Geocaching with Geohashing

Scaling weather APIs for Big Data Machine Learning

Alexander Kalmikov, PhD
Senior Data Scientist

QUANTUMBLACK
A MCKINSEY COMPANY

Confidential and proprietary: Any use of this material without specific permission of McKinsey & Company is strictly prohibited

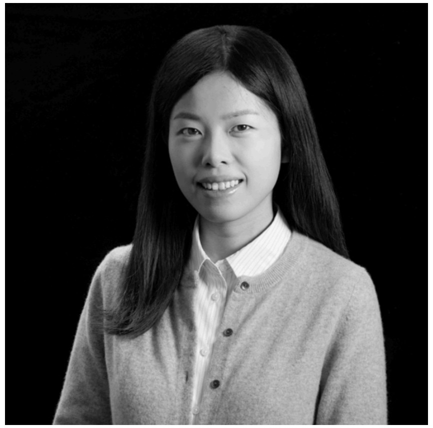
Team

Alexander Kalmikov

Senior Data Scientist

Yuhao Zhu

Data Engineer



Lingyi Zhang

Data Engineer

Jeffery Annor

Data Engineer

All content copyright © 2019 QuantumBlack, a McKinsey company

Agenda

01 Background - Geospatial Big Data
02 Geohash Algorithm
03 Caching with Geohash
04 Conclusions
05 Q&A

Geospatial Big Data

01

Geospatial Big Data

Objective: large-scale geospatial analytics on cloud and distributed computing systems

Latitude, longitude arrays

- NetCDF
- xarray

Location points (database records)

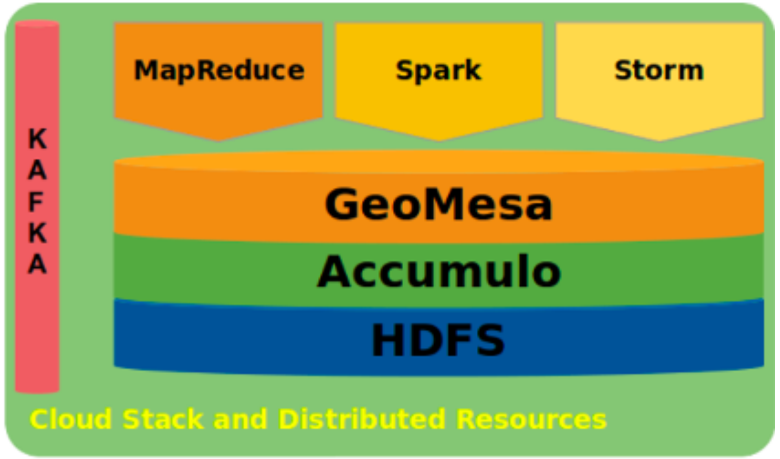
- NoSQL databases
- Low latency, massively scalable
- Hadoop Distributed File System (HDFS)

Indexing database records with Geocodes

- Location sensitive queries
- Proximity search
- Geospatial join

Geocoding - Locality preserving encoding of geographic coordinates for fast big data operations

- Distance preserving dimensionality reduction technique
- Locality-sensitive hashing (LSH), probabilistically defined – edge cases tolerated on average



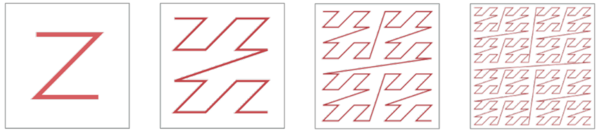
GeoMesa: Bigtable-based NoSQL database built on geohash

All content copyright © 2019 QuantumBlack, a McKinsey company

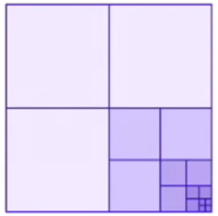
Geocoding systems

Geohash

Z-order curves (Lebesgue curves)



continuous fractal space-filling curves

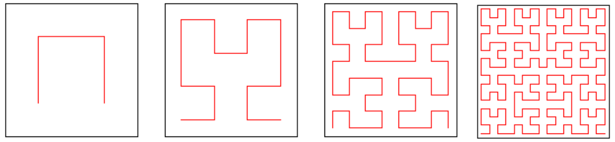


Squares

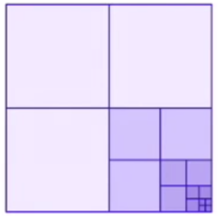
Source: Van Le, Hong. "Distributed Moving Objects Database Based on Key-Value Stores." 2016. http://ceur-ws.org/Vol-1671/paper4.pdf

