日志收集服务

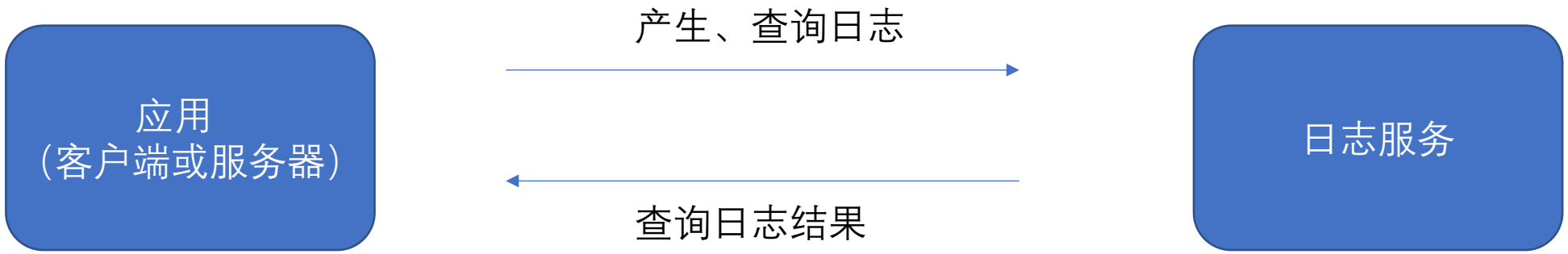
日志收集服务

By Guanyu Li

Outline

- 场景分析
- 整体架构
- Server设计
- 日志储存设计
- 性能测试
- 未完成的部分

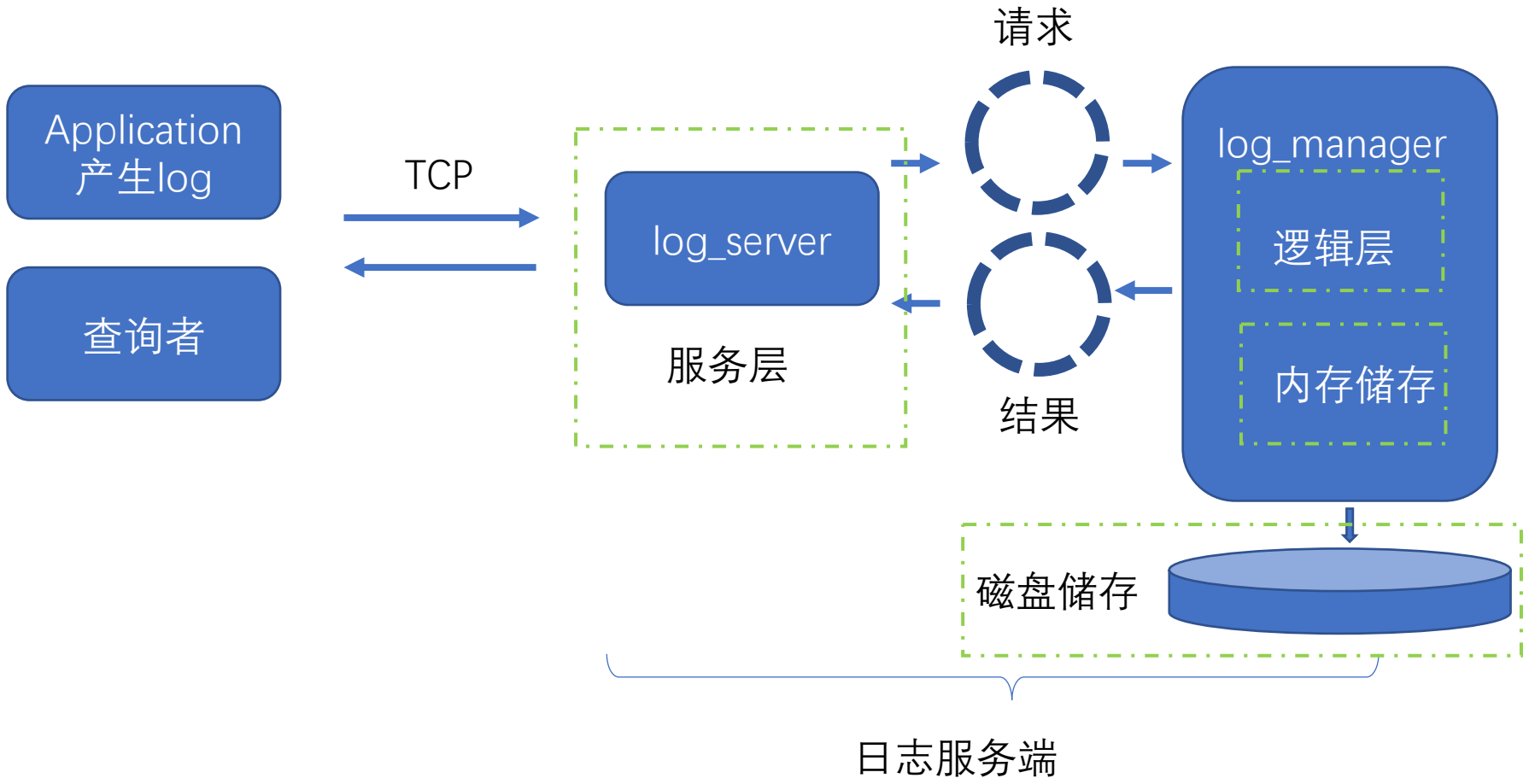
场景分析



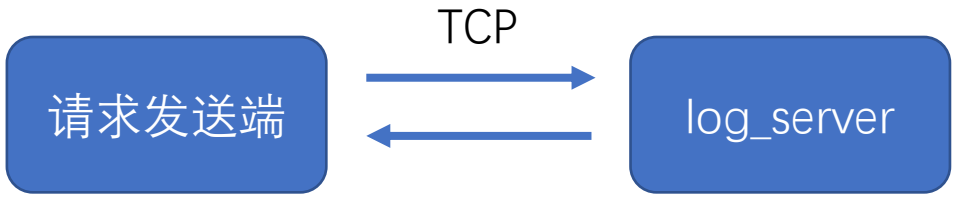
场景分析

- 应用产生日志
  - 文本，追加
- 日志查询
  - 需要建立日志数据的索引
- 日志储存：
  - 文本，追加为主
  - 日志数据需要持久化
  - 日志数据、索引的读入与释放，缓存策略

