LBS ENGINE

陈志东、吴志斌

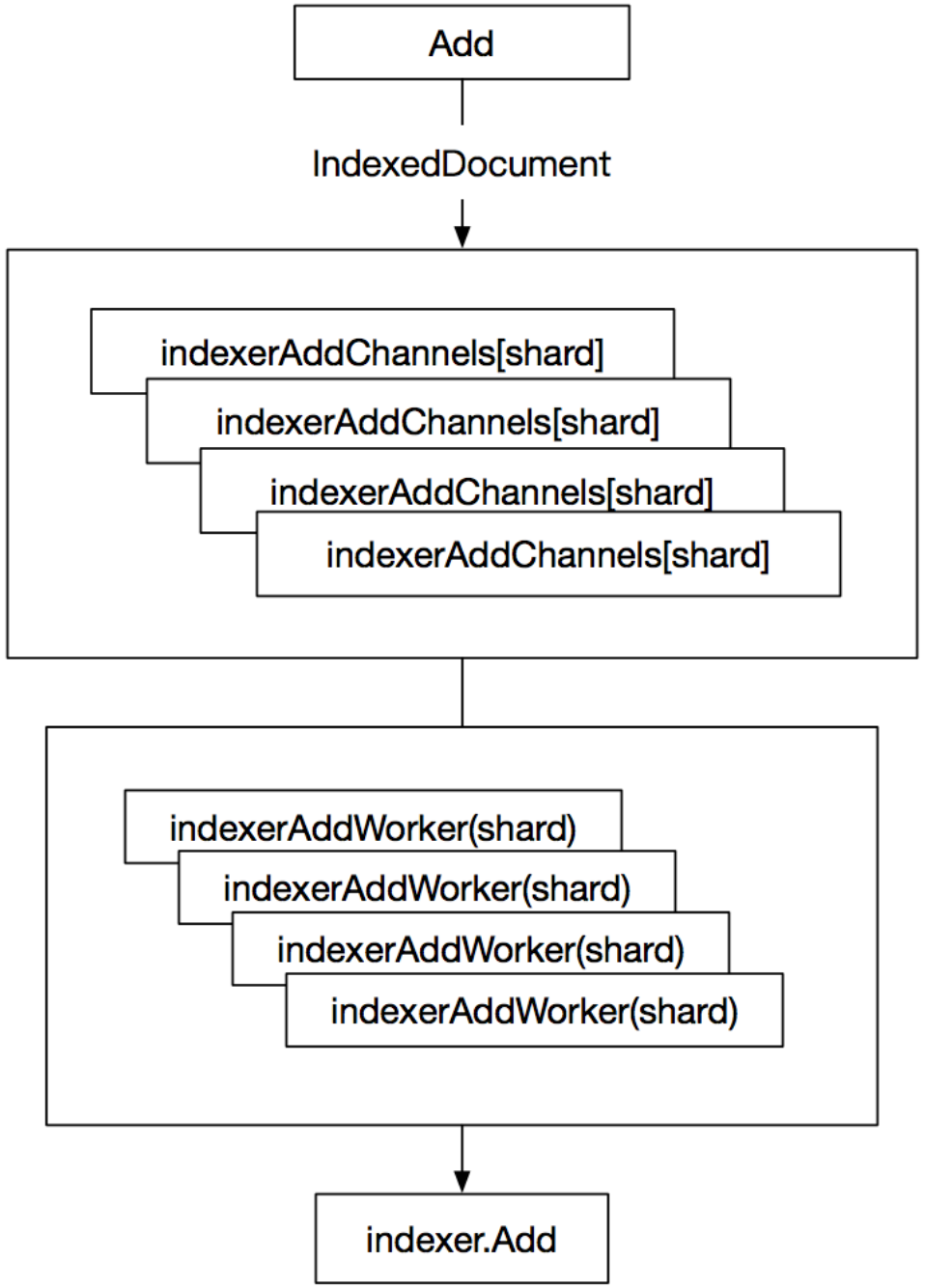
—— 2017Go基金会中国黑客马拉松

1. 项目背景

- 需求：搜索附近的人，并以性别、年龄等条件做筛选
- 解决方案：
  - MySQL，用where进行筛选，在sql中计算距离进行排序
  - MongoDB，使用MongoDB的空间搜索算法
  - ElasticSearch/Lucene，强大的全功能搜索引擎，Java，部署麻烦，吃内存
  - Redis，3.2.0开始支持GEO，性能非常好，但无法做到筛选
  - Bleve，Go开发的全文检索引擎，支持定位搜索，无法做到筛选
  - go-geoindex，全内存，快速，无法存储，重启依赖业务恢复数据
  - ...

