如何打造一个百万亿级的日志搜索引擎：Poseidon

魏自立
@zieckey

个人简介

- 2009~2011年 HIPIHI 创业团队
  - 虚拟世界（VR）
  - 游戏服务器、IM系统
- 2011年~至今 奇虎360
  - /360/核心安全事业部/云引擎团队
- 在360工作期间参与设计开发的系统
  - 云查杀
  - 网盾
  - 360搜索一级引擎
  - 骚扰电话拦截

大纲

- 背景
- 需求
- 开源项目调研
- Poseidon系统设计
- 展望
- 总结

项目背景

- APT(高级持续威胁)事件追踪团队的诉求
  - 某个样本在某个时候感染了什么电脑？
  - 这个样本是从哪里来的？
  - 样本家族是什么？
  - 样本源头是什么？
- 需要在海量的日志数据中搜寻蛛丝马迹

项目需求

- 对日志数据进行快速分析和检索
- 至少保留**三年**的历史数据
- 总数据量：**百万亿**条、大小**100PB**
- 交互式的搜索响应（目标：3秒，最长60秒）
- 故障转移、自动恢复、负载均衡

开源项目调研

- Elastic Search (ES)
- HBase

Elastic Search (ES)

- 源自搜索引擎，侧重搜索与全文检索
- 偏重于中小规模的数据提供全文检索服务
- 数据规模从数亿到千亿不等
- 索引和原始数据都存放在ES集群中
- 360公司内部使用情况
  - 某团队：12台服务器的一个集群，千亿级
- 业界使用情况
  - Github：20TB，13亿个文件，1300亿行代码
  - Fog Creek：400亿行代码，30万PV/天

HBase

- 源自HDFS，侧重于存储和字段索引
- 数据规模千亿、万亿不是问题
- 当前360公司的HBase集群没有集成Lucene
- 360公司内部使用情况
  - 某团队：每天800亿增量，1个月数据

为什么放弃ES、HBase等方案

- ES 缺点
  - 不足以支撑百万亿规模的数据集
  - 数据多存放一份（原始数据仍旧在Hadoop中）
  - 自定义分词不灵活
- HBase 缺点
  - 数据格式需要转换，导致现有的数百个MR任务都要做调整
  - 是在HDFS上进行kv存储，数据膨胀，7倍
  - 自定义分词不灵活

同类型产品——腾讯Hermes

- 海量数据实时检索分析平台
- 侧重数据分析
- 数据规模从亿到万亿不等
- 没有开源
  - 只知道其索引数据也放在HDFS中
  - 没有太多细节

Poseidon设计目标

- “波塞冬”，海神，寓意海量数据的主宰者
- 数据总量
  - 百万亿条(14个0)
  - 100PB(17个0)
- 原始数据不用额外存储
- 当前的Map/Reduce作业不用变更
- 自定义分词策略
- 秒级查询响应

搜素关键技术-倒排索引

- map<TokenID, DocIDList>
- map<DocID, Doc>

某业务基础数据分析

- 每天的增量数据
  - 700亿
  - 每条日志长850字节，100+个字段
  - 60TB明文数据，压缩后 10TB
- 针对一天的数据分词（所有字段）
  - 得到去重的分词个数 100亿，总计400G
  - 每个分词平均关联 277 行日志
  - DocId个数为 27700 亿，4字节，总计： 11T
    - 进行一定编码压缩，可控制在 2TB
- 倒排索引数据 2.4TB

数据存储

- 索引应该按天、按字段类型来创建和存储
- 索引数据也应该放在HDFS
  - 其他存储不合适，因为
    - 那样相当于存储两份
    - 6个副本

数据文件格式设计思路

- 搜索引擎必须能够随机读取每一行数据
  - 原始数据文件大小：100MB~2GB
  - 普通的压缩存储，不能随机读取
- GZip压缩有一个很切合的特点
  - 多个GZip数据块拼接到一起，GZip解压程序仍然能正常解压开