整体架构设计


Application
产生log
查询者
TCP
log_server
服务层
请求
结果
log_manager
逻辑层
内存储存
磁盘储存
日志服务端

通讯

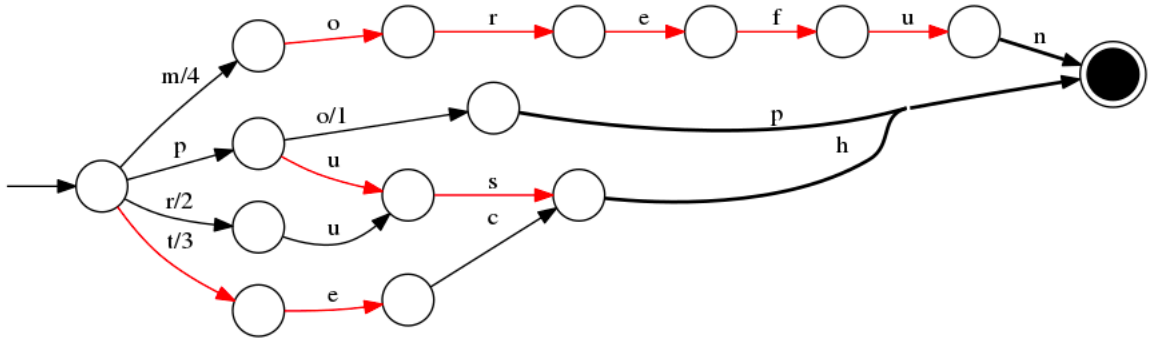


- 协议：
  - 基于TCP：可靠，即时性要求低
  - HTTP协议
    - 普遍性好
    - 解析请求行获取行为以及参数
    - append，recent，keyword，getkey，searchtime…
  - 考虑日志来源
    - 客户端：短连接，服务量大，设备可能是移动端
    - 服务器：服务器较稳定，长连接性能会好

