Case Study: Wind Sports Mashup on Google App Engine
Explaining the title

Case Study: Wind Sports Mashup on Google App Engine
Case Study: Wind Sports Mashup on Google App Engine

The problem: finding the wind

- direction
- speed
- spot
- time
Case Study: Wind Sports Mashup on Google App Engine
Agenda

Motivation, vision, and demo

Architectural overview

Problem: No cron jobs (GAE)

Challenge: Inequality filters on one property only (GAE)

Challenge: Result set <= 1000 entities (GAE)
Motivation
Motivation

http://www.dmi.dk/dmi/index/danmark/borgervejr.htm?map=map1&param=wind
Key predicate

\texttt{is\_surfable(direction, speed, spot, time)}
Problem

for s in spots:
    for h in hours:
        f = get_forecast(s, h)
        is_surfable(f, s, h)
wind sports info and logic

= A global mashup that assists practitioners of wind sports
Demo

http://welovewind.com
How to make it fly?

Serving infrastructure
Google App Engine
Google App Engine
GAE Restrictions Feb '09

Python only

Request duration $\leq 10$ seconds

Request only way to start processing

Inequality filters on one property only

...
Restrictions lifted since

Python only (Java, JRE subset)

Request duration $\leq 10$ seconds (30 seconds)

Request only way to start processing (cron jobs, however, only 20)

Inequality filters on one property only

Experimental Task Queue for offline processing
How to make it fly?

A web service for connecting all the distributed resources
Web service data model

- Weather Station
- Surf Spot
- Forecast Point
- Geo Point
- Observation
- Forecast
- Weather Data
Architecture

browser

GAE

cs.au.dk

weatherbug.com

yr.no
Architecture

GET /forecast_points/
GET /weather_stations/

browser

GAE

weatherbug.com

cs.au.dk

yr.no
GET /weatherapi/locationforecast/1.6/?lat=56.2274;lon=10.3083
Host: api.yr.no
PUT /forecast_points/56.2274,10.3083/

(JSON forecasts)
Architecture

GET /forecast_points/...
GET /spots/...
GET /weather_stations/...

POST /spots/
Problem:

How to flush out stale weather data?
Solutions:

Delete stale data with a cron job.
Solutions:

Delete stale data with a **cron job**. Maintain when inserting weather data.

- Update "existing" or insert new entity if non-existing
How? Reuse db keys

Forecast key names:
/forecast_points/-23.0161,-43.3063/time_delta/9/
/forecast_points/-23.0161,-43.3063/time_delta/12/
/forecast_points/-23.0161,-43.3063/time_delta/15/
...

Calculating time delta:
time_delta = forecast time - calculation time
Too resource intensive

~100 entities for each forecast point are updated
Solutions cont'd:

Combine the **one-to-many** relationship into one entity.
class ForecastPoint(db.Model):
    point = db.GeoPtProperty()
    calculation_time = db.DateTimeProperty()
    forecasts = db.TextProperty()
    ...

class **ForecastPoint**(db.Model):
    point = db.GeoPtProperty()
    calculation_time = db.DateTimeProperty()
    forecasts = db.TextProperty()

forecasts is a JSON list:

```json
[
  {
    "direction": 269.1,
    "speed": 6.2,
    "temp": 7.7,
    "time": "2009-10-04T23:00:00"
  },(...)
]
```
Forecasts as entities:

/api/forecast_points/58.4555,8.8848/?_method=PUT 200 4835ms 25947cpu_ms 24586api_cpu_ms
/api/forecast_points/58.0703,6.7771/?_method=PUT 200 6089ms 25800cpu_ms 24420api_cpu_ms

Forecasts as text:

/api/forecast_points/57.6206,10.2756/?_method=PUT 200 130ms 202cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/56.3921,10.9201/?_method=PUT 200 232ms 198cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/56.5185,0.596/?_method=PUT 200 133ms 195cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/57.2571,9.5803/?_method=PUT 200 129ms 203cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/57.0705,9.6803/?_method=PUT 200 139ms 194cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/57.1233,8.6291/?_method=PUT 200 117ms 200cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/57.0445,8.4787/?_method=PUT 200 119ms 201cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/56.5194,8.7413/?_method=PUT 200 96ms 197cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/58.1905,8.0719/?_method=PUT 200 1106ms 205cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/58.8831,5.6025/?_method=PUT 200 121ms 193cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/56.0929,12.4686/?_method=PUT 200 114ms 194cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/56.126,12.3105/?_method=PUT 200 341ms 210cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/56.1081,12.3692/?_method=PUT 200 99ms 200cpu_ms 115api_cpu_ms 0kb
/api/forecast_points/55.9272,12.5191/?_method=PUT 200 99ms 200cpu_ms 115api_cpu_ms 0kb
Agenda

Motivation, vision, and demo

Architectural overview

Problem: No cron jobs (GAE)

Challenge: Inequality filters on one property only (GAE)

Challenge: Result set <= 1000 entities (GAE)
Geo. queries are not directly supported
Too many points
SELECT *
FROM Spots
WHERE
  lat > 54 AND
  lat < 58 AND
  lon > 8 AND
  lon < 16;
SELECT * 
FROM Spots 
WHERE 
  lat > 54 AND lat < 58 AND lon > 8 AND lon < 16;

"Inequality Filters Are Allowed On One Property Only"
-- GAE
Bounding box query

Using index on lat. and index on lon.
Solution:

Convert points to values in a single dimension using a scheme that preserves proximity.
Geohash

Base32 = "0123456789bcdefghjkmnpqrstuvwxyz"
Value = 012... 31
"0" <=> 00000₂ <=> (-67.5°, -157.5°)
Geohash

Base32 = "0123456789bcdefghjkmnpqrstuvwxyz"
Value = 012...31
"00" <=> 00000 000002 <=> (-87.1875°, -174.375°)
Note:

Points in the **same** grid cell have the **same** geohash prefix
Prefix query for proximity points (SQL)

```sql
SELECT *
FROM Spots
WHERE geohash LIKE 'U1%'`
SELECT *
FROM Spots
WHERE geohash LIKE 'U1%'

LIKE not available on GAE!
Prefix query for proximity points (SQL)

```
SELECT * 
FROM Spots 
WHERE geohash LIKE 'U1\%' 

LIKE not available on GAE!

