Golang在百万亿搜索引擎中的应用

Poseidon

郭军@360

自我介绍

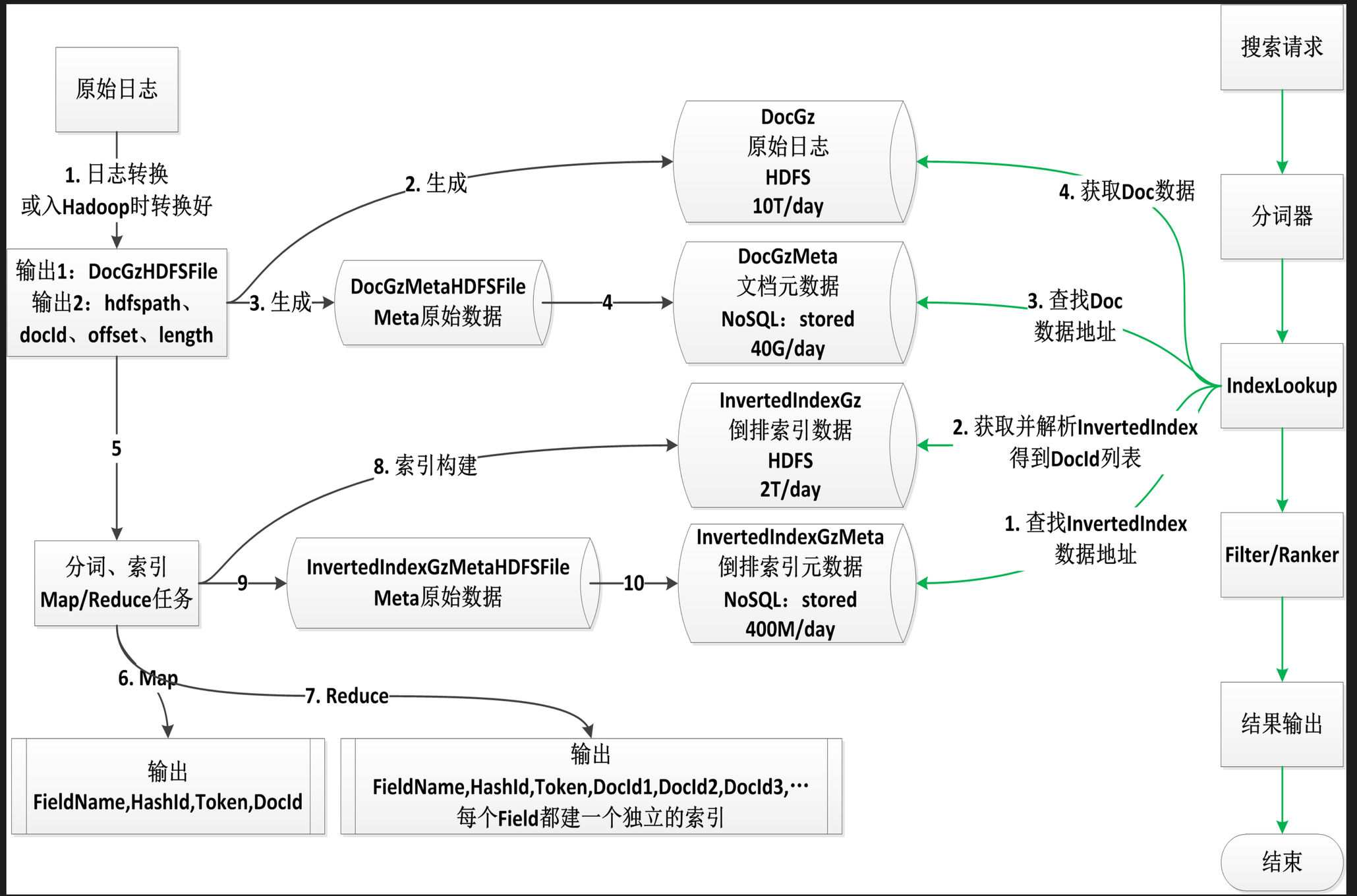
- 360核心安全事业部 云引擎开发组技术团队
- guojun-s@360.cn
- https://github.com/guojun1992

目录

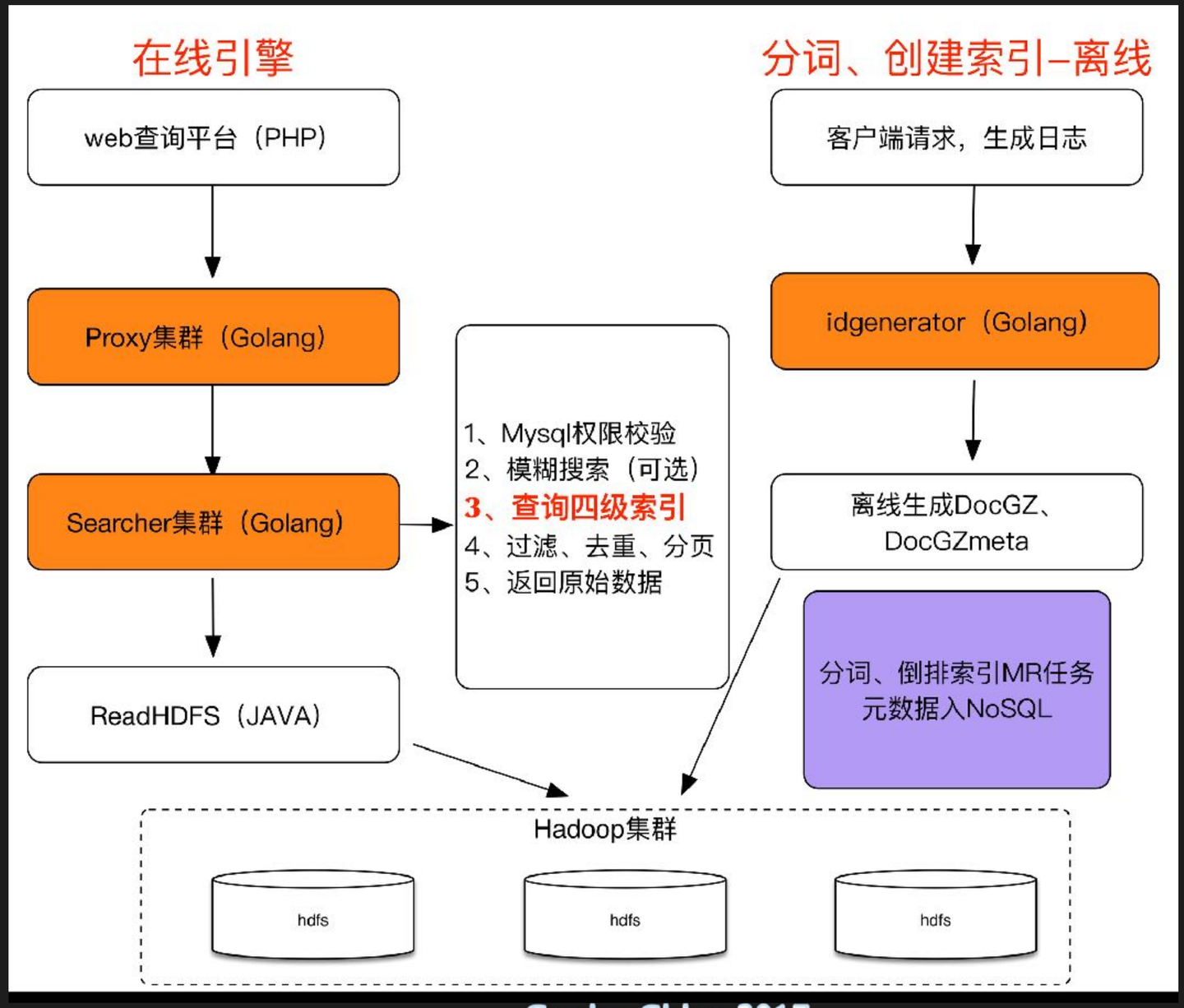
- 设计目标
- Go应用场景与遭遇的挑战
- 怎样应对?
- 开源的改变
- 总结