Google S2

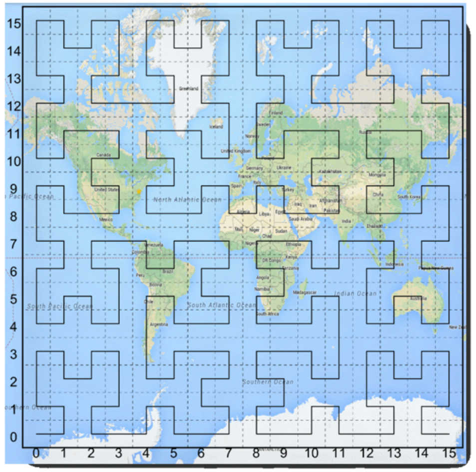
Hilbert curves



continuous fractal space-filling curves



Squares

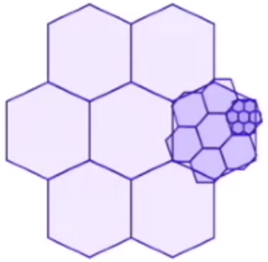


Source: GeoWave Documentation
https://locationtech.github.io/geowave/previous-versions/0.9.2.1/documentation.html

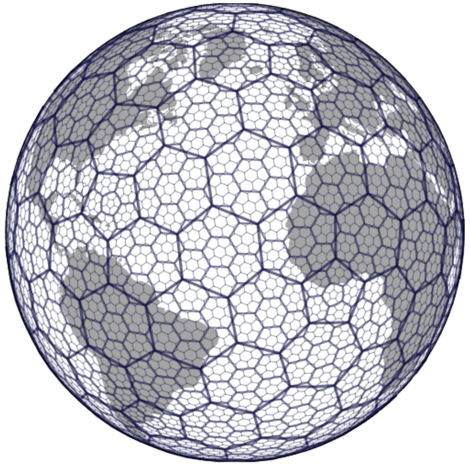
Uber H3

Hexagonal Hierarchical Spatial Index

Central Place Indexing (CPI)



Hexagons



Source: https://eng.uber.com/h3/

All content copyright © 2017 QuantumBlack, a McKinsey company

Geohash Algorithm

02

Geohash

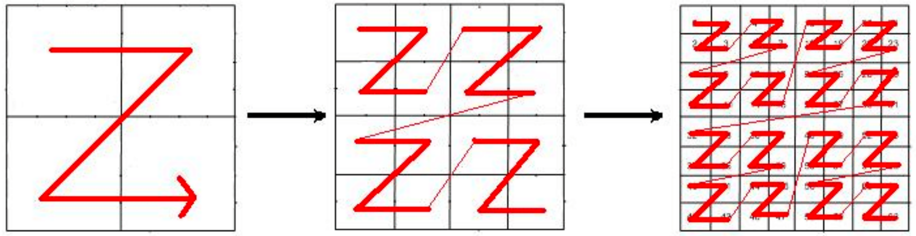
Popular public domain geocode system for hierarchical spatial gridding with one-dimensional distance preserving index

- Encodes a geographic location
- Hierarchical spatial data structure which subdivides space into nested grids
- 2D to 1D mapping with space-filling curves
- Z-order curves (Lebesgue curves, Morton curves)
- Arbitrary precision
- Gradual coarsening by removing characters from the end of the code to reduce its size (and gradually lose precision).
- Base32 encoding (alphanumeric)
- Hashing maximizes collisions, different from *cryptographic hashing*

Example - this room at 5m precision: **`drt2zkf7y`**

References:

- G. Niemeyer (2008) http://geohash.org
- G. M. Morton (1966) "A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing"