日志服务端

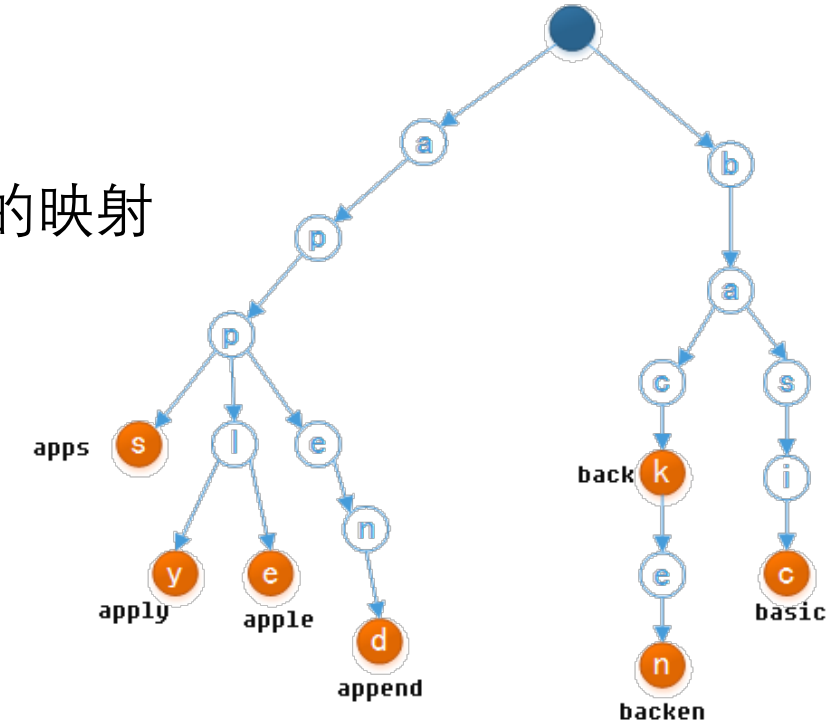
- log_server
  - Epoll 多路复用模型
  - 负责接收请求
  - 负责解析请求
  - 负责将合法任务传递给log_manager
  - 负责向查询者发送结果
    - id代表请求者客户端，构建id到session的映射确定结果的返回

日志储存前期调研



- ElasticSearch
  - 查询能力非常强，基于apache lucene
  - 利用了倒排索引，建立关键字到储存位置的映射
  - 利用FST(Finite State Transducer)减少内存使用

- 借鉴：
  - 利用倒排索引手段，构建关心字段到block、offset的映射
    - 方便进行全文检索，根据key查询
    - Key->bulk，每个bulk中key->serial
  - 索引数据结构：字典树
    - 共享前缀，插入时复杂度为o(n)
    - 复杂度稳定，没有碰撞
    - 比FST简单一些



日志储存前期调研

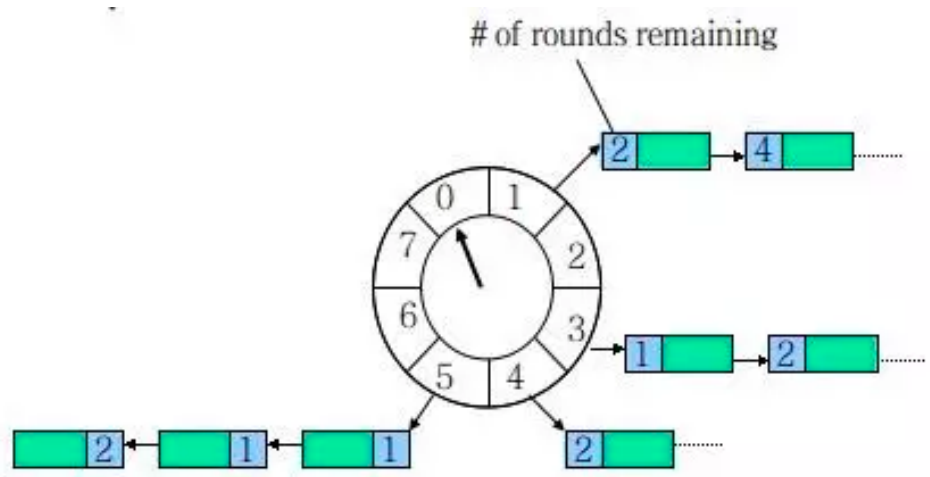
- Kafka
  - 分段储存，不仅保存日志文件，还保存索引文件
  - 用文件中第一条日志的序列号命名，方便定位某序列号的日志
  - 稀疏索引