设计目标

- 总数据量：保留3年历史数据，百万亿条、大小100PB
- 秒级交互式搜索响应
- 每天支持2000亿增量数据灌入
- 原始数据仅存一份
- 对现有的MR任务无侵略
- 自定义分词策略
- 故障转移，节点负载均衡，自动恢复
- 支持单/多天批量查询，批量下载


原始日志
1. 日志转换
或入Hadoop时转换好
输出1：DocGzHDFSFile
输出2：hdfspath、
docId、offset、length
2. 生成
DocGz
原始日志
HDFS
10T/day
3. 生成
DocGzMetaHDFSFile
Meta原始数据
4
DocGzMeta
文档元数据
NoSQL：stored
40G/day
5
8. 索引构建
InvertedIndexGz
倒排索引数据
HDFS
2T/day
分词、索引
Map/Reduce任务
9
InvertedIndexGzMetaHDFSFile
Meta原始数据
10
InvertedIndexGzMeta
倒排索引元数据
NoSQL：stored
400M/day
6. Map
7. Reduce
输出
FieldName,HashId,Token,DocId
输出
FieldName,HashId,Token,DocId1,DocId2,DocId3,…
每个Field都建一个独立的索引
搜索请求
分词器
IndexLookup
Filter/Ranker
结果输出
结束
4. 获取Doc数据
3. 查找Doc
数据地址
2. 获取并解析InvertedIndex
得到DocId列表
1. 查找InvertedIndex
数据地址

架构设计


在线引擎
web查询平台（PHP）
Proxy集群（Golang）
Searcher集群（Golang）
ReadHDFS（JAVA）
1、Mysql权限校验
2、模糊搜索（可选）
3、查询四级索引
4、过滤、去重、分页
5、返回原始数据
分词、创建索引–离线
客户端请求，生成日志
idgenerator（Golang）
离线生成DocGZ、DocGZmeta
分词、倒排索引MR任务
元数据入NoSQL
Hadoop集群
hdfs
hdfs
hdfs

用ProtoBuffer描述核心数据结构

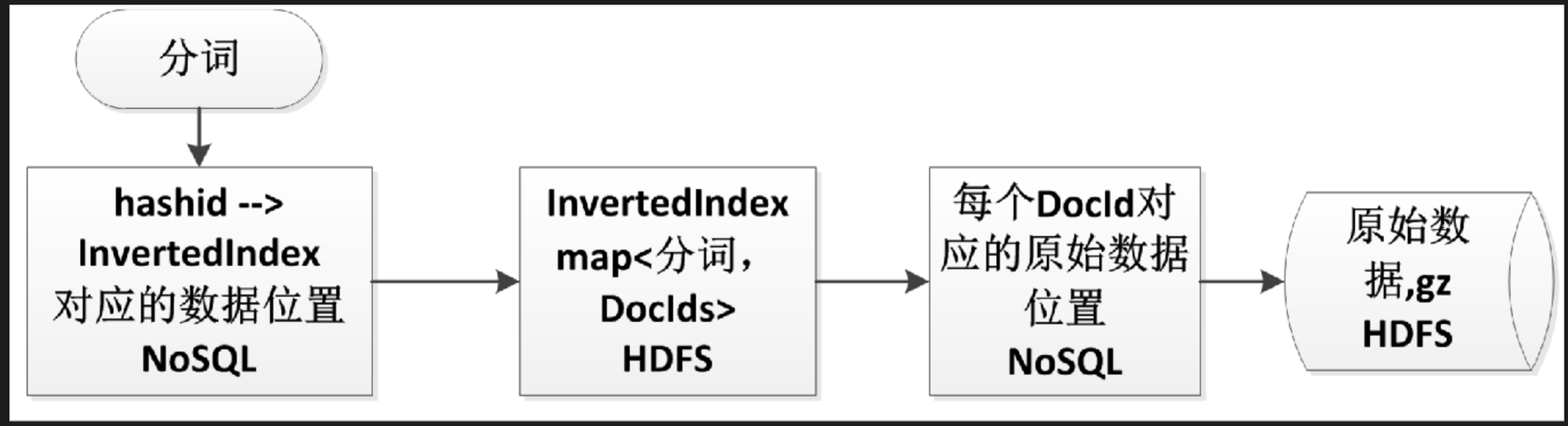
```
message DocId {
    int64 docId = 1;    // Document ID
    uint32 rowIndex = 2;// 在文档中的行号，从0开始编号
}
```

```
message InvertedIndex {
    map<string/*分词*/, DocIdList> index = 1;
}
```

```
// 存入NoSQL中，Key=int(hashid/N)
message InvertedIndexGzMeta {
    int64 offset = 1; // 某一个 *InvertedIndexGzMeta* 所在 hdfs 文件中的起始地址偏移量
    int64 length = 2; // *InvertedIndexGzMeta* 的所占数据长度
    string path   = 3; // 可以根据时间、索引表名、hashid等信息推断出来
    //uint32 hashid = 4; // 可以通过 *Token* 进行hash运算计算出来
}
```