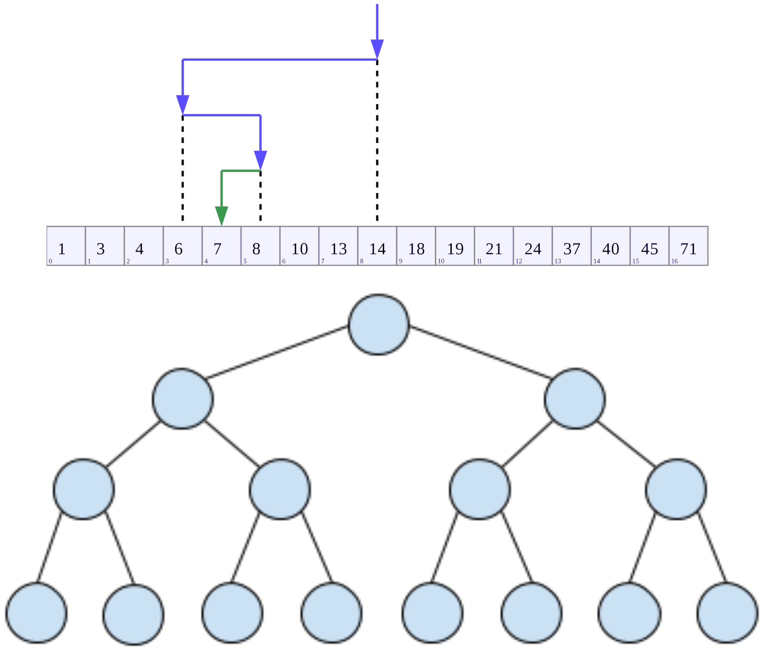


Source: https://www.movable-type.co.uk/scripts/geohash.html

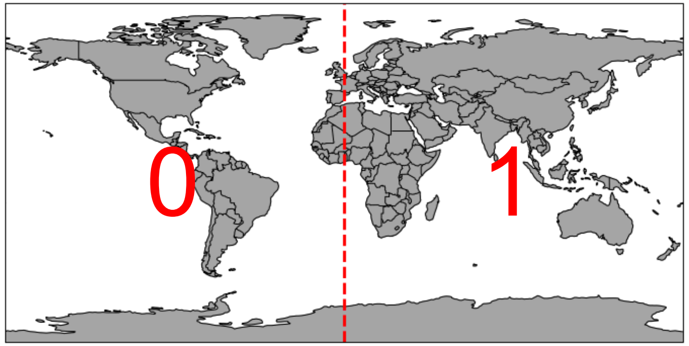
All content copyright © 2019 QuantumBlack, a McKinsey company

Geohash algorithm

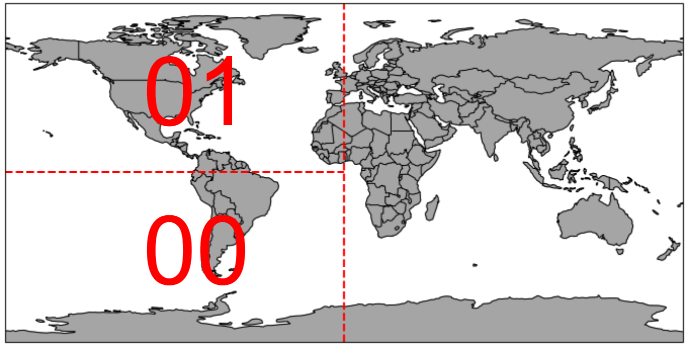
- Alternating latitude, longitude binary partitions
- Interweaving binary encoding
- Base-32 bit encoding