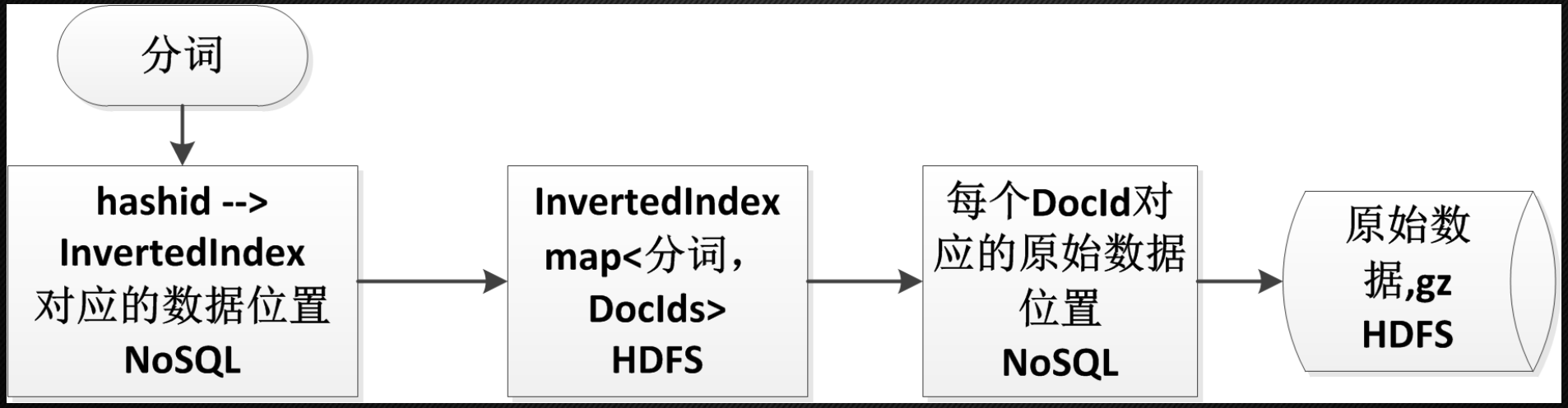
文档及ID编码设计

- 每行当做一个文档
  - 700亿，超出 int32 范围，所占空间太大
  - GZip压缩率低
- 每128行组合为一个文档
  - 5.5亿，int32 够用
  - 大大减少索引数据所占存储空间
  - 块存储，100KB，压缩后20KB
  - 较单行而言，128行的GZip压缩率高很多
  - 128可以最大限度利用 varint 编码的压缩优势