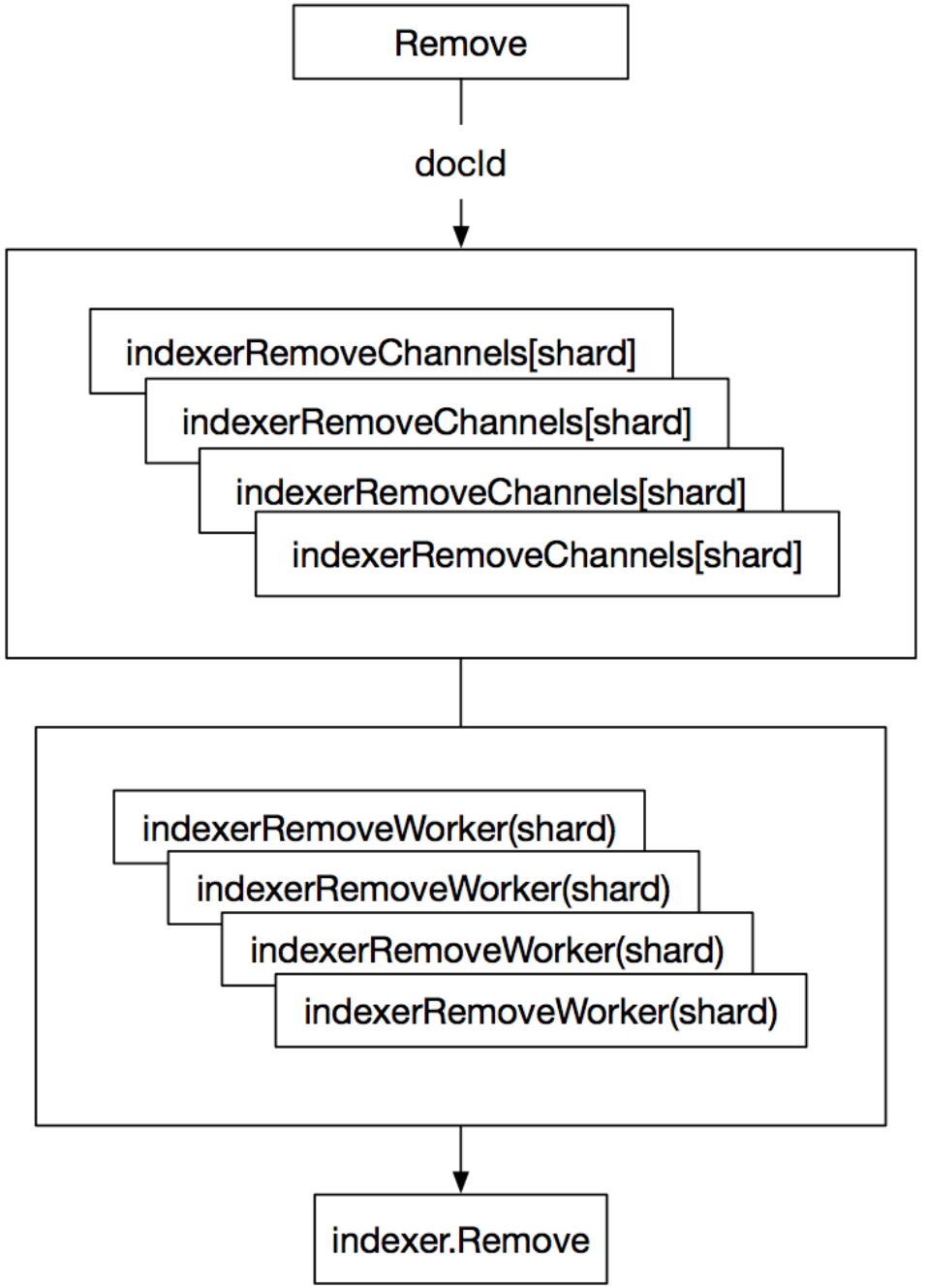
2. 引擎逻辑

写入


Add
IndexedDocument
indexerAddChannels[shard]
indexerAddChannels[shard]
indexerAddChannels[shard]
indexerAddChannels[shard]
indexerAddWorker(shard)
indexerAddWorker(shard)
indexerAddWorker(shard)
indexerAddWorker(shard)
indexer.Add