- 借鉴：
  - 索引也保存入磁盘，查询时读入
  - 命名采用了Kafka的风格，用文件中第一条日志的序列号命名
    - 在重建时可能有意义（未实现）
  - 稀疏索引可以节省索引文件的空间（未实现）

日志储存前期调研

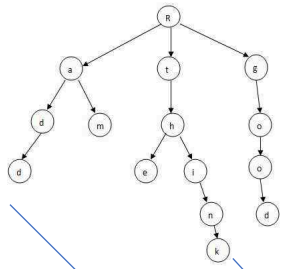
- 需求：需要定时器做定时任务
  - 希望能定时持久化日志文件
  - 希望读入索引后能定时驱逐，起到一定的缓存的作用

- 时间轮
  - O(1)的插入、删除效率
  - 用户态实现
  - Kafka也采用了时间轮



日志储存

Key -> Bulk字典树

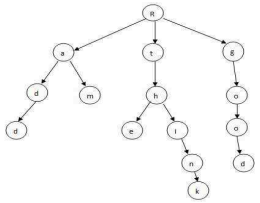


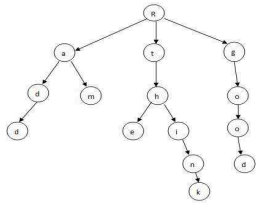
log 1
log 2
log 3
...

Offset索引
(每条日志的offset,
序列)

Timestamp索引
(每条日志的ts,
序列)

Key字典树





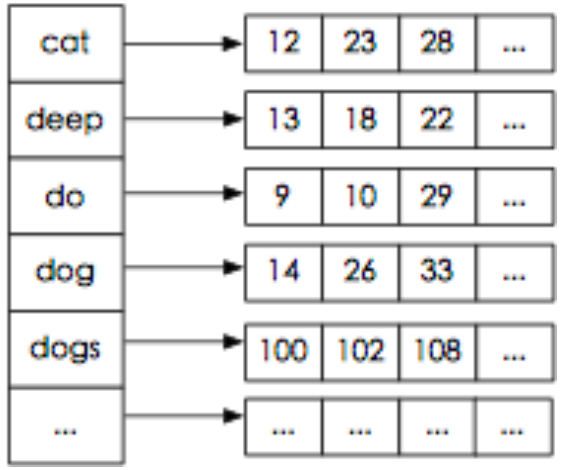
Keyword字典树

日志文件

索引文件

(将索引序列化)

查询key & 时间



1. 查询key对应的bulk
2. 查询时间范围内的bulk
   - 二分法，记录bulk的最大最小时间
3. 查询bulk中的key对应的日志条目序号
4. 根据序号定位在所选时间内的日志
   - 利用timestamp索引进行二分查找
   - 假设：同一key的日志上传到server的是按时间顺序的
5. 收集结果返回

不安全 | 10.85.4.241:5000/recent?count=100&id=123

总5000W条下

```
Result use time 73 microseconds
[key: Mh6FLkWqV7, time: 16666667] I must make merry before the Spring is spent.
[key: LOgv86LNzU, time: 16666667] Yet with the moon as friend and the shadow as slave
[key: pipITStA7E, time: 16666667] Listless, my shadow creeps about at my side.
[key: 7ZlnsRIwaZ, time: 16666666] The moon, alas, is no drinker of wine;
[key: UobAj488Ex, time: 16666666] For he, with my shadow, will make three men.
[key: ELMrf5Ytz, time: 16666666] Raising my cup I beckon the bright moon,
[key: ZcURnFquFl, time: 16666665] I drink alone, for no friend is near.
[key: xgUDUKu4jv, time: 16666665] A cup of wine, under the flowering trees;
```