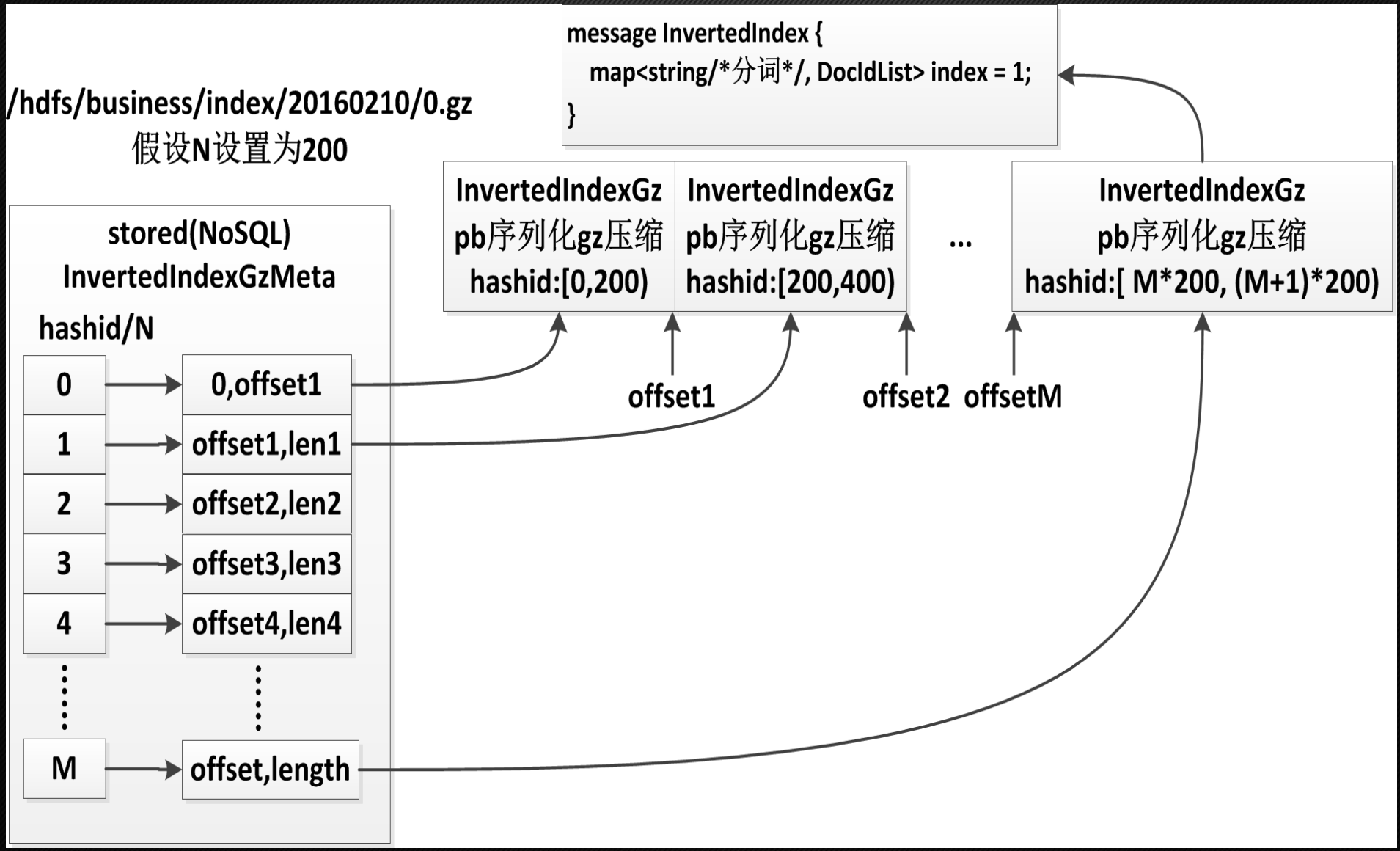
文档存储格式

- 每128行称为一个Doc
- 每个Doc进行GZip压缩，称为 DocGz
- 多个 DocGz 连接到一起形成一个HDFS文件
  - 这种文件依然能被GZip正常解压读取
- 不破坏现有的Map/Reduce作业
- 不用额外存储
- 索引数据的存储也采用类似的块存储格式