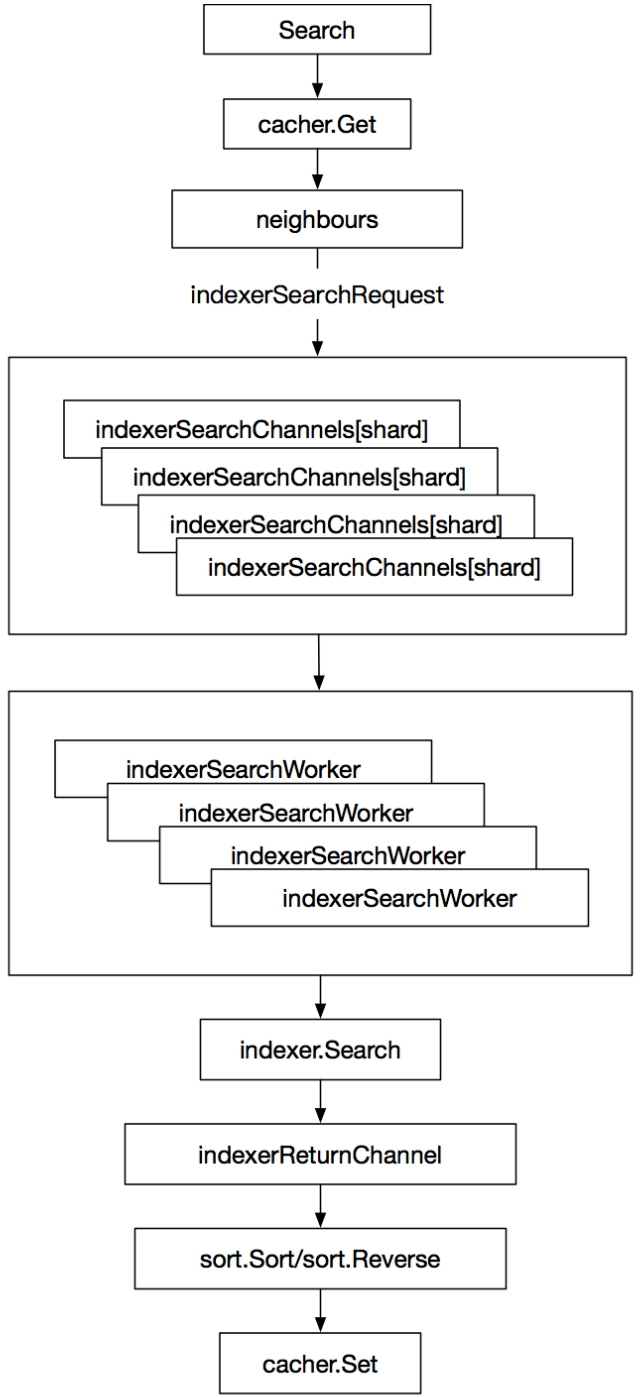
2. 引擎逻辑

删除


Remove
docId
indexerRemoveChannels[shard]
indexerRemoveChannels[shard]
indexerRemoveChannels[shard]
indexerRemoveChannels[shard]
indexerRemoveWorker(shard)
indexerRemoveWorker(shard)
indexerRemoveWorker(shard)
indexerRemoveWorker(shard)
indexer.Remove

2. 引擎逻辑

搜索


Search
cacher.Get
neighbours
indexerSearchRequest
indexerSearchChannels[shard]
indexerSearchChannels[shard]
indexerSearchChannels[shard]
indexerSearchChannels[shard]
indexerSearchWorker
indexerSearchWorker
indexerSearchWorker
indexerSearchWorker
indexer.Search
indexerReturnChannel
sort.Sort/sort.Reverse
cacher.Set

