Feed系统结构浅析

人人网 张铁安

Feed系统的定位及功能描述

- 是SNS的核心功能
- 是SNS网站中用户信息的扩散传播通道
- 需要很高的实时性
- 与各业务系统联系紧密（input & output）
- 高效、稳定、抗压力强
- 系统的复杂度高

面临的挑战

- 用户产生的数据量巨大
  - 假定按平均1000条/秒计算，用户每天产生近亿条数据
- Feed的扩散范围大（从几个人到几百万人）
- 合并、去重、排序规则复杂，要求实时，响应快速
- 用户请求量大
- 根据各业务的需要，提供个性化的筛选策略

关于Push Or Pull 的思考

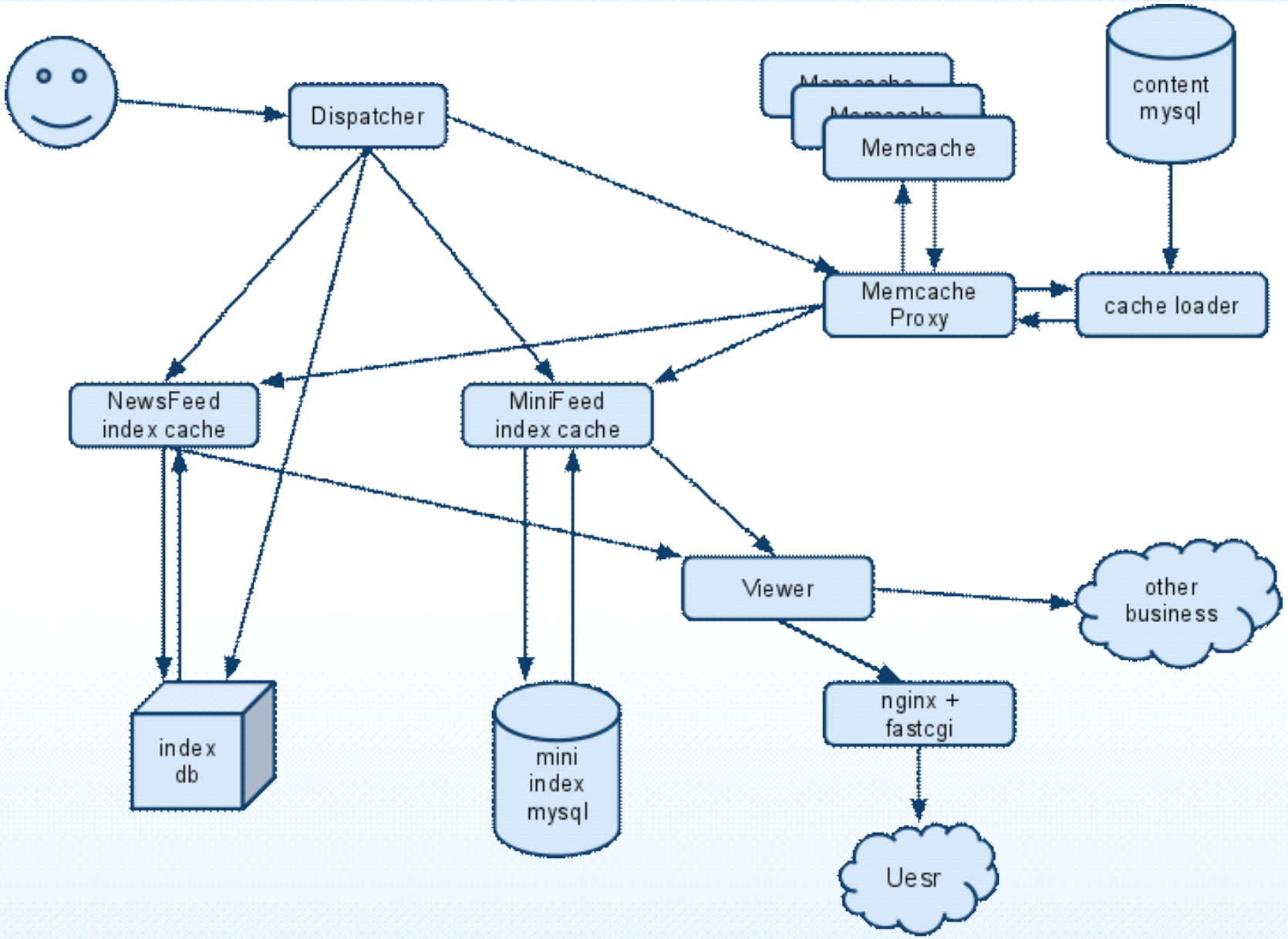
人人网 renren.com

- 获取数据的两种方式
  - 推模式
  - 拉模式
- 结论
  - 从查询的效率考虑，推模式更合适

Feed System构成

- Dispatch
- NewsFeed Index Cache
- User interaction feedback
- Sorting algorithm & Friend Rank
- MiniFeed Index Cache
- FeedContent Cache
- NewsFeed Index Persistence (index db)
- Rendering engine (data + template)

系统结构图


Dispatcher