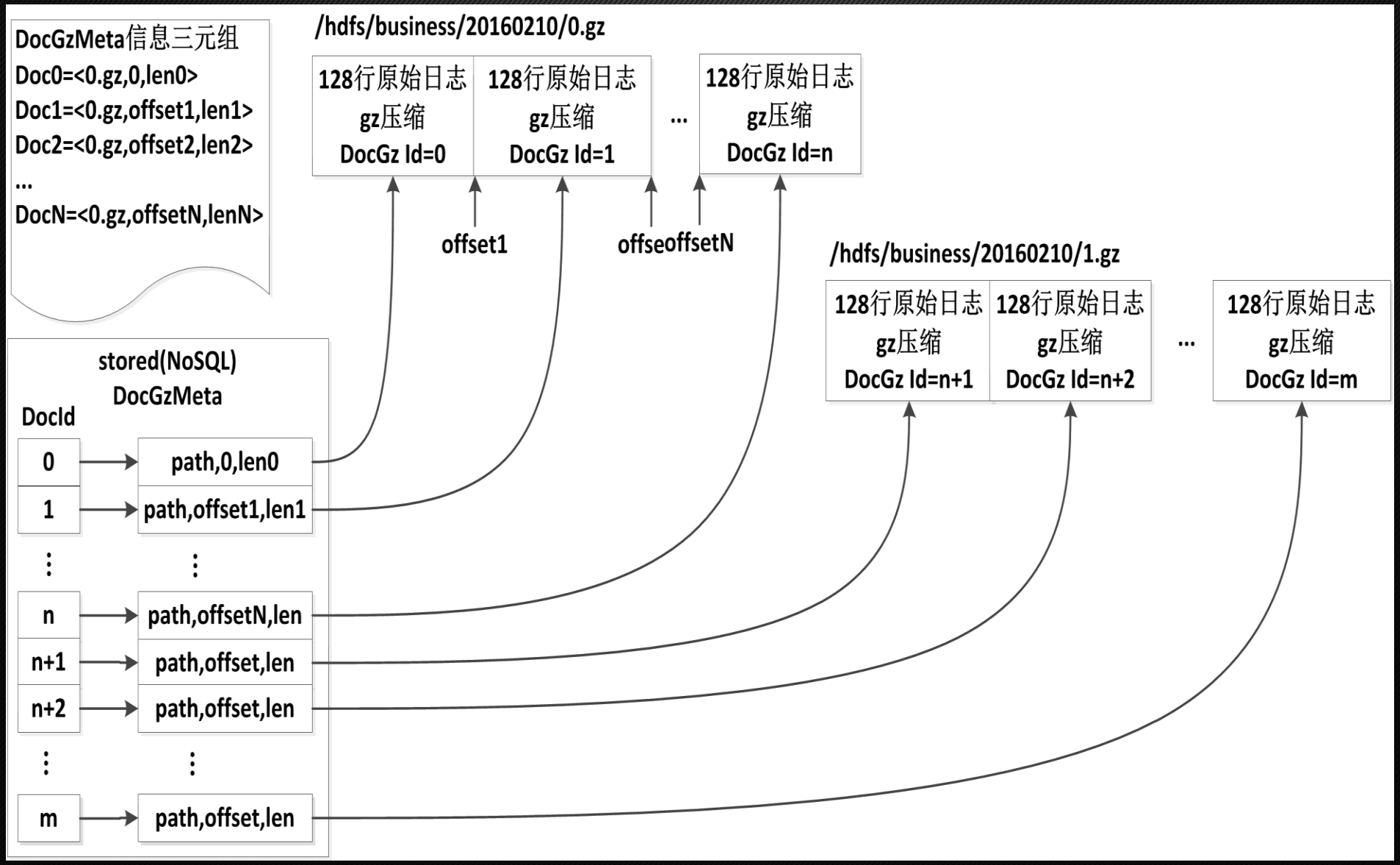
四级索引结构


分词
hashid --> InvertedIndex 对应的数据位置 NoSQL
InvertedIndex map<分词，DocIds> HDFS
每个DocId对应的原始数据位置 NoSQL
原始数据,gz HDFS