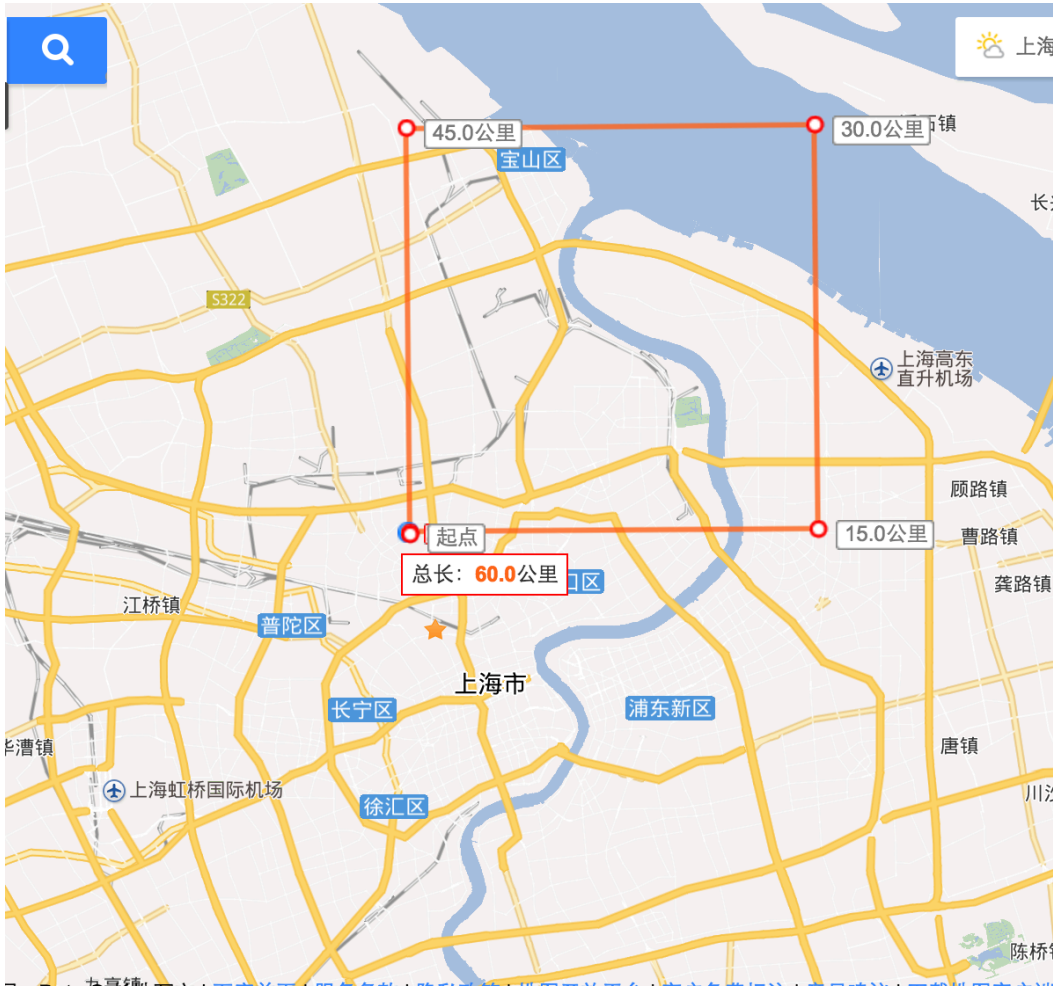
3 核心中的核心 - 存储

- 内存缓存
  - 5分钟有效期
  - 缓存一次搜索的所有排序结果
  - 为了用户取第二页数据时候不用再查一遍
- 永久存储
  - 借助Redis（其他KV结构存储也可以）
  - 使用Hash，按照网上流传的优化参数，限制1000个数据一组，可配置
  - 存储格式