Memcache
content
mysql
Memcache
Proxy
cache loader
NewsFeed
index cache
MiniFeed
index cache
Viewer
other
business
index
db
mini
index
mysql
nginx +
fastcgi
Uesr

技术细节

- Feed的分发系统
- Feed的Cache系统
  - Index Cache
  - Content Cache
  - 数据压缩
- Index的持久化存储系统
- 页面显示用的渲染引擎
- 基于内容及用户行为反馈的排序算法（略）

关于Open Source

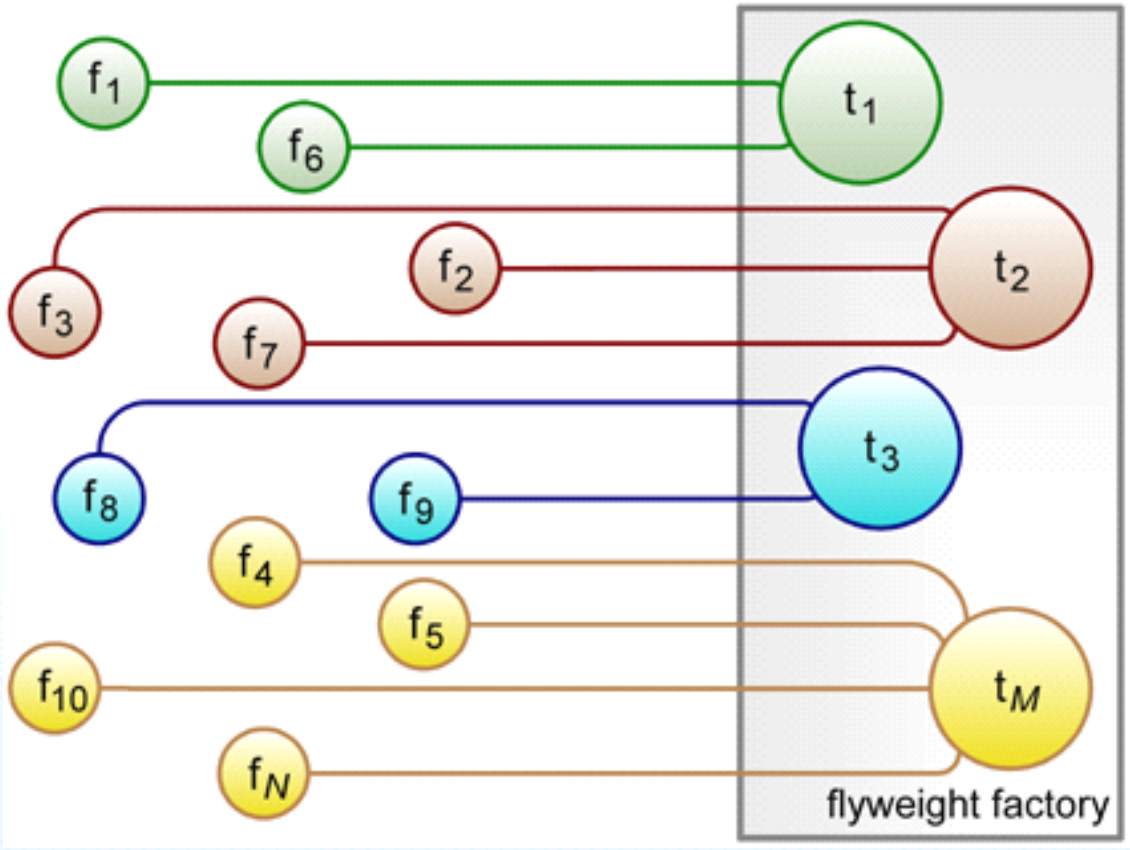
- Feed系统中使用的OpenSource项目
  - ICE（通信框架）
  - Mysql（DB）
  - Memcache + libmemcached（Content 内存Cache）
  - Google Protobuf （对象的序列化及反序列化）
  - Quicklz （二进制数据压缩）
  - Boost multi-index container（多索引结构）
  - Tokyo Tyrant （key-value存储引擎）
  - Google Ctemplate （数据的模板渲染引擎）
  - Nginx + FastCgi （WebServer）