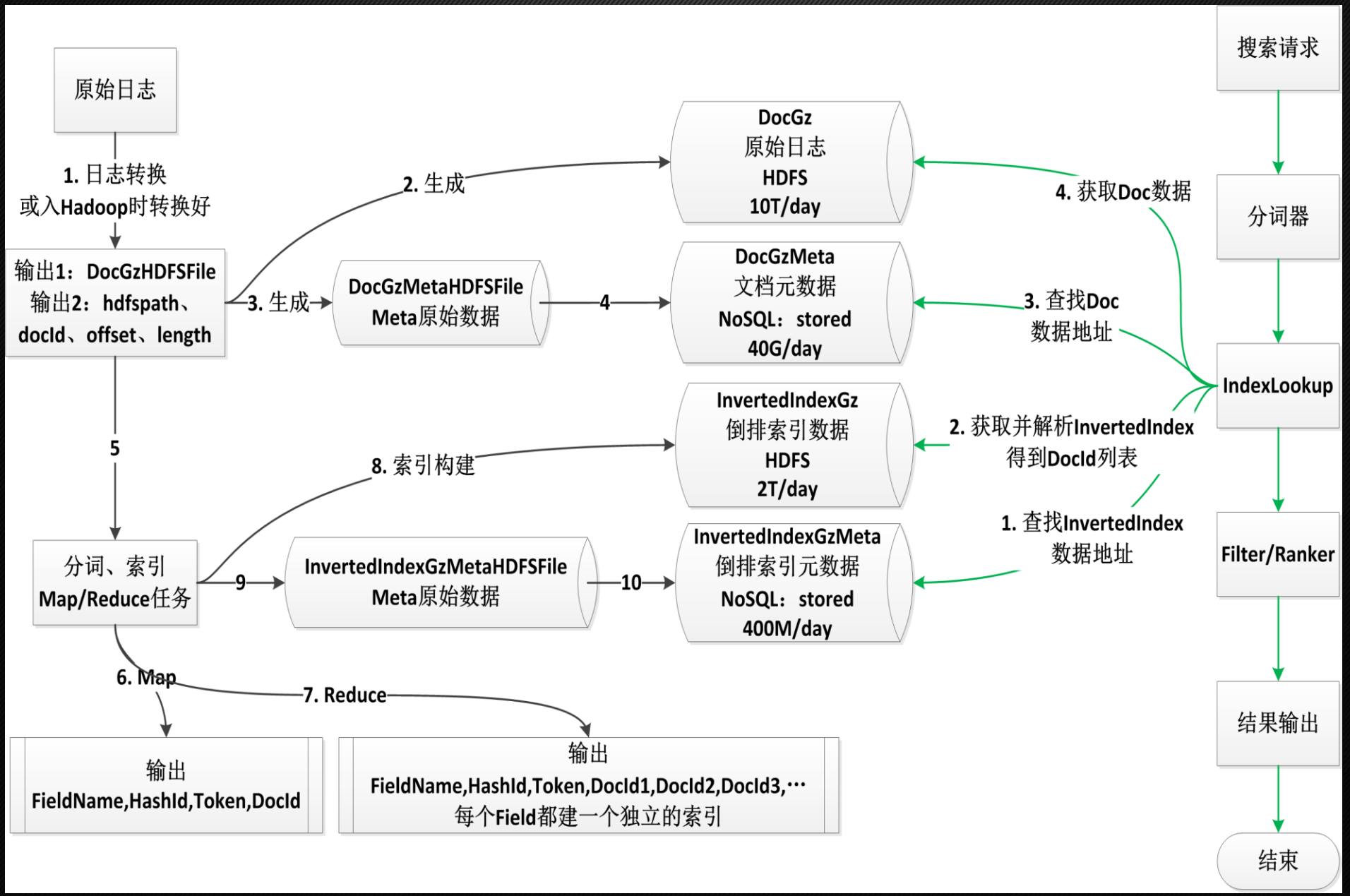
InvertedIndexHDFSFile文件格式及索引



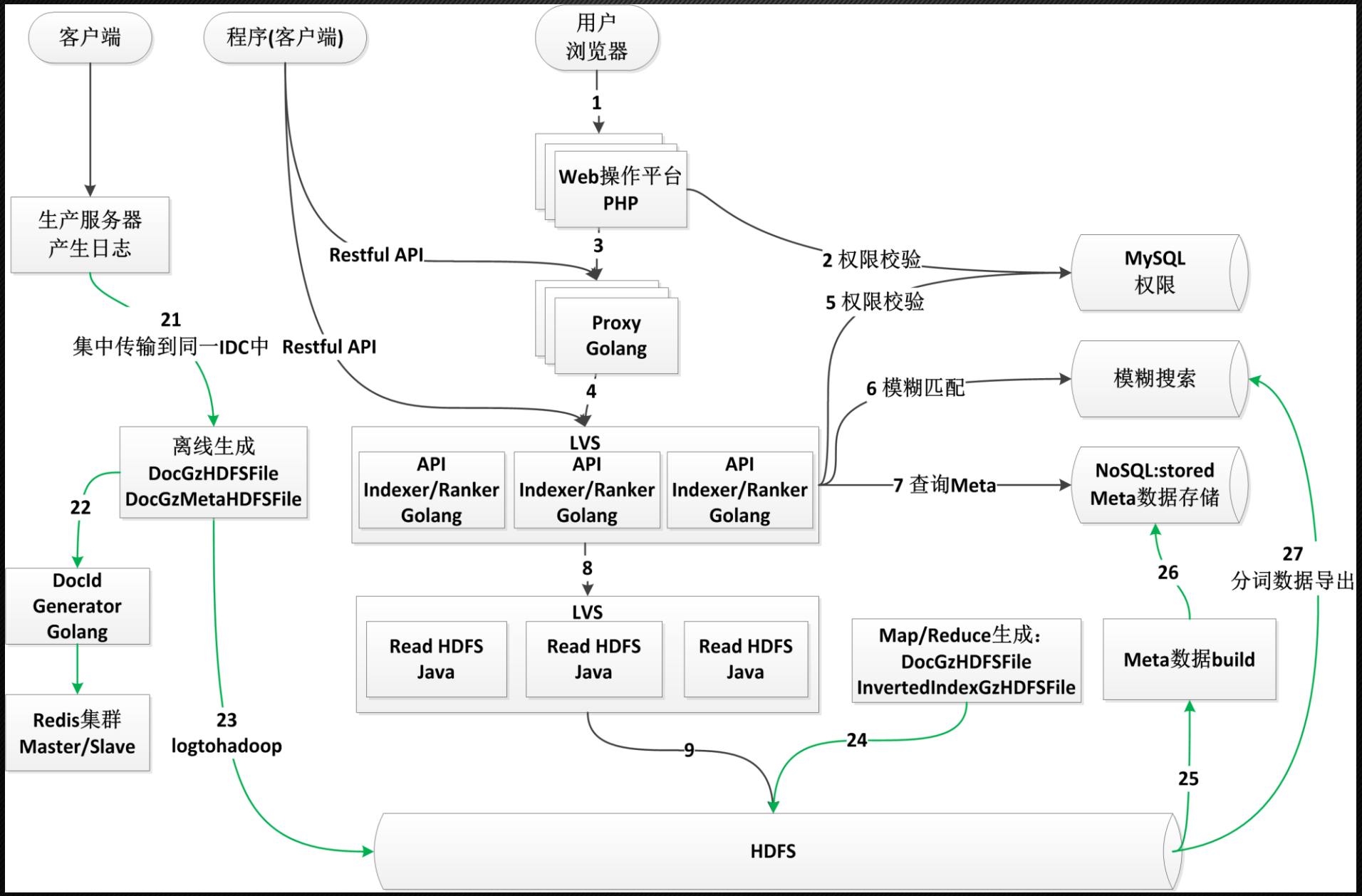
DocGzHDFSFile文件格式及索引



构建索引和搜索过程


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

架构图


客户端
程序(客户端)
用户
浏览器
1
Web操作平台
PHP
生产服务器
产生日志
Restful API
3
2 权限校验
MySQL
权限
Proxy
Golang
5 权限校验
21
集中传输到同一IDC中
Restful API
6 模糊匹配
模糊搜索
4
离线生成
DocGzHDFSFile
DocGzMetaHDFSFile
LVS
API
Indexer/Ranker
Golang
API
Indexer/Ranker
Golang
API
Indexer/Ranker
Golang
7 查询Meta
NoSQL:stored
Meta数据存储
22
8
DocId
Generator
Golang
LVS
Read HDFS
Java
Read HDFS
Java
Read HDFS
Java
Map/Reduce生成:
DocGzHDFSFile
InvertedIndexGzHDFSFile
Meta数据build
26
27
分词数据导出
Redis集群
Master/Slave
23
logtohadoop
9
24
25
HDFS

Poseidon系统现状

- 接入17个数据源
  - 日志类