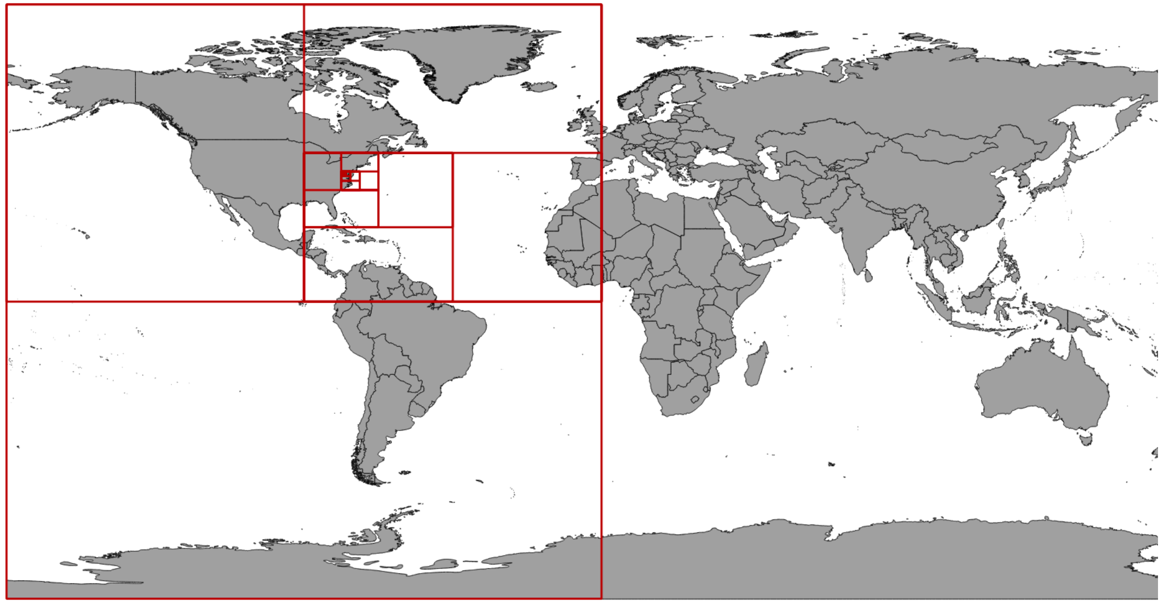


Longitude split



Latitude split





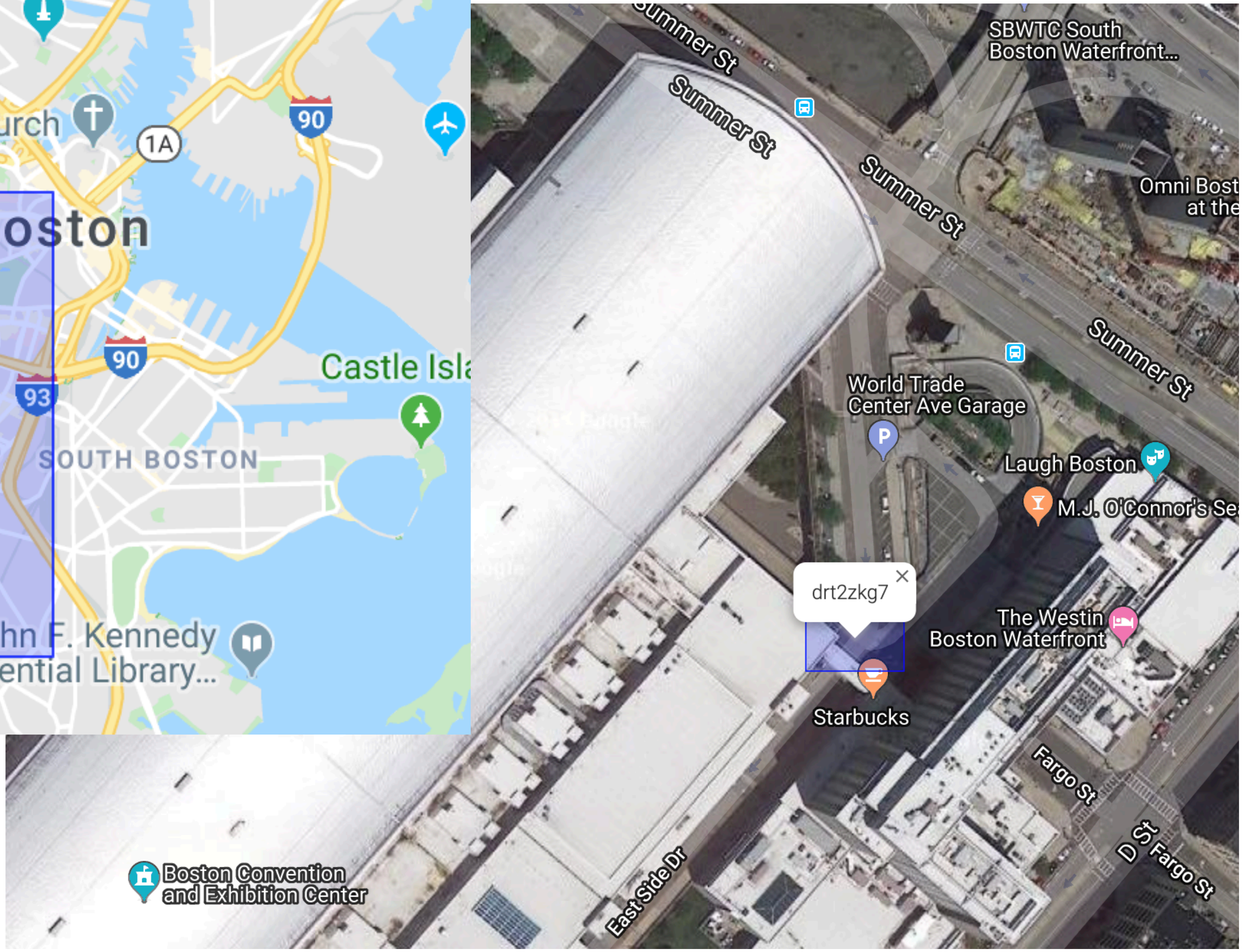
Source: https://www.geomesa.org/documentation/2.0.2/user/appendix/utils.html

