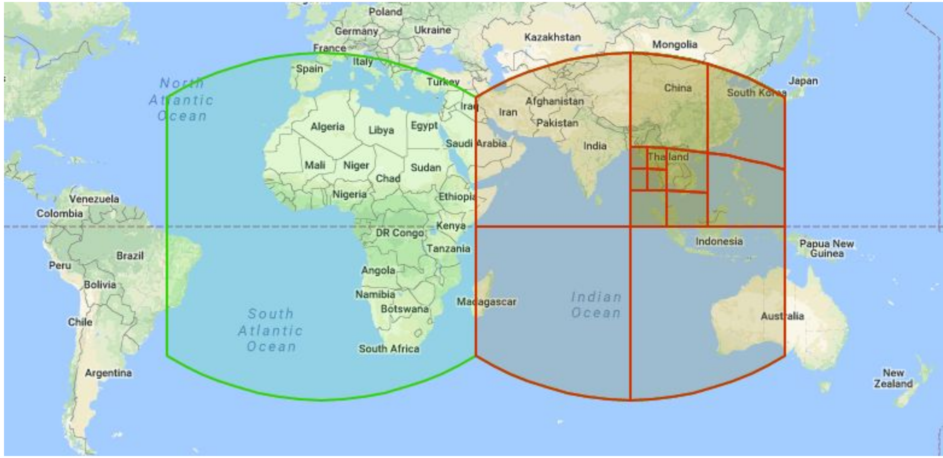
S2 Geometry + Go-MemDB = BFFs

Google is your friend

Standing on the shoulders of giants

Introducing S2 spherical geometry

- A framework for decomposing the unit sphere into a hierarchy of cells
- Represents data on a 3D sphere
- Low distortion compared to the actual shape of the earth
- Robust constructive (unions, intersections) & boolean predicate (containment) operations
- Full C++ support - partial ports in Go(40%), Java, Python

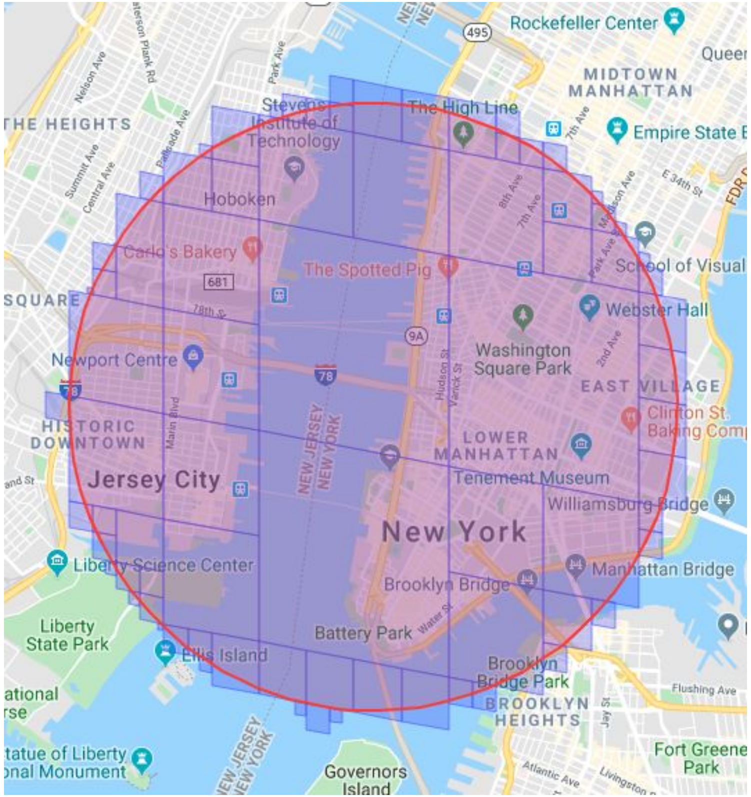


BEAT

Google S2 Cells

The core of the S2 spherical geometry

- 31 hierarchy levels [0 to 30]
- “Cell” area ranges from .74 cm^2 (leafs) to approx. 85 million km^2 (faces)
- Cell IDs are conveniently stored as uint64
- Efficient region coverage (cell unions) - number of cells vs detail tradeoff
- Cells are indexed using Hilbert Curves



Region Covering (levels 4 to 20)