弥补倒排索引的缺陷—四级索引

- 关键字 -> DocidList　Docid -> Doc

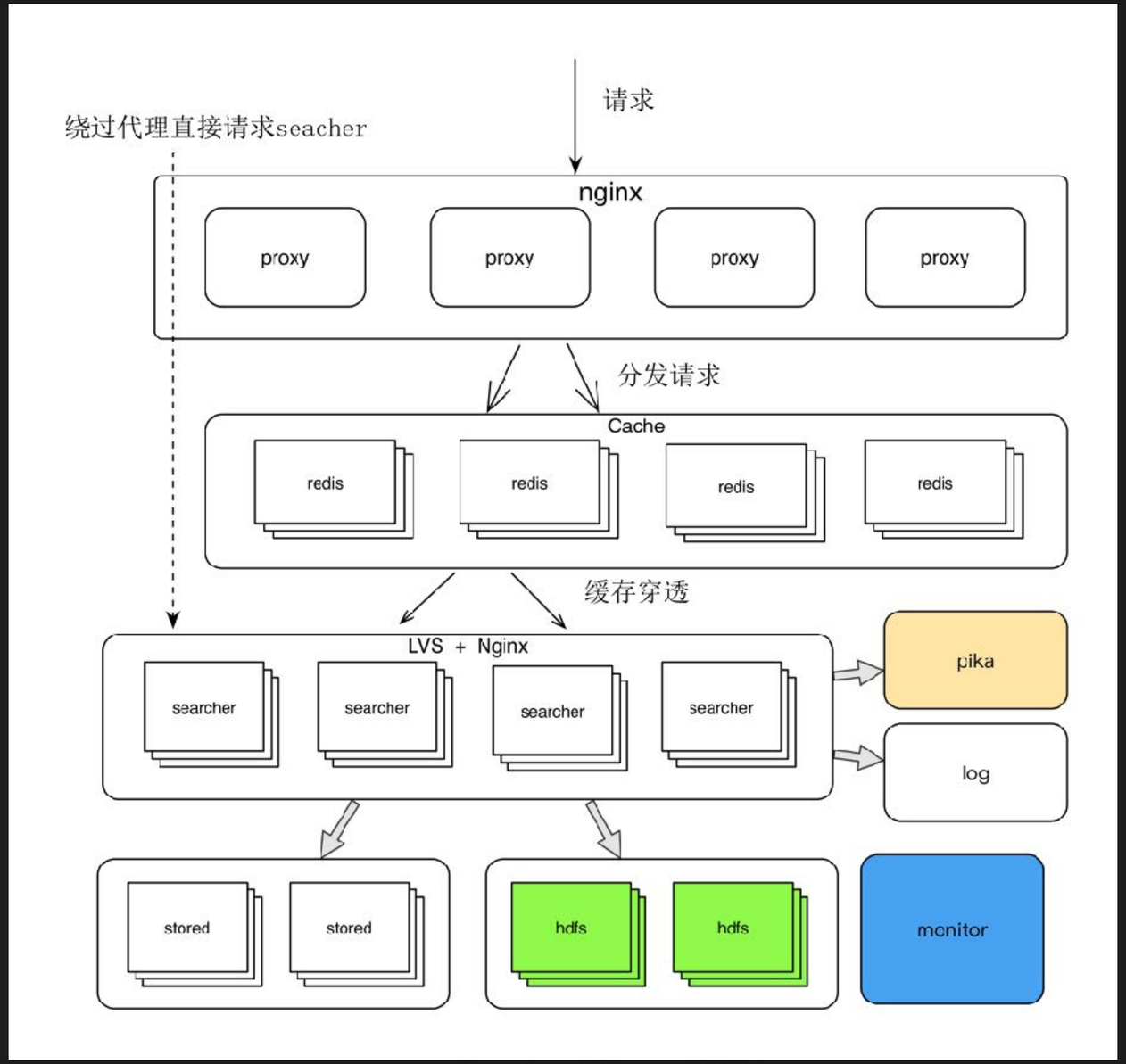


- “计算机科学领域的任何问题都可以通过增加一个间接的中间层来解决”