Geohash: **tdr1wxyp5dn7v**

11001	01100	10111	00001	11100	11101	11110	10101	00101	01100	10100	00111	11011
t	d	r	1	w	x	y	p	5	d	n	7	v

All content copyright © 2019 QuantumBlack, a McKinsey company

Geohash Precision Levels



Level	Cell width	Cell height
1	5000km	5000km
2	1250km	625km
3	156km	156km
4	39.1km	19.5km
5	4.89km	4.89km
6	1.22km	0.61km
7	153m	153m
8	38.2m	19.1m
9	4.77m	4.77m

All content copyright © 2019 QuantumBlack, a McKinsey company

Caching with Geohash

03

Caching weather API requests

Latency of requests

- Linear in number of requests O(n)
- Doesn't scale for big data

Leverage redundancy and data similarity

- Similar weather at neighboring locations
- Slow change of weather signal
- Repeated data requests

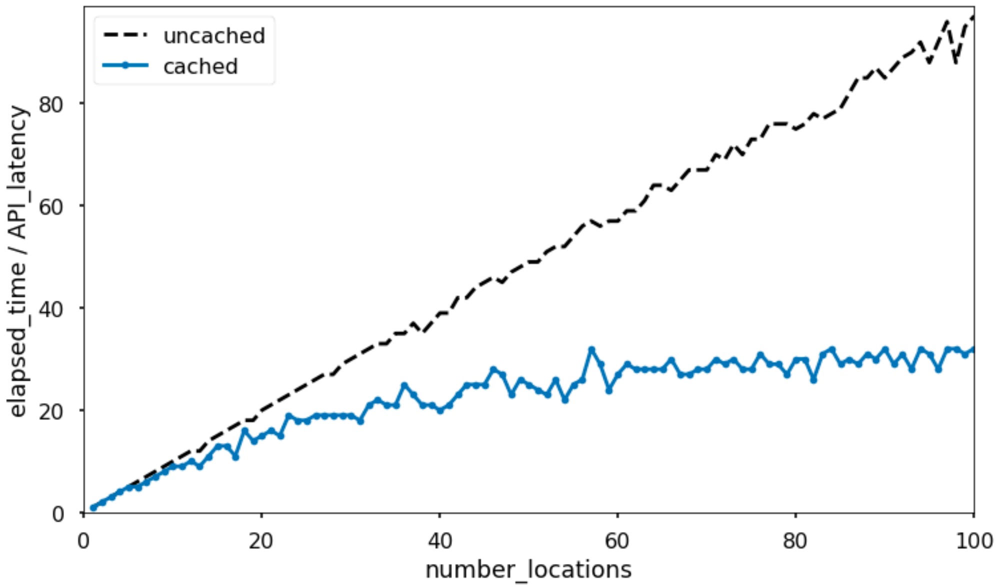
Cache - reuse previously computed values

- Requires identical function call arguments
- Need to discretize continuous location coordinates