73微秒 VS 5600微秒
是否缓存中

不安全 | 10.85.4.241:5000/searchtime?key=tA7ELMrf5Y&start=2000&end=3000&id=123

```
Result use time 967 microseconds
[key: tA7ELMrf5Y, time: 2003] He has endeavoured to prevent the population of these States; for that purpose obstru
[key: tA7ELMrf5Y, time: 2009] He has plundered our seas, ravaged our coasts, burnt our towns, and destroyed the liv
[key: tA7ELMrf5Y, time: 2014] I watch the Moon, then bend my head
[key: tA7ELMrf5Y, time: 2027] So saying, I was drunk all the day,
[key: tA7ELMrf5Y, time: 2033] For he, with my shadow, will make three men.
[key: tA7ELMrf5Y, time: 2051] He has refused his Assent to Laws, the most wholesome and necessary for the public good.
[key: tA7ELMrf5Y, time: 2057] For depriving us in many cases, of the benefit of Trial by Jury:
[key: tA7ELMrf5Y, time: 2069] He has refused for a long time, after such dissolutions, to cause others to be electe
```

967微秒 VS 11768微秒
是否缓存中

不安全 | 10.85.4.241:5000/search?keyword=for%20our%20most&id=123

```
Result use time 68228911 microseconds
[key: Aj488ExgUD, time: 8] For taking away our Charters, abolishing our most valuable Laws and alterir
[key: vlKLOgv86L, time: 24] For taking away our Charters, abolishing our most valuable Laws and alter:
[key: vtxvccMh6F, time: 41] For taking away our Charters, abolishing our most valuable Laws and alter:
[key: 88ExgUDUKu, time: 58] For taking away our Charters, abolishing our most valuable Laws and alter:
[key: nsRIwaZcUR, time: 74] For taking away our Charters, abolishing our most valuable Laws and alter:
[key: vccMh6FLkW, time: 91] For taking away our Charters, abolishing our most valuable Laws and alter:
[key: xgUDUKu4jv, time: 108] For taking away our Charters, abolishing our most valuable Laws and alter
[key: IwaZcURnFq, time: 124] For taking away our Charters, abolishing our most valuable Laws and alter
```

返回结果100W条时
68秒

需要考虑极端情况，例如可以设置一个最多搜索的阈值

内存控制策略

- 额外的线程进行内存的持久化与淘汰工作

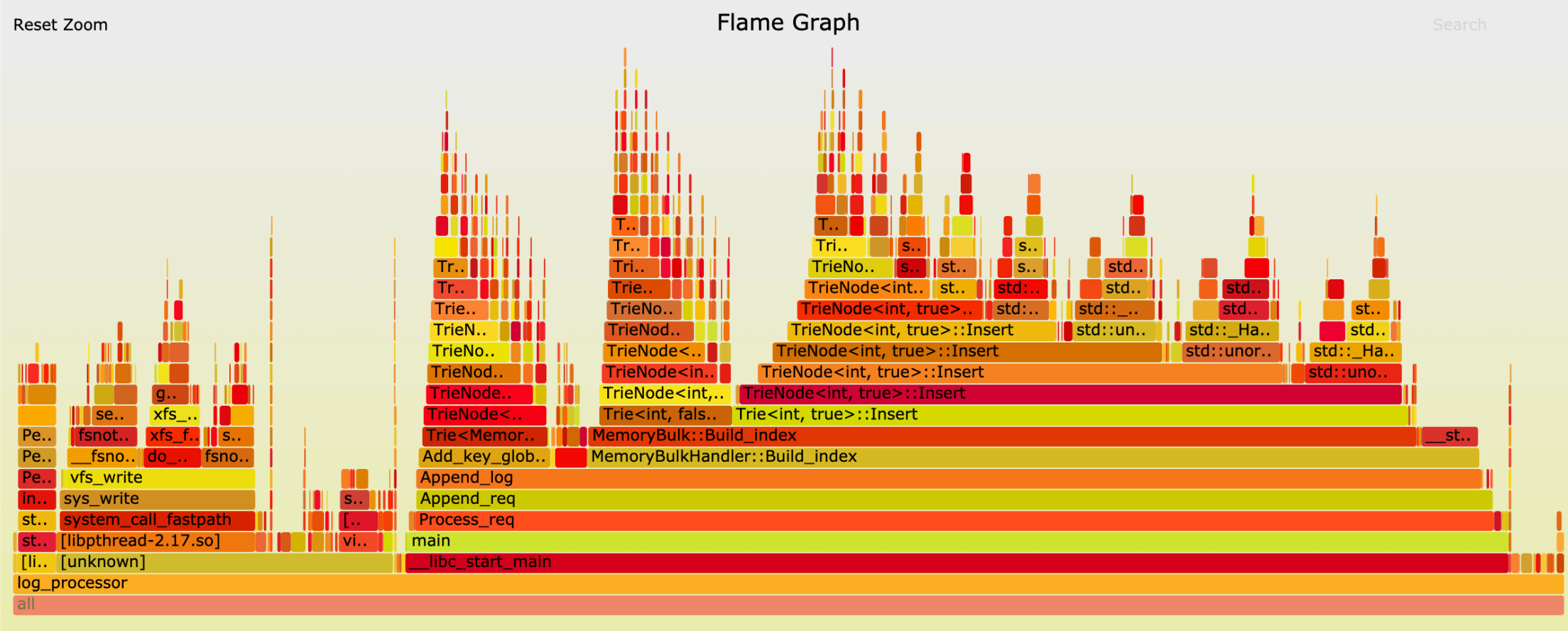
- 持久化策略
  - 内存中的bulk使用环形队列，根据已使用比例释放
    - 内存复用，避免每次为新Bulk进行内存分配
    - 持久化线程定时检查已用的槽位是否超过阈值，释放部分
  - 定时持久化bulk
    - 每次append，视为hot，重新插入定时持久化事件，延长寿命