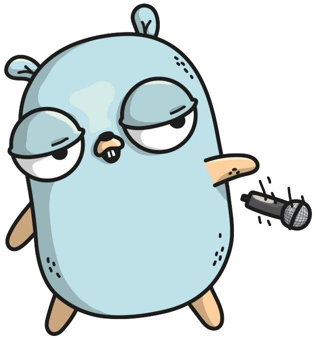
S2 Cell Statistics
S2 coverage demo
The S2 Space-Filling Curve

Hashicorp's Go MemDB

Fast & reliable in-memory storage, courtesy of Hashicorp

Simple in-memory database

- Built on immutable radix trees
- Provides **Atomicity**, **Consistency**, **Isolation** from ACID
- Lacks **durability**
- Multi-Version Concurrency Control (MVCC)
- Transaction support
- Rich **indexing**, a single object can be indexed in multiple ways

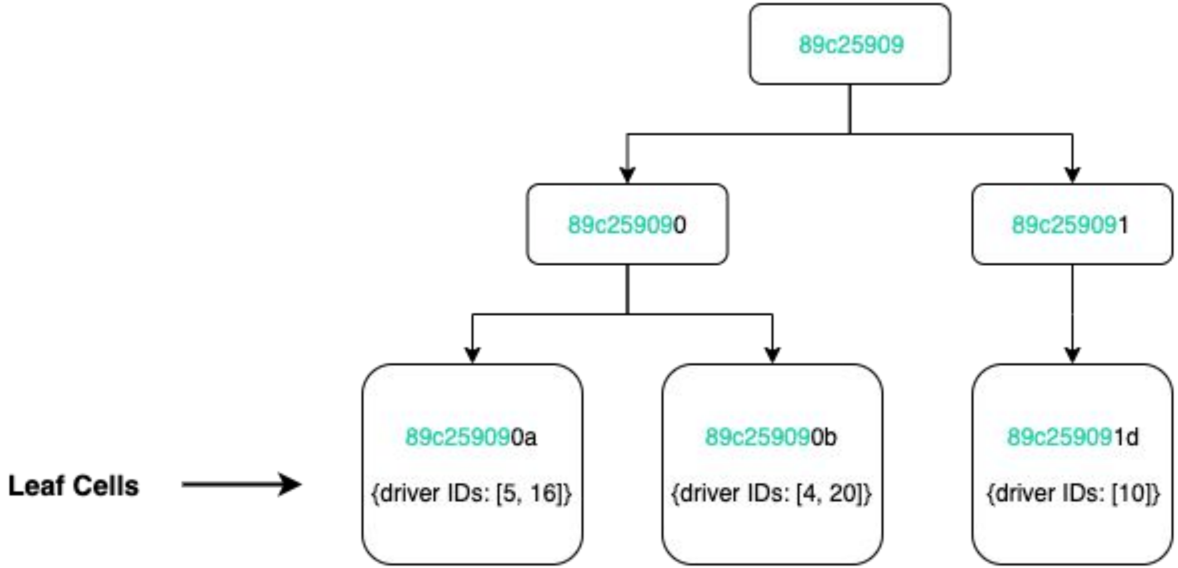


Hashicorp's Go MemDB

BEAT

Prefix Search

Turning spatial indexing into a simple prefix search



Example (1/2)

Location Insert Operation

```
func (m *memdb.MemDB) insert(ID int, lat float64, lon float64) error {
	// Initialize write transaction.
	txn := m.Txn(true)
	err := txn.Insert(table, &user{
		ID:     ID,
		CellID: s2.CellIDFromLatLng(s2.LatLngFromDegrees(lat, lon)).String(),
	})
	// Abort transaction on error.
	if err != nil {
		txn.Abort()
		return err
	}
	// Commit changes in order to complete the transaction.
	txn.Commit()
	return nil
}
```

BEAT

Example (2/2)

Near Search Operation

```
func (m *memdb.MemDB) Near(lat float64, lon float64, radius int, limit int) ([]user, error) {
	// Define an S2 Region on the sphere using the provided point and radius.
	// This can be done in several ways, one of which is defining a spherical cap.
	r := s2.Region(sphereCap)
	// Get the CellUnion representing the specified area.
	covering := s2.coverer.Covering(r)
	// Find users in each of the cells in a single transaction.
	txn := s.db.Txn(false)
	for _, cell := range covering {
		// MemDB supports prefix indexing without explicit definition.
		iter, err := txn.Get(table, cellIdx+"_prefix", cell.String())
		// Process result iterator.
	}
	return processed, nil
}
```

A working example can be found here and here

BEAT

Performance Comparison

Turns out, the impact on latency was huge!