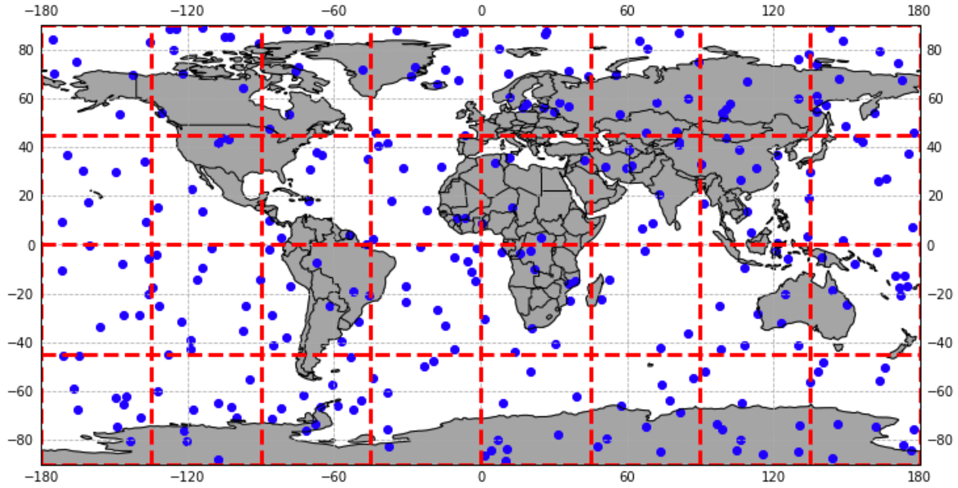
```
API_get(location)                # uncached pull

API_get_cached(geohash(location)) # cached pull
```

- Cache hits – reuse cached data (*fast*)
- Cache misses – new API access (*slow*)

Elapsed API access time (normalized)