- 淘汰策略
  - 查询时根据种类读入索引，根据需要读入日志data
  - 定时淘汰被重读入的日志和索引
  - 若再次查询，视为hot，重新插入定时淘汰事件，延长寿命

性能

- 测试环境
  - 6线程， Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz，docker
- 测试样本
  - 30种key随机选取(10字节~3字节)
  - 日志来自古诗和独立宣言
    - 单条32~85字节，50种反复发送共5000W条
  - 100ms检测一次，每个bulk 2Mb，环形队列64个槽，阈值48
- 效果：
  - 长连接：72150 QPS，瓶颈在log_manager
  - 短连接：25162 QPS，瓶颈在log_server

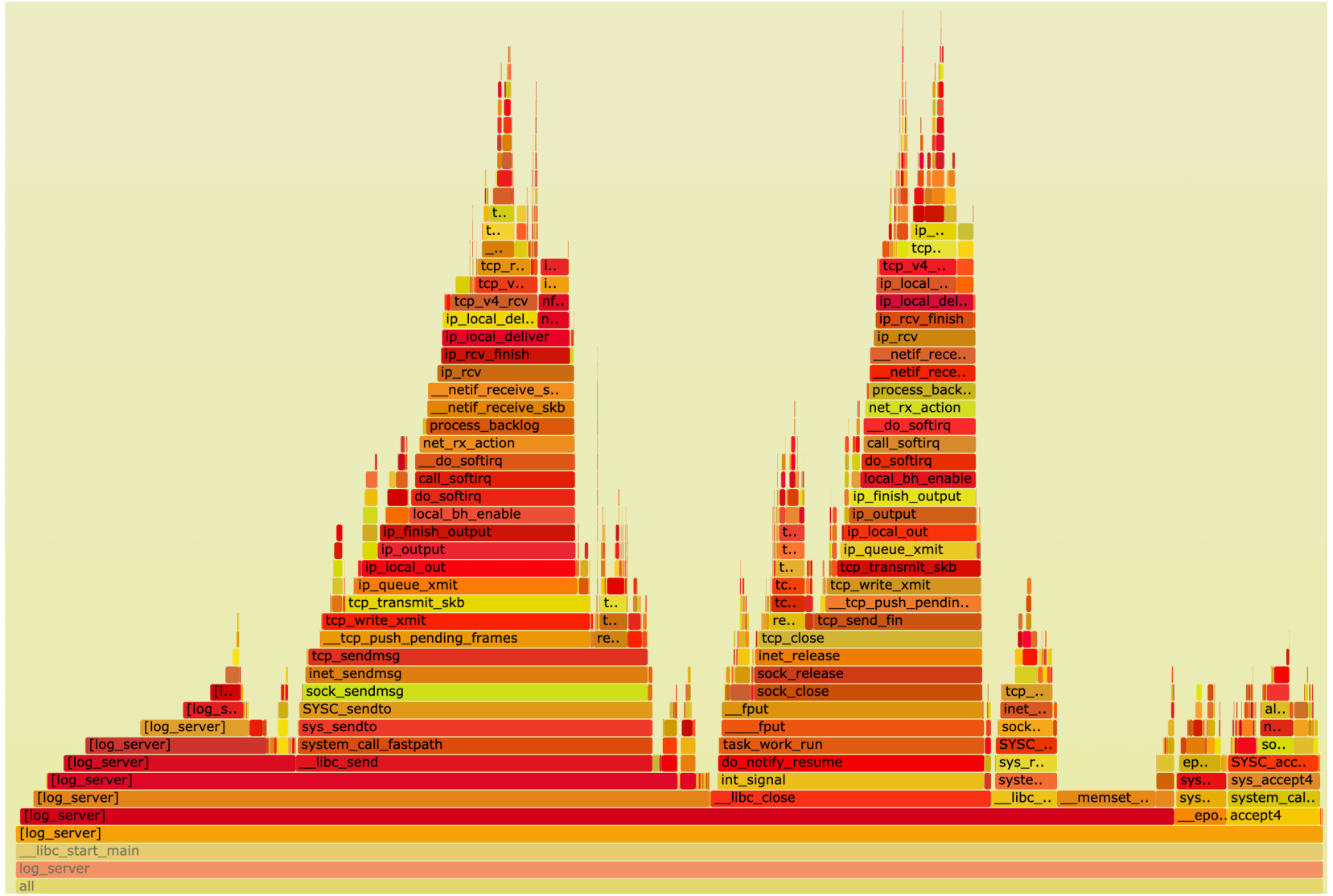
性能分析 长链接



长连接时性能都消耗在建立索引

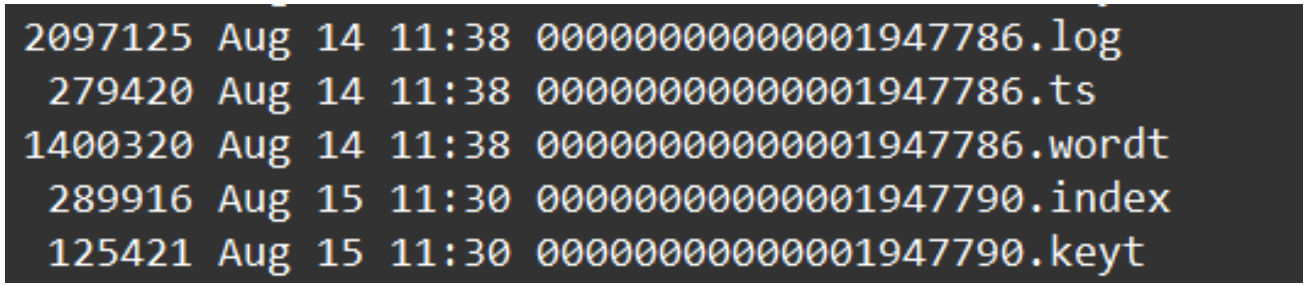
- 考虑改进数据结构
- 考虑利用线程异步建立索引

性能分析：短连接



时间在send、close等系统调用上

文件大小


2097125 Aug 14 11:38 00000000000001947786.log
279420 Aug 14 11:38 00000000000001947786.ts
1400320 Aug 14 11:38 00000000000001947786.wordt
289916 Aug 15 11:30 00000000000001947790.index
125421 Aug 15 11:30 00000000000001947790.keyt
日志文件
索引文件

还需要压缩

需要改进的点

- 数据结构
  - 更优秀的空间压缩度可以更高
    - Fst，双数组Trie树
  - 索引储存可以采用一些更紧凑的方式
    - 稀疏索引
    - 仅储存增量
- 重启后重建
- 多备份等

日志收集服务

- Server
  - Epoll
- 日志储存
  - 倒排索引，快速搜索
  - 持久化策略与缓存淘汰策略