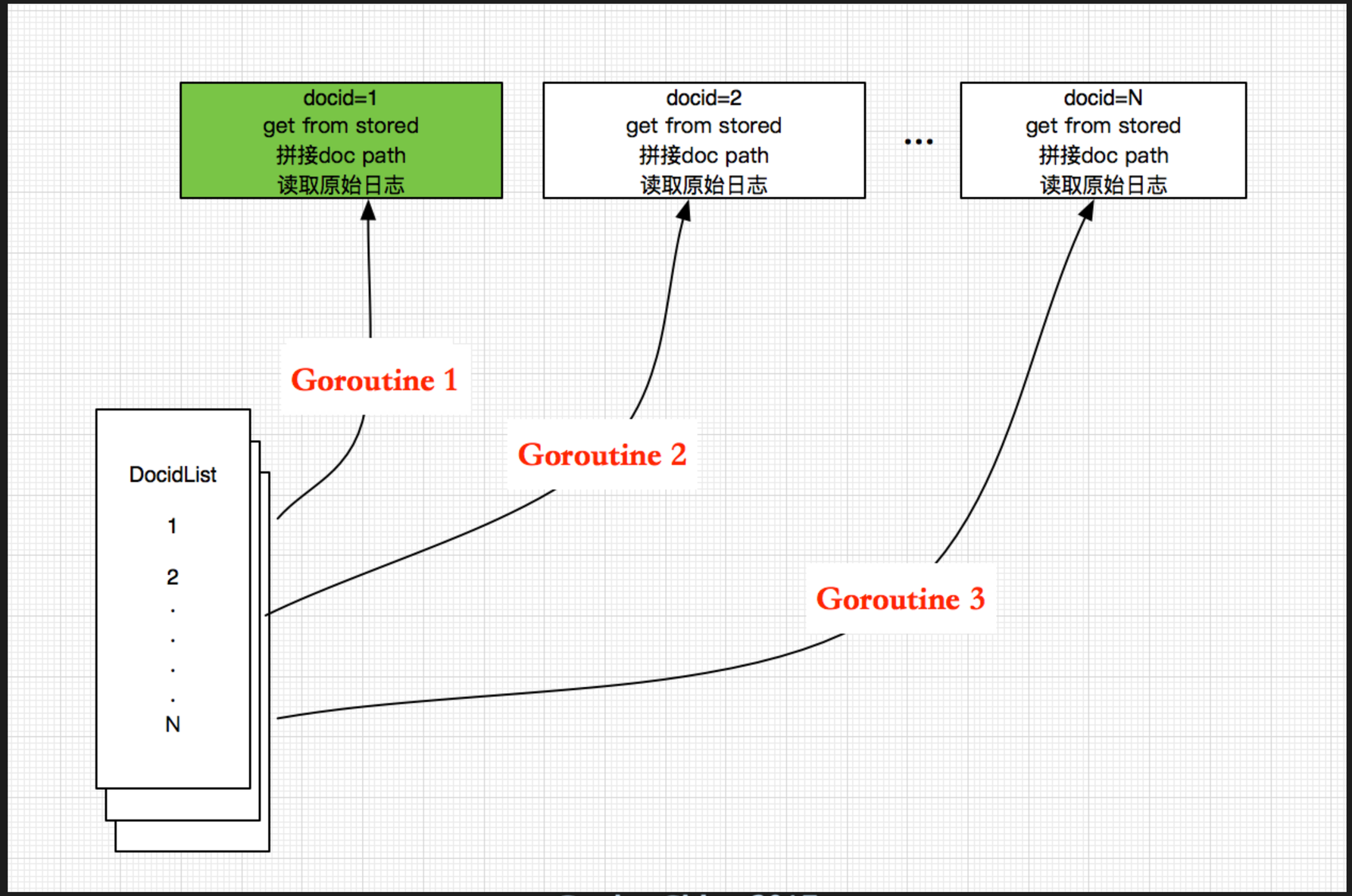
idgenerator

- docid：每天27700亿，4字节，总计11TB
- 采用分段区间获取降低qps，每天的id重新从0开始分配
- key：业务名+时间

Proxy/Searcher详细设计

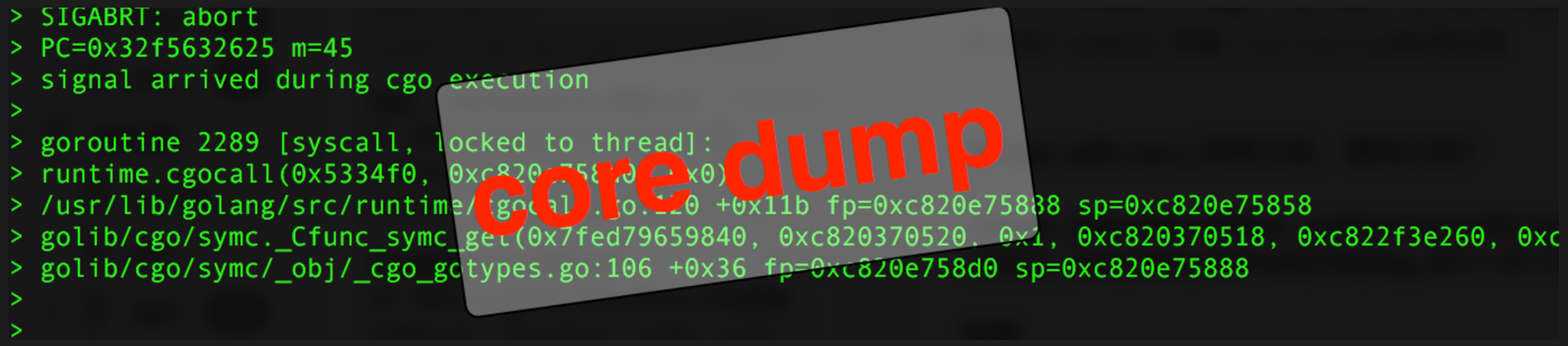


Searcher并发模型



问题与瓶颈

- 基础组件是c++，c++ -> c -> CGO -> Go
- gdb调试 进程过多



解决方案

- 用Go重新实现一遍
- 将组件作为http服务，Go Client调用，做集中式处理

问题与瓶颈

- 大量使用goroutine，子协程panic在主协程不能被recover，如何统一处理？

解决方案

- 通道类型为struct，封装正常数据和error，在主协程从通道取出数据，统一做处理

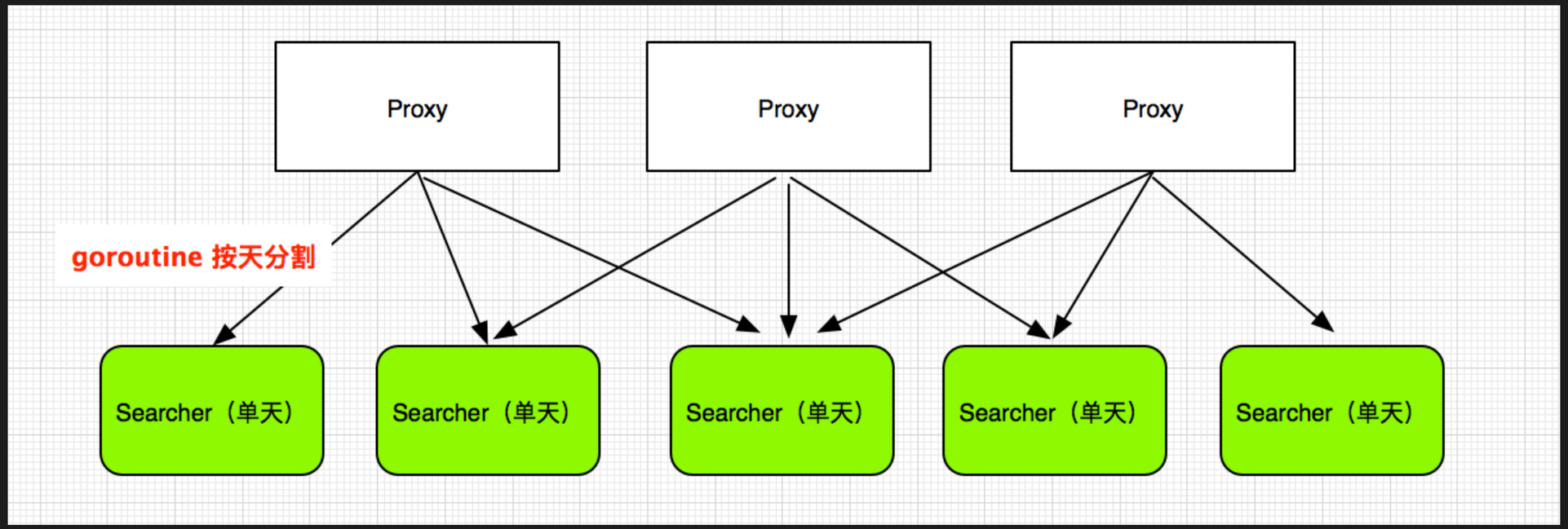
```
type DocDataResult struct {
    DocId    int64
    RowIndex int32
    Data     []byte
    Err      error
}
```