Experiment: uniform random location samples



All content copyright © 2019 QuantumBlack, a McKinsey company

Algorithm performance

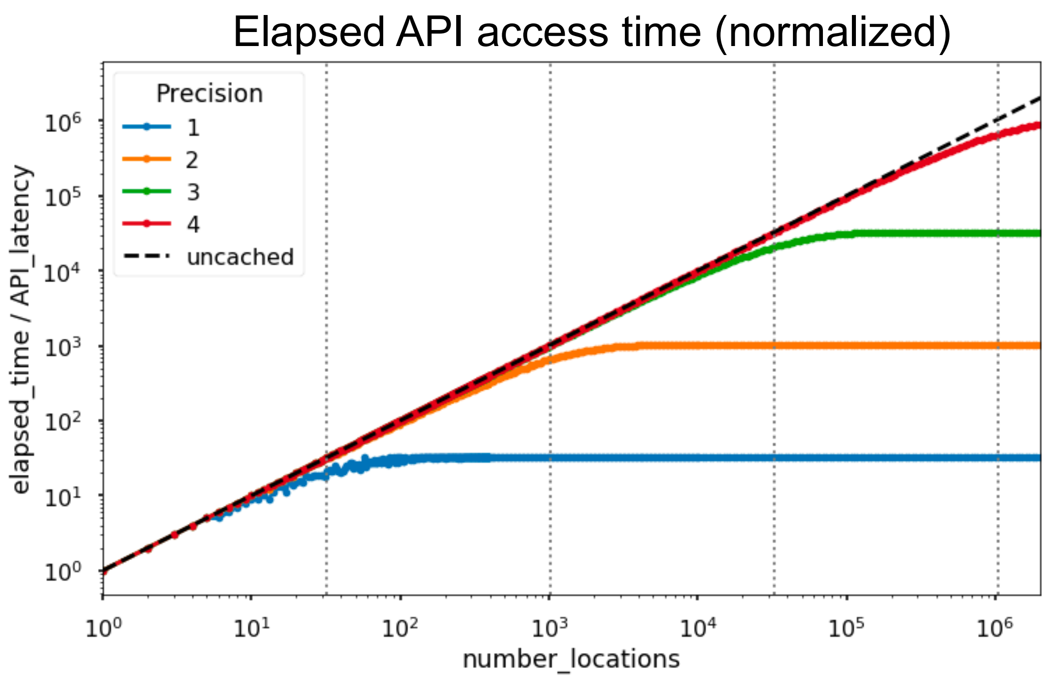
Metrics

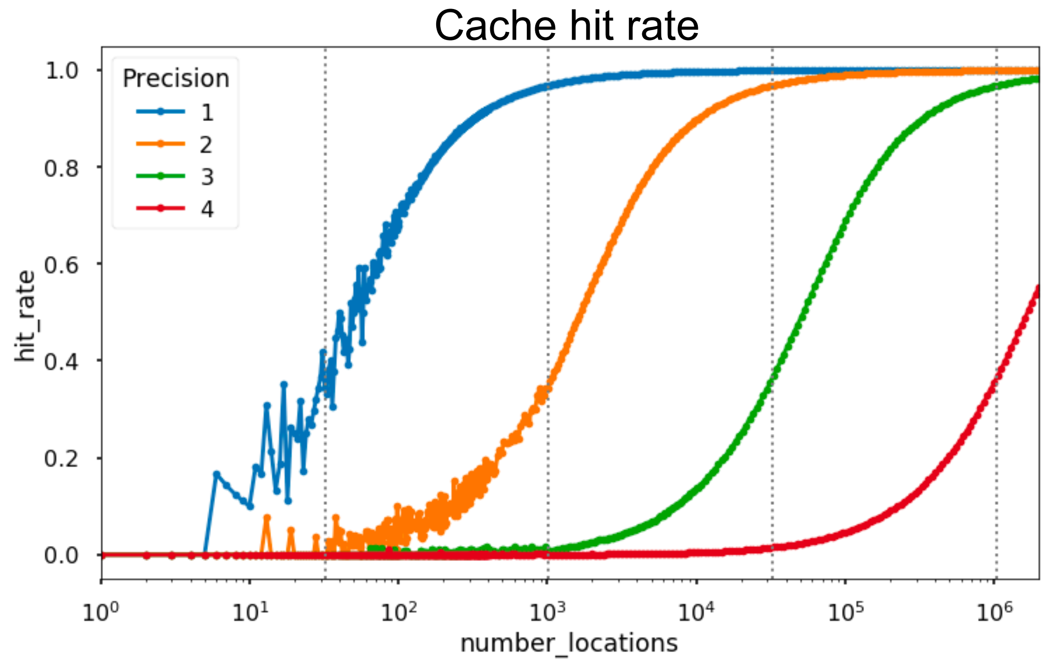
1. Elapsed API access time (normalized)
2. Cache hit rate

for different geohash precisions

precision	max cache size
1	32
2	1024
3	32768
4	1048576

- Low cache hit rate – latency of cached pulls follows uncached linear trend
- Cache hit rate saturates capping access time
  - Lower precisions saturate with smaller number of locations
  - Scale of saturation consistent with max geohash counts
- Tradeoff of precision and cached API acceleration
  - Data system architecture
  - Infer feasible spatial precision for given run time constraints





Conclusions

04

Key takeaways

Summary

- Accelerated API access with geohash and caching
- Geohash algorithm
- Empirical analysis of caching performance

Advantages

- Easy to use open source package
- Code complexity reduction
- Storage and fast operations in NoSQL databases

Impact

- Weather data integration in distributed big data infrastructure
- Enabling geospatial analytics with weather data at scale

All content copyright © 2019 QuantumBlack, a McKinsey company

Questions

05

Python Implementation

Concise Python syntax

Import packages

```python
import pygeohash as geohash
from functools import lru_cache
import requests
```

Cached API pull

```python
@lru_cache(maxsize=1024)
def pull_weather_cached(latitude, longitude, time_start, time_end, weather_variable):

    weather_url = (
        "https://hydro1.gesdisc.eosdis.nasa.gov/daac-bin/access/timeseries.cgi?"
        + "variable=GLDAS2:GLDAS_NOAH025_3H_v2.1:{}"
        + "&location=GEOM:POINT({},%20{})&startDate={}&endDate={}&type=asc2".format(
          weather_variable, longitude, latitude, time_start, time_end)
    )

    return requests.get(weather_url)
```

Geohash location encoding

```python
def geohash_location(latitude, longitude, precision=5):

    geohash_key = geohash.encode(latitude, longitude, precision=precision)

    return dict(
        key=geohash_key,
        latitude=geohash.decode_exactly(geohash_key)[0],
        longitude=geohash.decode_exactly(geohash_key)[1],
    )
```

All content copyright © 2019 QuantumBlack, a McKinsey company

Python Implementation – Using cached API pulls

```
locations.head(3)
```

	lat	lon
0	34.245170	-67.003683
1	48.903527	-105.347359
2	39.633557	-104.694833

- Loop over table of locations
- Geohash encoding
- Cached API pull

```
for _, location in locations.iterrows():

    hashed = geohash_location(location.lat, location.lon, precision=1)

    w = pull_weather_cached(hashed['latitude'],
                            hashed['longitude'],
                            time_start, time_end,
                            weather_variable="Tair_f_inst")
```