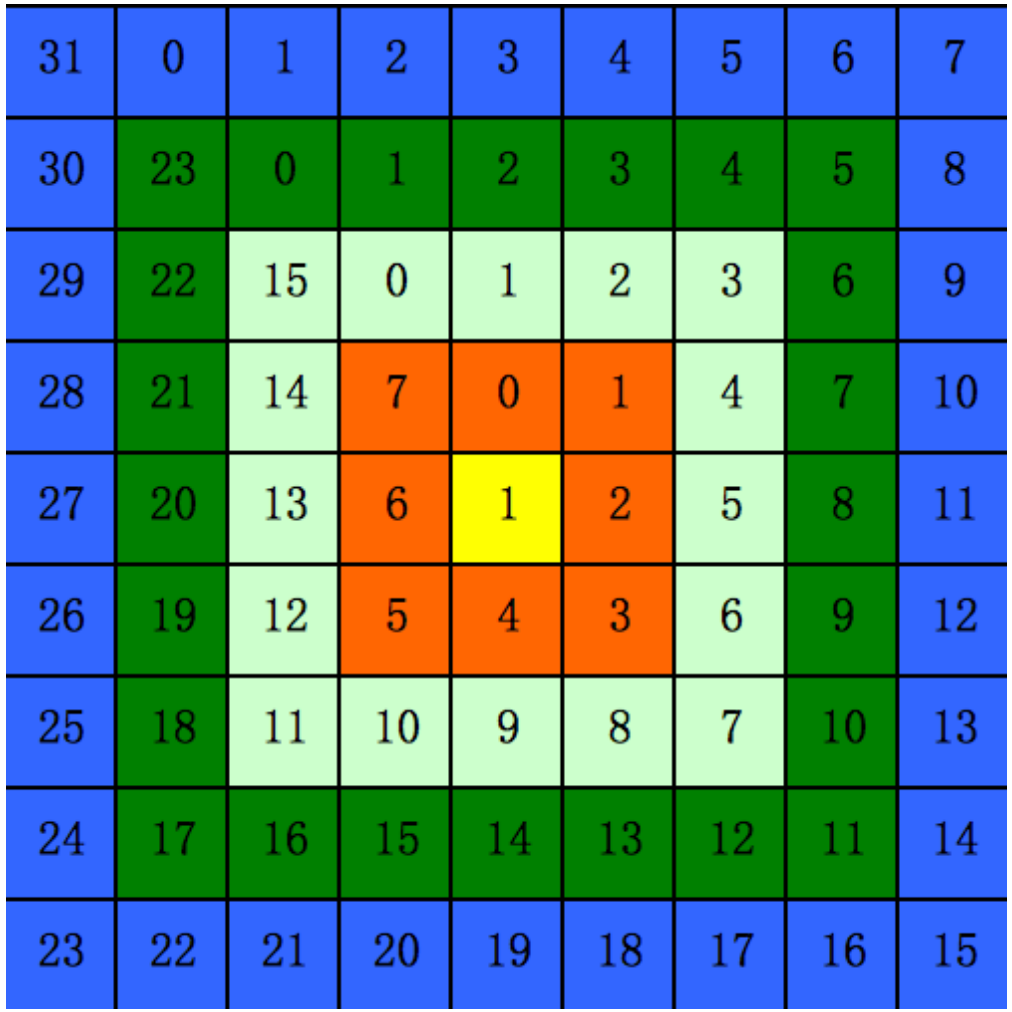
3. 核心中的核心 - GEOHASH

GEOHASH长度	宽	长
1	5009.4km	4992.6km
2	1252.3km	624.1km
3	156.5km	156km
4	39.1km	19.5km
5	4.9km	4.9km
6	1.2km	609.4m
7	152.9m	152.4m
8	38.2m	19m
9	4.8m	4.8m
10	1.2m	59.5cm

3. 核心中的核心 - GEOHASH



3. 核心中的核心 - GEOHASH



圈数	宽km	长km	面积(km2)
1	3.6	1.8282	6.58152
2	6	3.047	18.282
3	8.4	4.2658	35.83272
4	10.8	5.4846	59.23368
5	13.2	6.7034	88.48488

3.3 核心中的核心 - 距离计算(1)