Feed的分发系统

- 数据的拆分
  - Index + content
- 收消息用户列表的Cache策略
  - LRU & Update Notify
- 异步线程池
  - 合理设置线程个数解决脉冲式请求

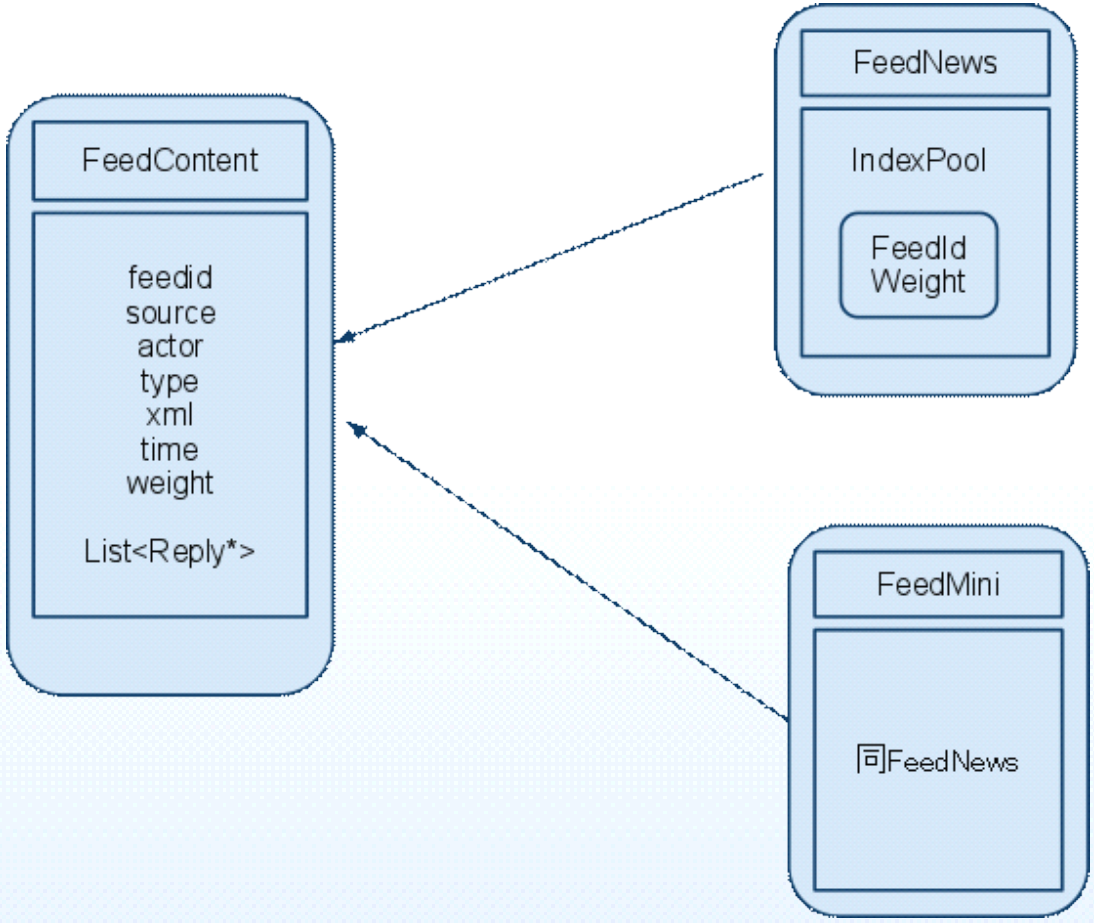
Feed Cache的内存优化