Nearest Search Latency	p50	p99
MongoDB	110ms	400ms
S2 & Go-MemDB	2.6ms	9ms

Insert/Update Latency	p50	p99
MongoDB	220ms	600ms
S2 & Go-MemDB	2.5ms	4.97ms

BEAT

Before you ask

S2 & Go look cool, but did you...

Q: Try MongoDB in-memory storage engine?

A: Yes,we did, but the actual bottleneck of the process is spatial indexing, not disk I/O.

Q: Consider another off-the-shelf DB, like PostgreSQL with PostGIS?

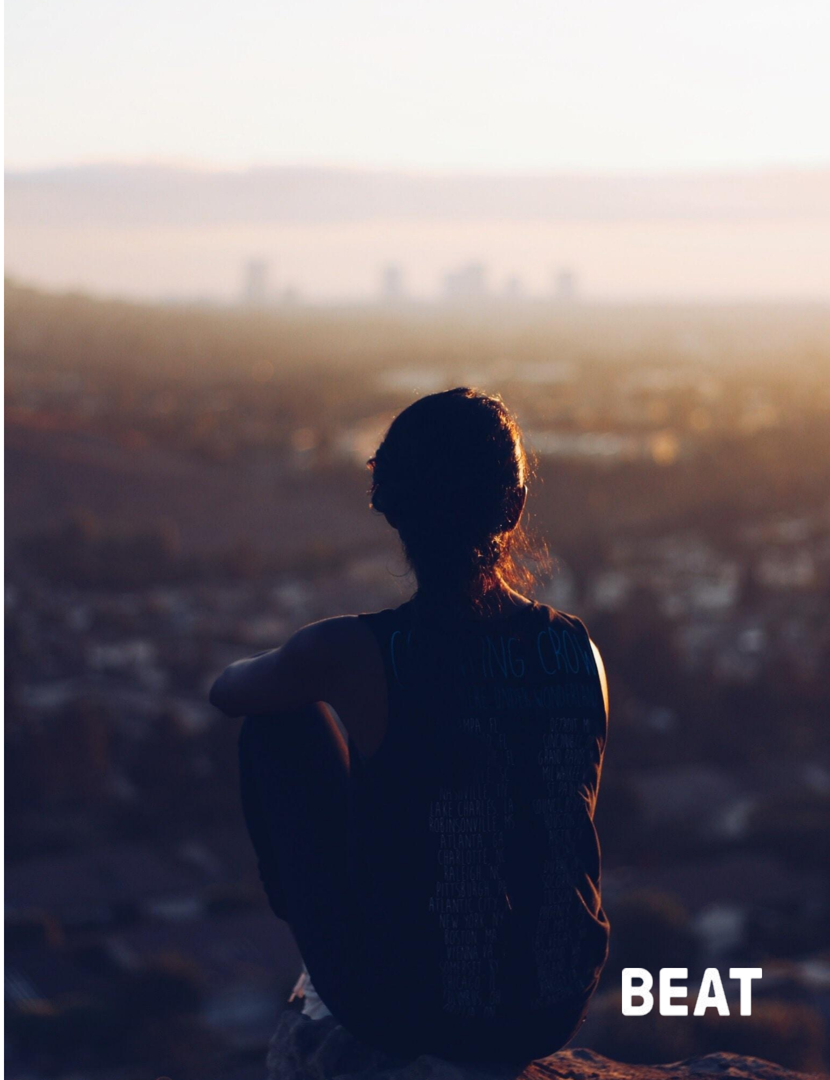
A: Yes, we did, but nothing can beat in-memory storage performance.

BEAT

Considerations

A couple more things to bear in mind

- Data Consistency - in-memory data needs to be manually replicated among service instances (handled w/ Kafka messaging)
- Increased service setup time
- MongoDB has much easier setup process plus there is no need to maintain code in order to combine S2 with Go MemDB



Key Takeaways

What makes us proud of this custom database

- MemDB has a very high write throughput (more than 15K writes/sec observed)
- Cost & resource effectiveness
- Largest Beat market - 3 pods with **2vCPUs** and less than **200MB** of RAM each

WE ARE HIRING

beat.careers

THANK YOU!

Fotis Papadopoulos

Senior Backend Engineer

linkedin.com/in/fpapadopou

f.papadopoulos@thebeat.co

BEAT