经验小结

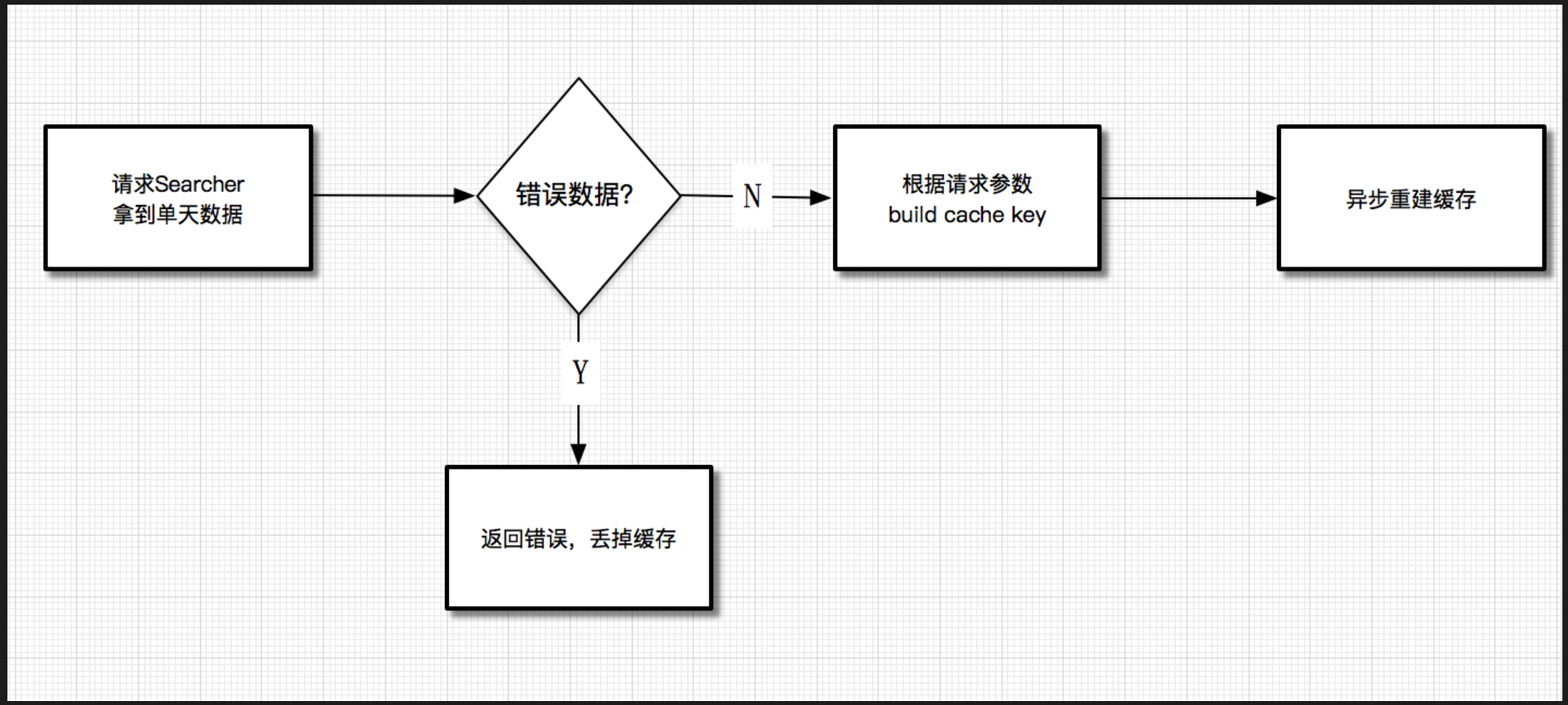
- 即使精通多种语言，最好**不要混用**，**谨慎**引入其他语言的解决方案
- 不要完全相信recover，它**不能恢复**runtime的一些panic

```
defer func() {
    if err := recover(); err != nil {
        in = []reflect.Value{reflect.ValueOf(err)}
        method := vc.MethodByName("OutputError")
        method.Call(in)
    }
}()
```

Proxy多天并发查询设计



Proxy多天并发Build Cache设计



索引数据变“冷”

- 同一份数据在缓存时间内不会走整个readhdfs流程
- build index 程序挂了会报警感知，但是数据错误却是未知
- 修复数据重建索引需要时间

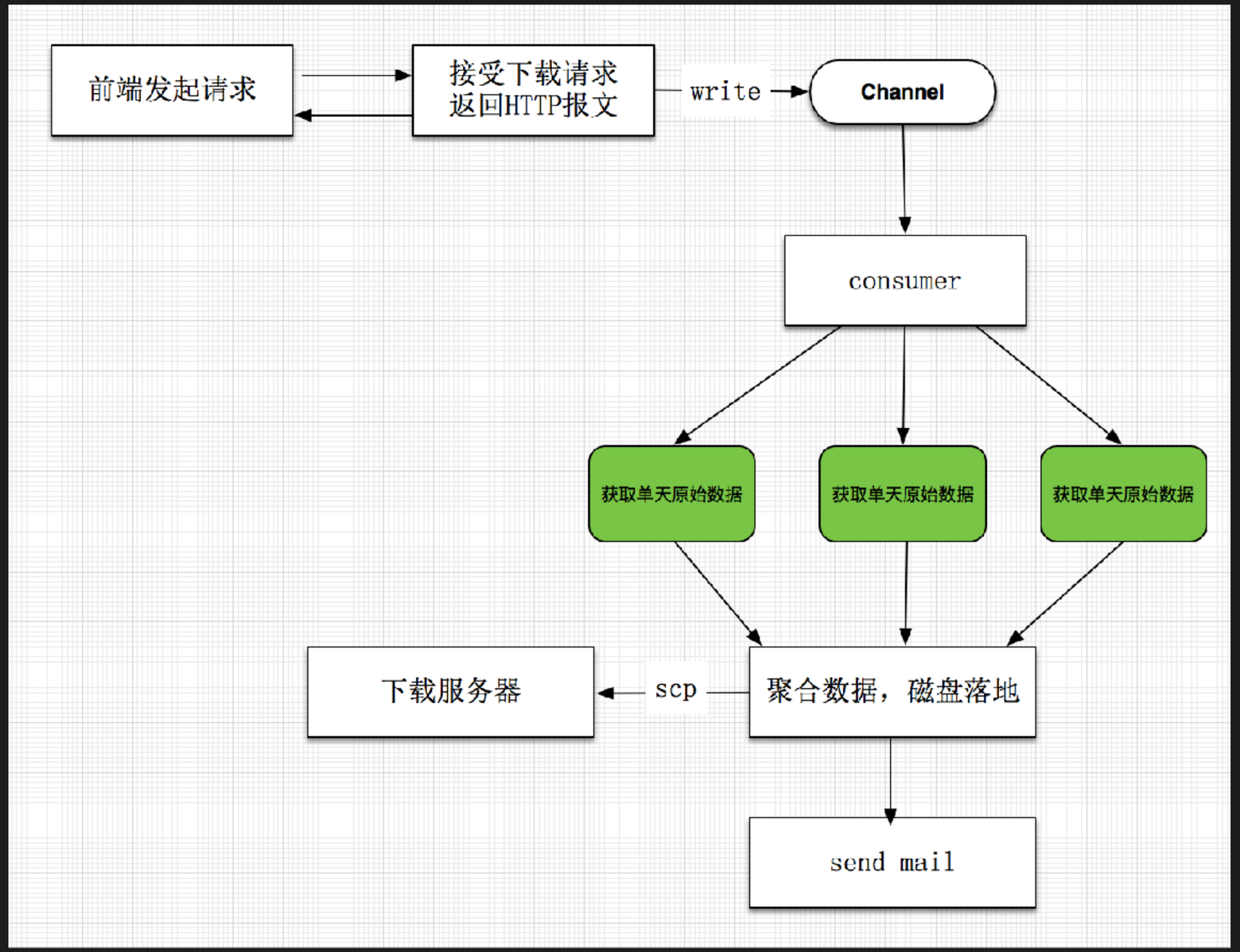
解决方案

- 减少缓存时间，在可容忍错误数据的时间内，用户查询及时发现问题
- 参考NSQ，利用for+select的不确定性来分流，随机流量到cache和hdfs做热测试，缺点：开发成本较高