- 距离计算是根据两个点经纬度坐标求两个点的距离。做了三个算法作为对比，并分不同场合下使用
- 1.标准球体测距算法（haversine formula）

以地球球心为原点，本初子午线和赤道的两个交点的连线为x轴，赤道所形成的圆截面上垂直于x轴的直线作为y轴，南北极点的连线为Z轴建立xyz三维坐标系

（实际上如果更追求精确度应该使用椭球型的算法，因为地球并不是标准球体，东西方向的直径12756km比南北方向的直径12714km要长）

大致的数学推导过程是根据A,B两点的经纬度ja,wa,jb,wb求得A,B两点对应坐标系上的值为

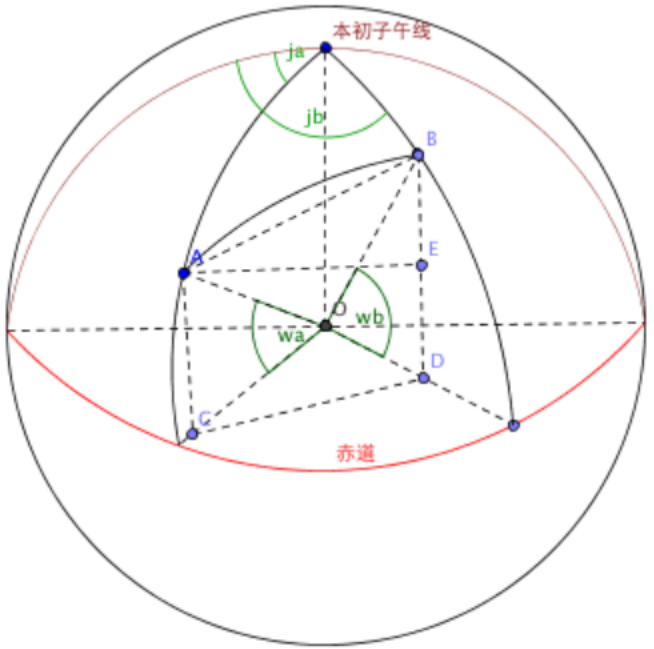
Xa = R·cos(wa)cos(ja)　　Xb = R·cos(wb)cos(jb)

Ya = R·cos(wa)sin(ja)　　Yb= R·cos(wb)sin(jb)

Za = R·sin(wa)　　Zb= R·sin(wb)

然后根据余弦定理求得夹角AOB的弧度值α，那么弧线AB的长度就是

R ·α = R·arccos(cos(wa)cos(wb)cos(jb-ja) + sin(wa)sin(wb))



3.3 核心中的核心 - 距离计算(2)

- 2.二维平面近似算法

然而我们的项目是基于本地服务的，经纬度每一度代表的距离是dis = 35.433km。上海作为中国最大的城市之一，面积6340平方公里，如果以圆面积算，那么这个圆半径为44.93km，略大于35.433km。那么实际上在本地服务的两点测距上就可以采用把三维球面近似为二维平面

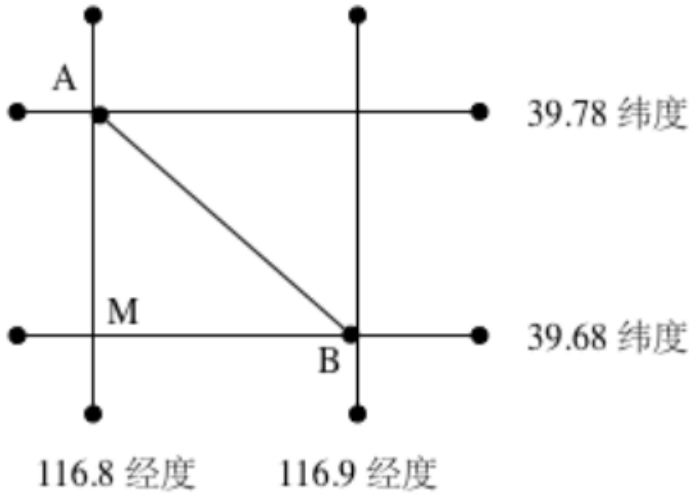
的方法来提升计算效率（直接使用勾股定理）。

AM = |wa-wb|*dis ; BM = |ja-jb| * dis * cos((wa+wb)/2)

$AB = sqrt(AM^2 + BM^2)$

- 3.二维平面近似算法(预设本地城市经纬度)

但是上面的算法效率还是可以提高，因为既然是本地服务，同一个城市的纬度不可能差异过大，所以可以通过预设本地城市的经纬度来去掉**cos((wa+wb)/2)**这部分的计算