  - 非日志类（将Poseidon当数据库使用）
- 每日增量数据： 1200亿条
- 数据总量超过：30万亿条
- 最近6个月新增：
  - 148名用户
  - 183538次查询请求

更多实现细节

- DocId生成器
- Map/Reduce建索引过程及填坑
- 过滤模块
- 模糊搜索子系统
- 跨时间区间查询的Proxy设计
- 权限校验系统
- Restful API设计
- Safe2加密HDFS集群设计及Poseidon搜索实现
- 分词策略：Path、URL、PE String、HEUR
- 倒排索引的压缩算法

展望

- 如果实现实时入数据实时搜索？
  - 可以做到文件级别，将实时数据放到NoSQL中
- 非日志类的大数据存储和检索？
  - 样本本身的存储
  - 样本属性信息
  - 样本pcap抓包数据
  - 杀软scan log
- 跨索引的数据关联搜索
- 开源（正在进行中...）

总结

- 不要迷信现有方案
  - 要自己动手做测试
  - ES太慢、规模上不去
  - HBase数据膨胀7倍，太浪费资源
- 系统设计方案要简单、优雅
- 激励团队
  - 有意思、有难度、有挑战
  - 也要有信心
- 争取更多的支持
  - 上级的支持
  - 团队内同事的理解和目标一致性设定
  - 寻求外部资源（系统部、DBA）

总结(续)

- 尽快与目标用户接触和互动
  - 用户的参与和反馈可以很好地激励团队
  - 让需求和目标更加明确
- 要有自己的开发节奏
  - 每两周发布一个版本
  - 将需求拆分，合理的安排到2周的版本迭代中
- 合理选择开发语言和工具
  - Java/Golang/PHP

Thanks !

Q & A