经验小结

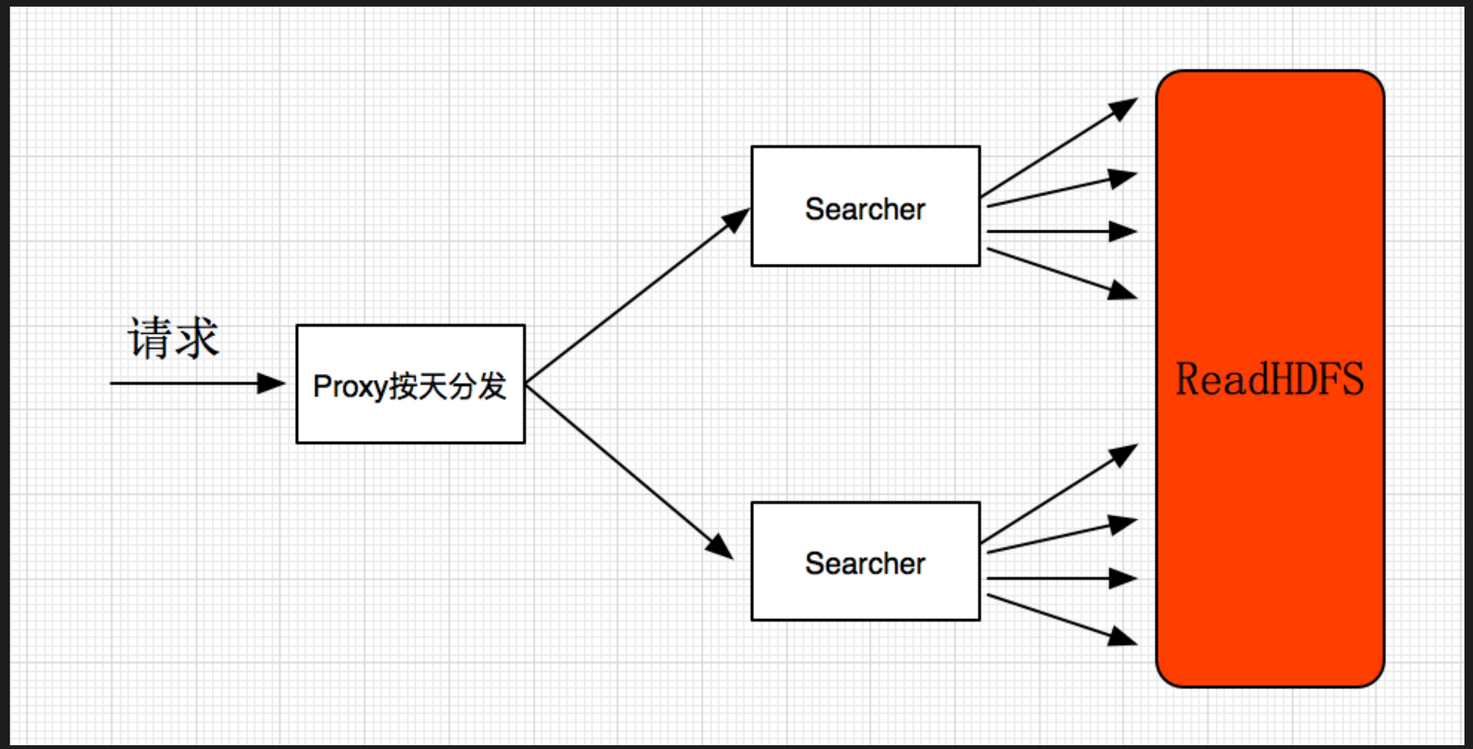
- 选择简单、有效、够用的解决方案
- 利用goroutine设计并发程序很方便，但是程序的并发运行模型要hold住