3.3 核心中的核心 - 距离计算性能对比

```
C:/go/bin/go.exe test -v [C:/code/github/src/lbsengine/distanceMeasure]
=== RUN   TestStardard
--- PASS: TestStardard (1.93s)
      distanceMeasure_test.go:49: Haversine公式标准球体算法耗时: 1.927125E+09 ns
=== RUN   TestQuickWithoutSetLocation
--- PASS: TestQuickWithoutSetLocation (0.46s)
      distanceMeasure_test.go:61: 平面近似二维距离算法（勾股定理，未预设城市经纬度）的耗时: 4.637418E+08 ns
=== RUN   TestQuick
设置了城市坐标，城市名：  上海 经纬度坐标：  &{121 31}
本次快速测距算法设置的城市为上海,输入的基准经纬度坐标为[东经121.000000°， 北纬31.000000°]如果当前保存的城市与你期望的城市不符，请重新
--- PASS: TestQuick (0.12s)
      distanceMeasure_test.go:68: 尚未输入本地城市的基准经纬度坐标
      distanceMeasure_test.go:82: 平面近似二维距离算法（勾股定理,已预设城市经纬度）的耗时: 1.233284E+08 ns
=== RUN   TestCompare
--- PASS: TestCompare (2.44s)
      distanceMeasure_test.go:105: 10762.34508070739 3.0198840829758573e+07
      distanceMeasure_test.go:107: 精度为0.100000度时，换算以米为单位时表示搜索范围距离在11131.884502米内时:
            标准差组1为0.034581米,组2为1.831782米
PASS
ok      lbsengine/distanceMeasure  9.086s
成功: 进程退出代码 0.
```

4. Benchmark

```go
func RandomPoint() (lat, lng float64) {
	lat = 40.8137674 + godata.Round(rand.Float64()*0.01, 7)
	lng = -73.8525142 + godata.Round(rand.Float64()*0.01, 7)
	return
}
```

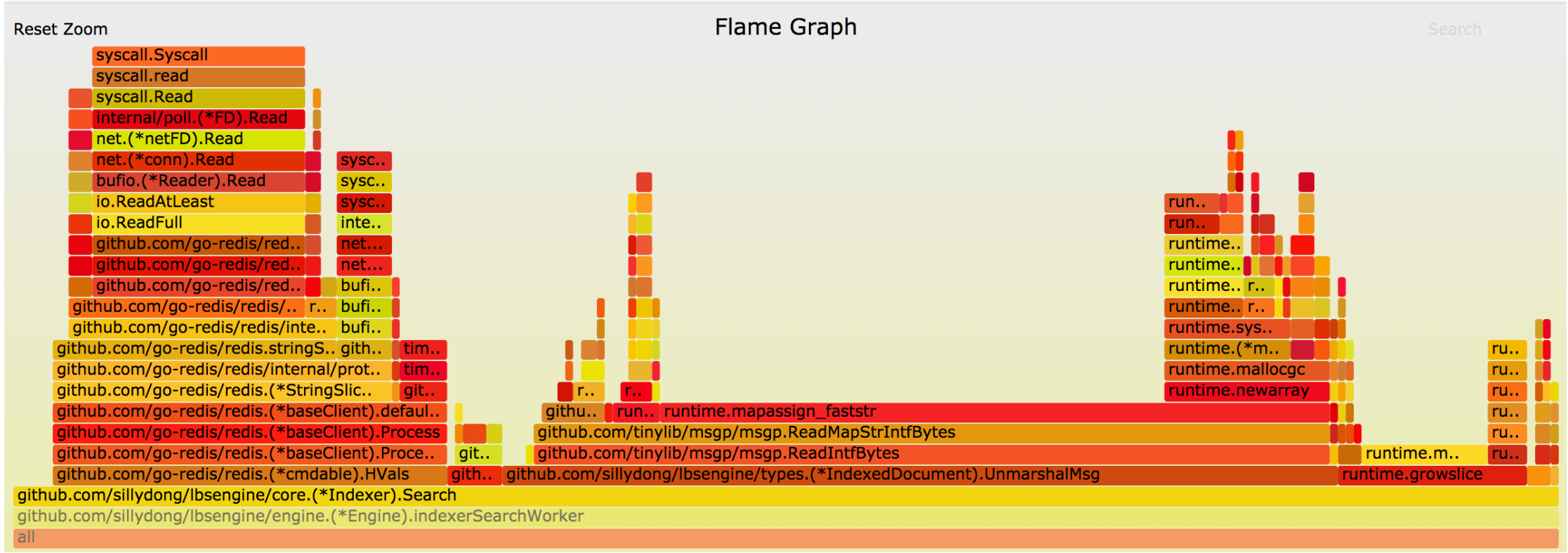
```
# Memory
used_memory:2175024
used_memory_human:2.07M
used_memory_rss:18726912
used_memory_rss_human:17.86M
used_memory_peak:55851232
used_memory_peak_human:53.26M
used_memory_peak_perc:3.89%
used_memory_overhead:1013670
used_memory_startup:963312
used_memory_dataset:1161354
used_memory_dataset_perc:95.84%
total_system_memory:8589934592
total_system_memory_human:8.00G
```

```
CZD:engine chenzhidong$ go test -v -run="-" -bench="Benchmark" -benchmem
goos: darwin
goarch: amd64
pkg: github.com/sillydong/lbsengine/engine
BenchmarkAdd-4                 10000            124572 ns/op            1564 B/op         44 allocs/op
BenchmarkSearch-4                100          16791664 ns/op         5122677 B/op      80649 allocs/op
```

```
127.0.0.1:6379[4]> keys *
1) "geo"
127.0.0.1:6379[4]> zrange geo 0 10 withscores
 1) "1387"
 2) "17918772281009 73"
 3) "8904"
 4) "17918772281010 90"
 5) "9268"
 6) "17918772281015 84"
 7) "4848"
 8) "17918772282244 05"
 9) "9883"
10) "17918772282552 14"
11) "8058"
12) "17918772282632 34"
13) "9440"
14) "17918772282633 15"
15) "9611"
16) "1791877228
17) "1346"
18) "1791877228
19) "6297"
20) "1791877228
21) "2696"
22) "17918772282766 32"
```

```
# Memory
used_memory:2228336
used_memory_human:2.13M
used_memory_rss:23691264
used_memory_rss_human:22.59M
used_memory_peak:55851232
used_memory_peak_human:53.26M
used_memory_peak_perc:3.99%
used_memory_overhead:1013014
used_memory_startup:963312
used_memory_dataset:1215322
used_memory_dataset_perc:96.07%
total_system_memory:8589934592
total_system_memory_human:8.00G
```

```
CZD:dbtest chenzhidong$ go test -v -run="-" -bench="BenchmarkR" -benchmem
goos: darwin
goarch: amd64
pkg: github.com/sillydong/lbsengine/dbtest
BenchmarkRAdd-4            10000            105924 ns/op             409 B/op         13 allocs/op
BenchmarkRSearch-4         10000            105889 ns/op             953 B/op         21 allocs/op
```