- Print state of the cache

```
pull_weather_cached.cache_info()
```

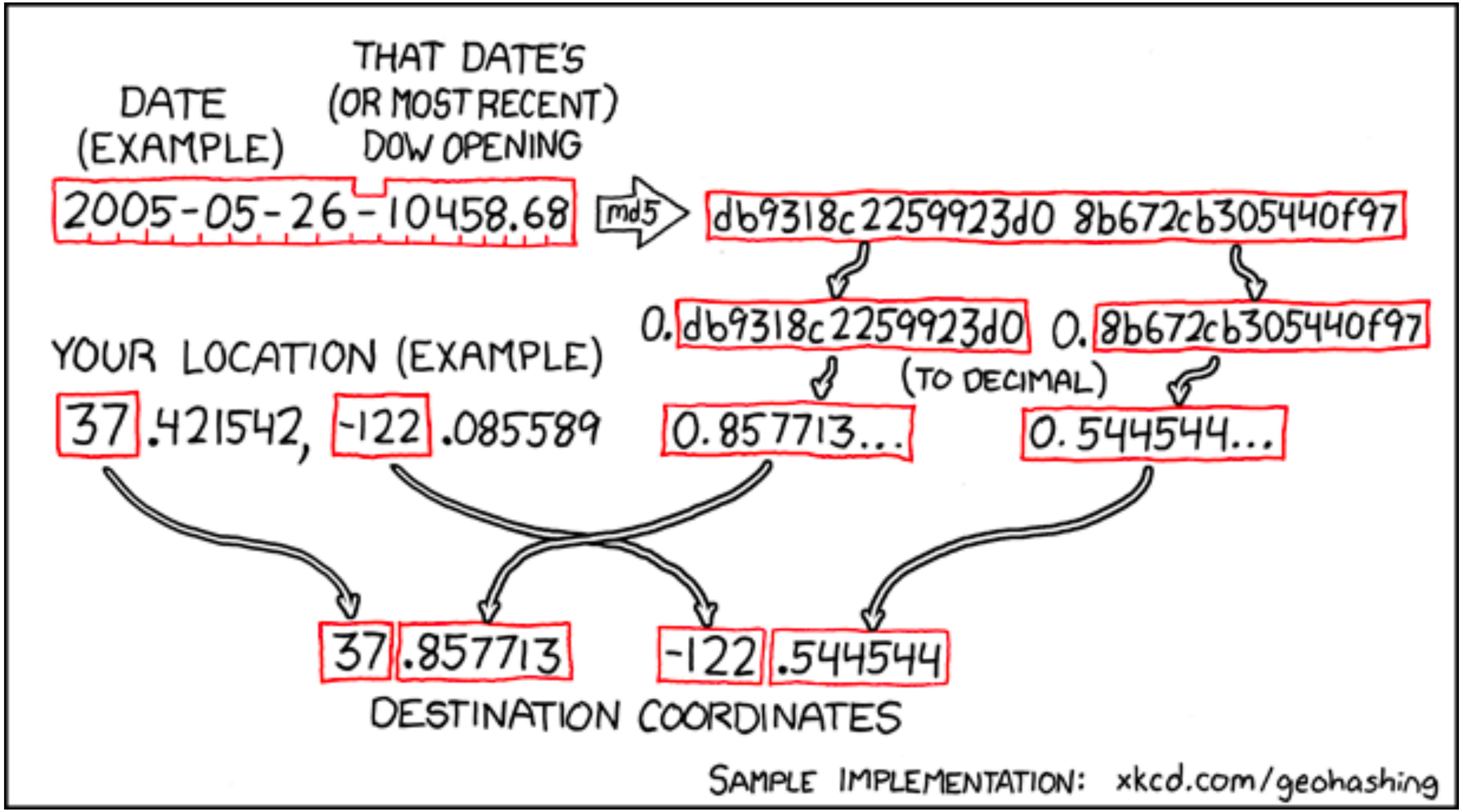
```
CacheInfo(hits=6, misses=4, maxsize=1024, currsize=4)
```

- Cache hits – reuse cached data (fast)
- Cache misses – actual API access (slow)
- Cache hit rate 60% (6 out of 10)

All content copyright © 2019 QuantumBlack, a McKinsey company

Do not confuse with ...

https://www.xkcd.com/426/



All content copyright © 2019 QuantumBlack, a McKinsey company