Proxy多天异步下载



ReadHDFS雪崩效应

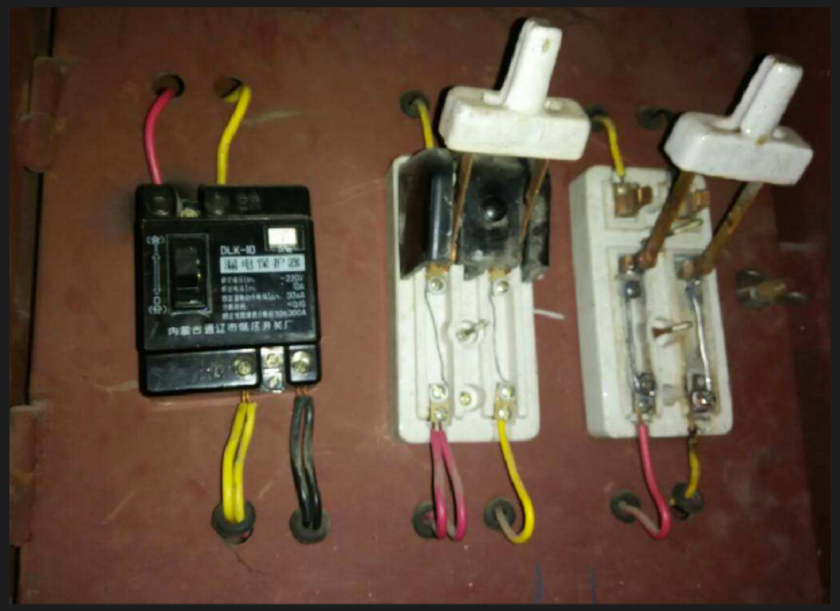
- goroutine太多，底层readhdfs挂掉



ReadHDFS雪崩效应

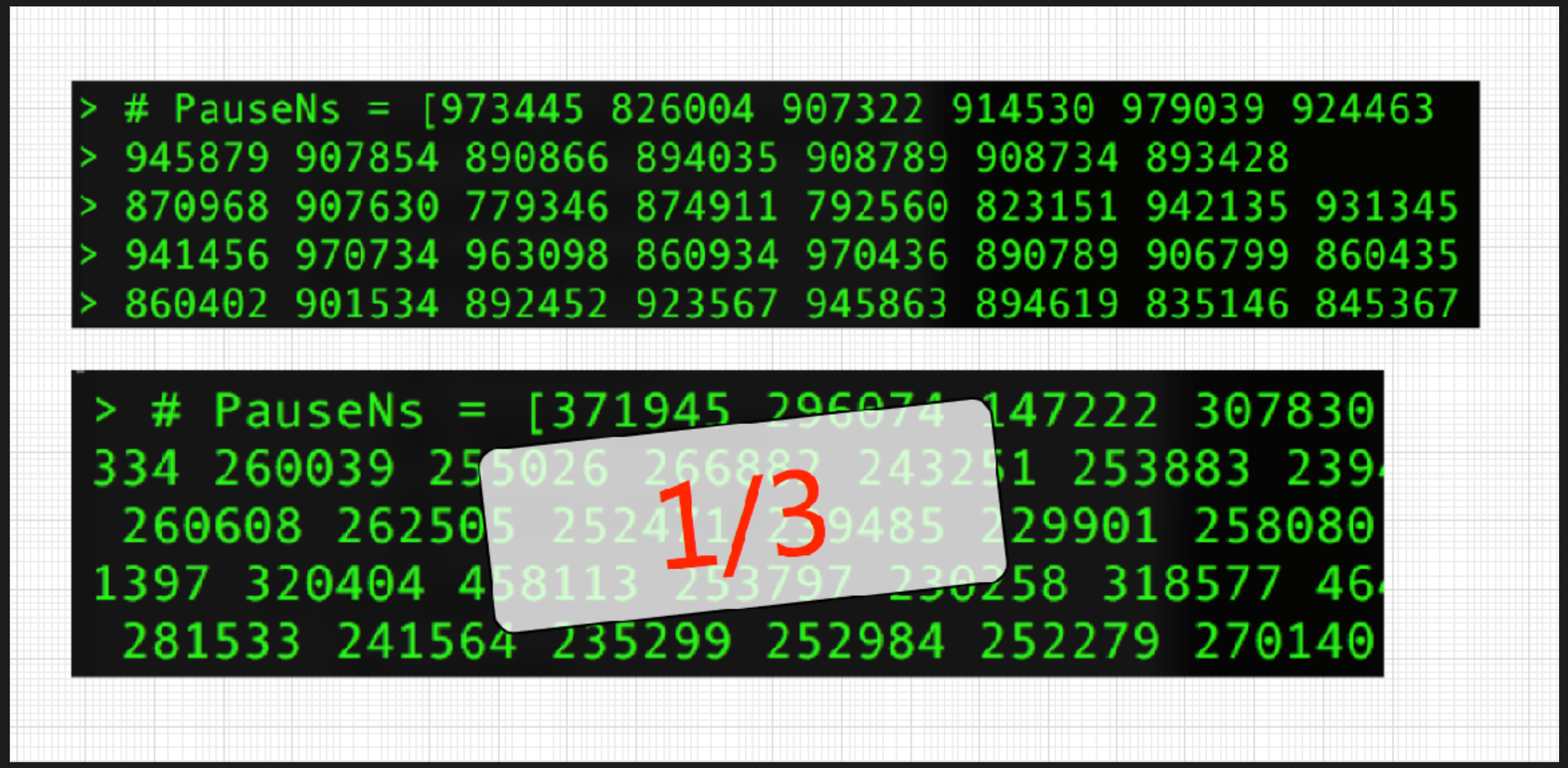
解决方案

- 连接池
- 熔断机制—超过连接数，直接返回error

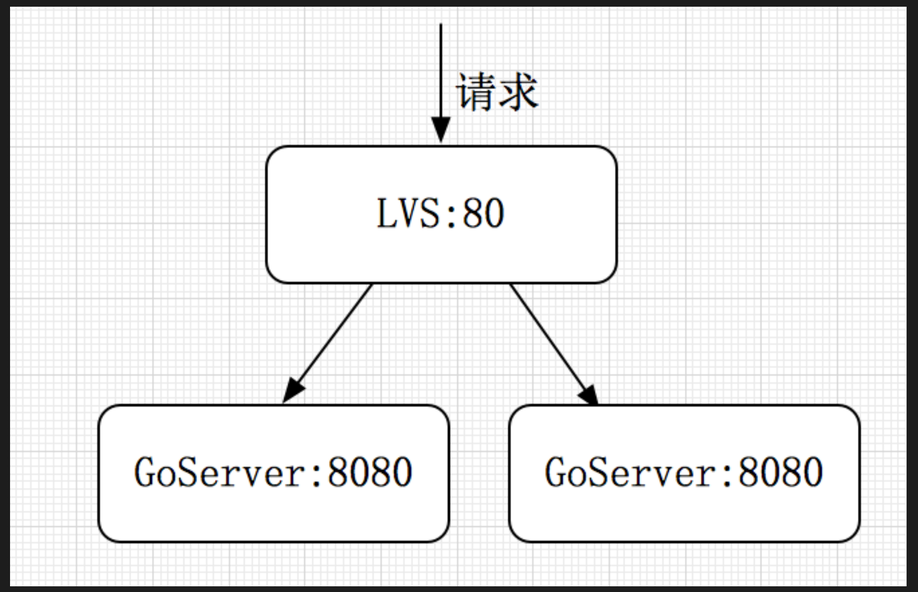


升级至1.7 GC变化

- GC在我们的系统中本不是瓶颈，仅仅是交流升级测试结果

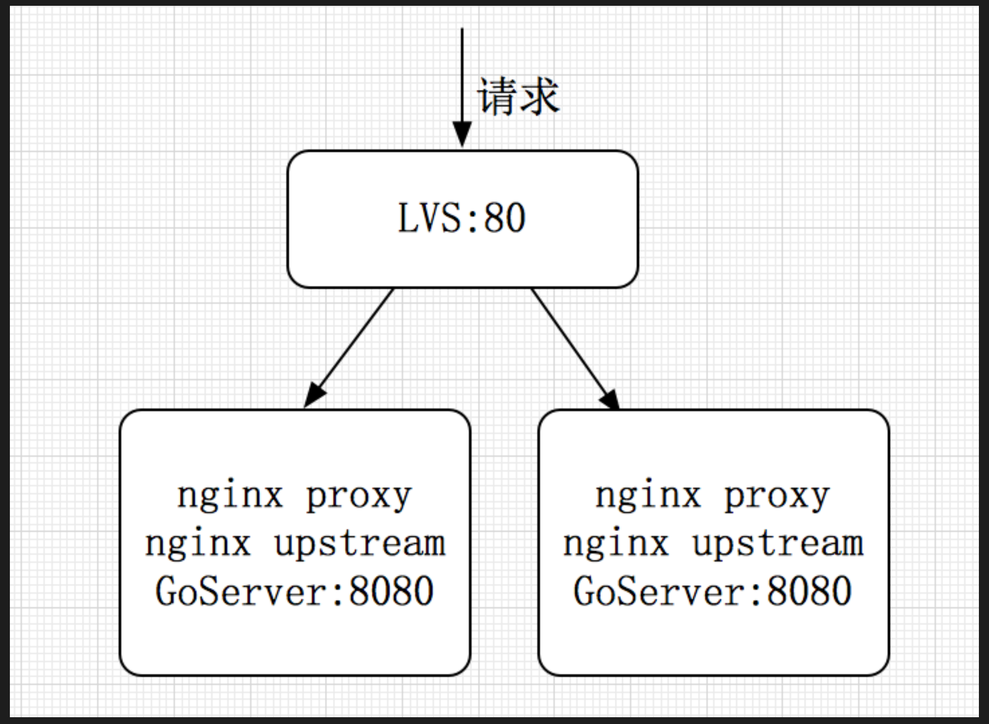


nginx代理问题



- 前端要不要加nginx代理?