SELECT * 
FROM Spots 
WHERE geohash >= 'U1' AND 
    geohash < 'U2'
```
Prefix query for proximity points (GAE)

```
query = db.Query(Spot)
query.filter('geohash >=', 'u1')
query.filter('geohash <', 'u1' + u'\ufffd')
```

The largest possible unicode char: ?
Advantage: proximity queries supported by index

<table>
<thead>
<tr>
<th>Kind</th>
<th>Property</th>
<th>Value</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spot</td>
<td>geohash</td>
<td>sws8whkz7yzb</td>
<td>.</td>
</tr>
<tr>
<td>Spot</td>
<td>geohash</td>
<td>u1vvsqdlrzrb</td>
<td>.</td>
</tr>
<tr>
<td>Spot</td>
<td>geohash</td>
<td>u1yznthncyzb</td>
<td>.</td>
</tr>
<tr>
<td>Spot</td>
<td>geohash</td>
<td>u1zjy5pd7fxg</td>
<td>.</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spot</td>
<td>geohash</td>
<td>u3bqklwvrgzy</td>
<td>.</td>
</tr>
</tbody>
</table>
Challenge:

"If more than 1000 entities match the query, only the first 1000 results are returned"

-- GAE doc.
Solution:
Apply paging using the geohash index.
Paging: only by using the geohash index

<table>
<thead>
<tr>
<th>Kind</th>
<th>Property</th>
<th>Value</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spot</td>
<td>geohash</td>
<td>sws8whkz7yzb</td>
<td>...</td>
</tr>
<tr>
<td>Spot</td>
<td>geohash</td>
<td>u1vvsqdlrzrb</td>
<td>...</td>
</tr>
<tr>
<td>Spot</td>
<td>geohash</td>
<td>u1yznthncyzb</td>
<td>...</td>
</tr>
<tr>
<td>Spot</td>
<td>geohash</td>
<td>u1zjy5pd7fxg</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Spot</td>
<td>geohash</td>
<td>u3bqklwvrgzy</td>
<td>...</td>
</tr>
</tbody>
</table>
Spots Paging: using the geohash index

```python
PAGE_SIZE = 2
def index(request):
    prefix = request.GET.get('gh_prefix', '')
    offset = request.GET.get('gh_offset', prefix)

    (...)
```
Spots Paging: using the geohash index

.../api/spots/?gh_prefix=u1&gh_offset=u1zrfef3xbzg

PAGE_SIZE = 2
def index(request):
    prefix = request.GET.get('gh_prefix', '')
    offset = request.GET.get('gh_offset', prefix)

    q = db.Query(Spot)
    q.filter('geohash >=', offset)
    q.filter('geohash <', prefix + u'�')
    q.order('geohash')

    spots = q.fetch(PAGE_SIZE + 1)

    (...
Spots Paging: using the geohash index

.../api/spots/?gh_prefix=u1&gh_offset=u1zrfe3xbzg

PAGE_SIZE = 2
def index(request):
    prefix = request.GET.get('gh_prefix', '')
    offset = request.GET.get('gh_offset', prefix)

    q = db.Query(Spot)
    q.filter('geohash >=', offset)
    q.filter('geohash <', prefix + u'�')
    q.order('geohash')

    spots = q.fetch(PAGE_SIZE + 1)

    has_next_page = len(spots) > PAGE_SIZE
    if has_next_page:
        qs = request.GET.copy()
        qs['gh_offset'] = spots[-1].geohash
        spots = spots[:-1]
    # create representation with uri to next page (...)


Spots Representation:

http://welovewind.com/api/spots/?gh_prefix=u1

{
"items":[
{
"name": "Bork Havn",
"lon": 8.2757949829101562,
"lat": 55.84650606768372,
"uri": "/api/spots/dk/bork_havn/",
"forecast_point":
"/api/forecast_points/55.8465,8.2758/",
"country_code": "dk"
},(...)],
"next":
"/api/spots/?gh_prefix=u1&gh_offset=u1zrfef3xbzg"
}
Challenge:

The proximity property is **not preserved** in all cases with geohash.
Problem: Proximity property of geohash
Include all neighbor cells

http://www.welovewind.com/examples/geohash/index.html
Conclusion

In this talk
Motivation and vision
Architectural overview
Problem: No cron jobs
Challenge: Limited inequality operators
Challenge: Result set $\leq 1000$ entities

The challenges are your friend.

The result
A mashup designed with high scalability.
Conclusion

In this talk
  Motivation and vision
  Architectural overview
  Problem: No cron jobs
  Challenge: Limited inequality operators
  Challenge: Result set <= 1000 entities

The **challenges** are your friend.

The **result**
  A mashup designed with high scalability.

More info
  http://welovewind.com/about

Thank you.