Reset Zoom
Flame Graph
Search
syscall.Syscall
syscall.read
syscall.Read
internal/poll.(*FD).Read
net.(*netFD).Read
net.(*conn).Read
bufio.(*Reader).Read
io.ReadAtLeast
io.ReadFull
github.com/go-redis/red..
github.com/go-redis/red..
github.com/go-redis/red..
github.com/go-redis/redis/..
github.com/go-redis/redis/inte..
github.com/go-redis/redis.stringS..
github.com/go-redis/redis/internal/prot..
github.com/go-redis/redis.(*StringSlic..
github.com/go-redis/redis.(*baseClient).defaul..
github.com/go-redis/redis.(*baseClient).Process
github.com/go-redis/redis.(*baseClient).Proce..
github.com/go-redis/redis.(*cmdable).HVals
sysc..
sysc..
sysc..
inte..
net...
net...
bufi..
r..
bufi..
bufi..
gith..
tim..
tim..
git..
git..
gith..
r..
r..
githu..
run..
runtime.mapassign_faststr
github.com/tinylib/msgp/msgp.ReadMapStrIntfBytes
github.com/tinylib/msgp/msgp.ReadIntfBytes
github.com/sillydong/lbsengine/types.(*IndexedDocument).UnmarshalMsg
run..
run..
runtime..
runtime..
runtime..
runtime..
runtime.sys..
runtime.(*m..
runtime.mallocgc
runtime.newarray
r..
r..
runtime.m..
runtime.growslice
ru..
ru..
ru..
ru..
ru..
ru..
github.com/sillydong/lbsengine/core.(*Indexer).Search
github.com/sillydong/lbsengine/engine.(*Engine).indexerSearchWorker
all

5. 优化点

- 维护一份内存索引，启动时从永久存储初始化内存索引
- 优化序列化/反序列化方法
- 参考Redis优化geohash算法

6. 示例

7. 谢谢