访问控制和负载均衡




```
upstream searchers {
    server                          weight=100;
    server                          weight=100;
    server                          weight=100;
    server                          weight=100;
    server                          weight=100;
    server                          weight=100;
    server                          weight=100;
    server                          weight=100;
}
```

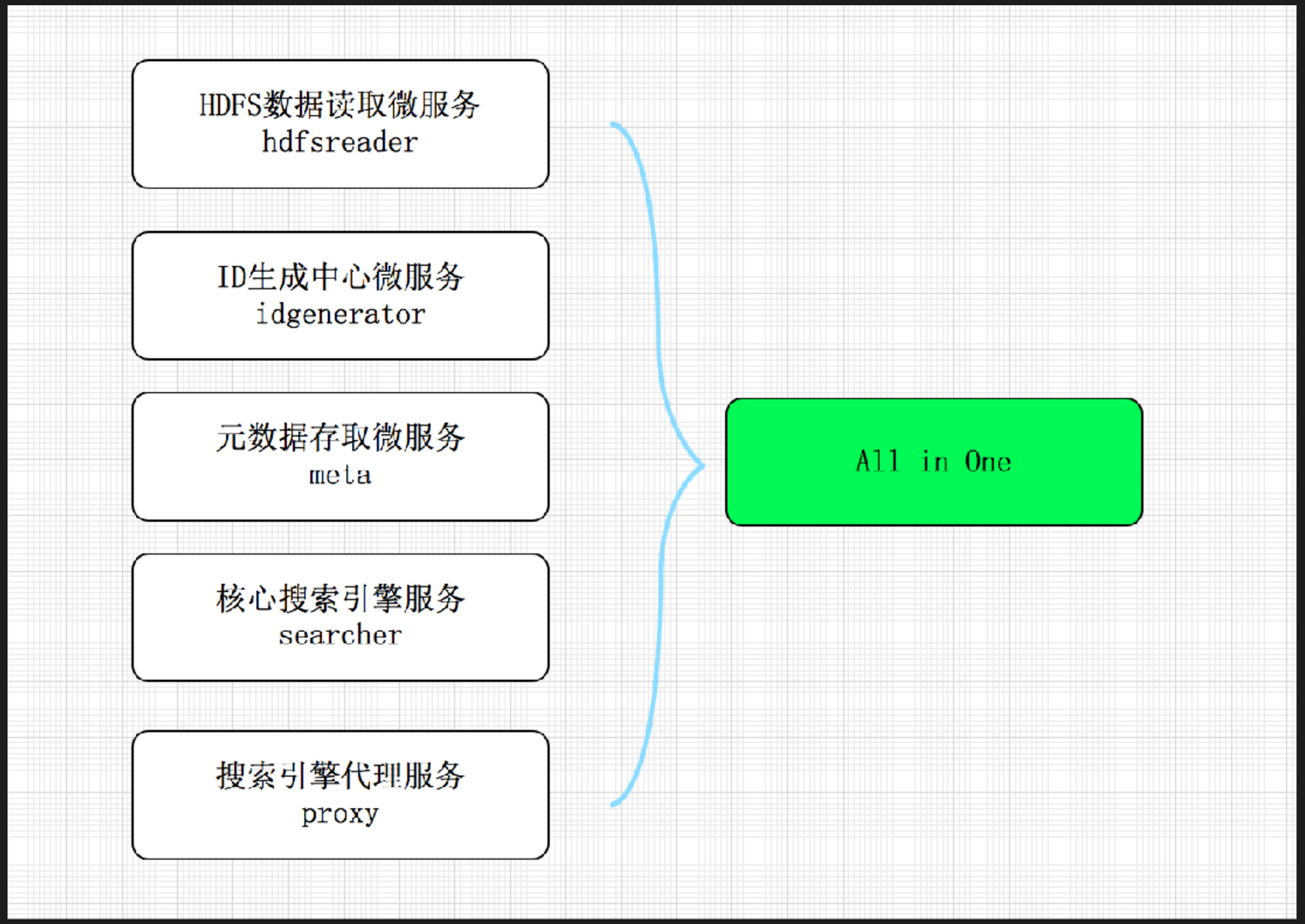
- 我们用nginx解决了访问权限问题，无需再在go程序做开发
- 利用upstream 做简单负载均衡

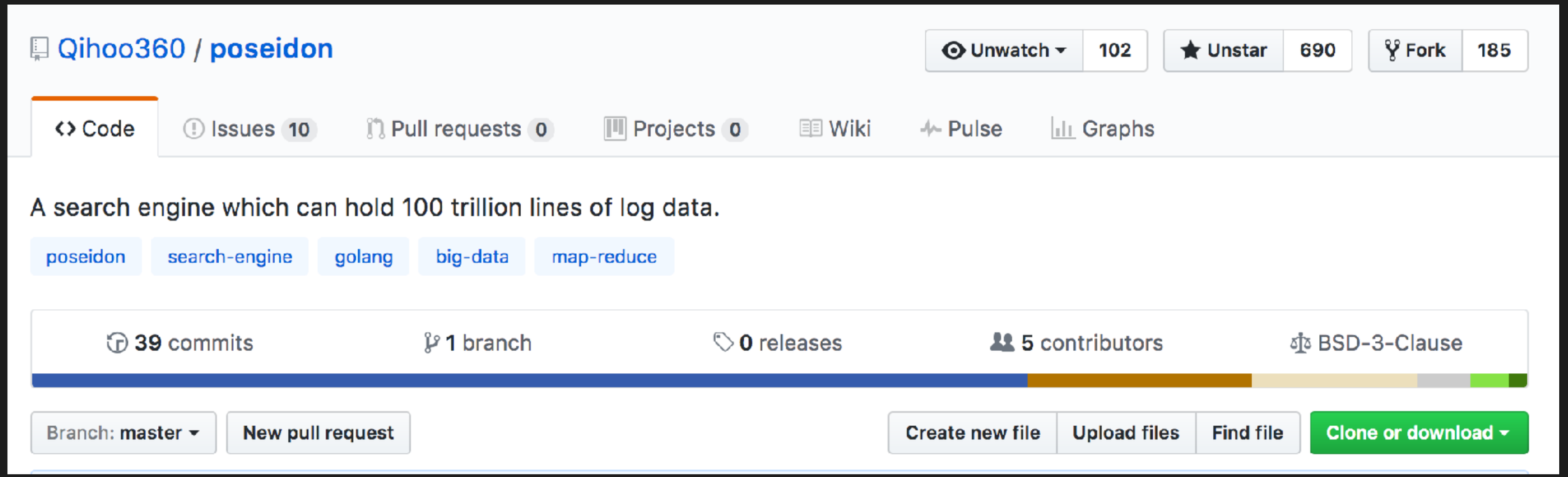
开源

- https://github.com/Qihoo360/poseidon

- Godep+vendor机制解决第三方依赖

All-In-One


HDFS数据读取微服务
hdfsreader
ID生成中心微服务
idgenerator
元数据存取微服务
meta
核心搜索引擎服务
searcher
搜索引擎代理服务
proxy
All in One


Qihoo360 / poseidon
Unwatch 102
Unstar 690
Fork 185
Code
Issues 10
Pull requests 0
Projects 0
Wiki
Pulse
Graphs
A search engine which can hold 100 trillion lines of log data.
poseidon
search-engine
golang
big-data
map-reduce
39 commits
1 branch
0 releases
5 contributors
BSD-3-Clause
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download

总结回顾

- 开发体验好，性能高，服务稳定，可以满足大部分场景的性能需求
- go语言的程序开发需要在代码可读性与性能之间做好平衡取舍，应用程序并发模型要在控制之内
- 谨慎与其他语言结合使用，即使对两种语言都很熟
- 合理引进第三方解决方案，在运维成本和系统维护成本做平衡

Q & A

谢 谢