- FlyWeight的设计思想



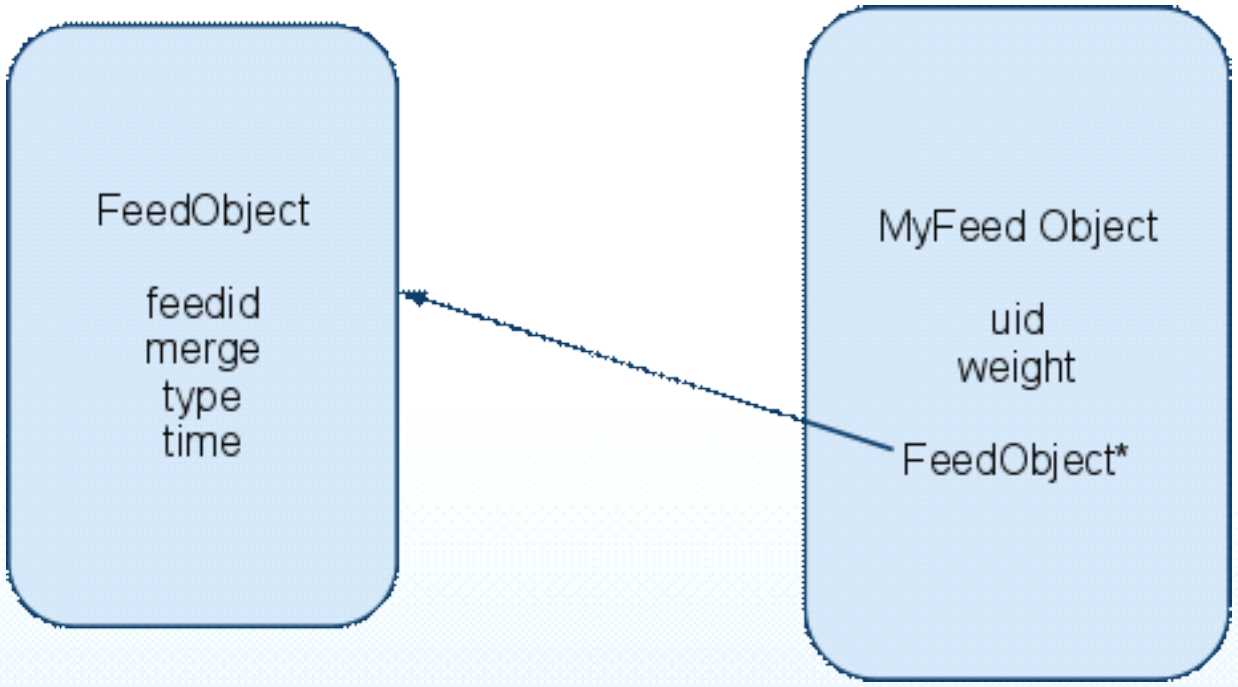
基于FlyWeight思想的Cache结构

- FeedContentCache & Index Cache服务间的FlyWeight



Index服务内步的FlyWeight结构

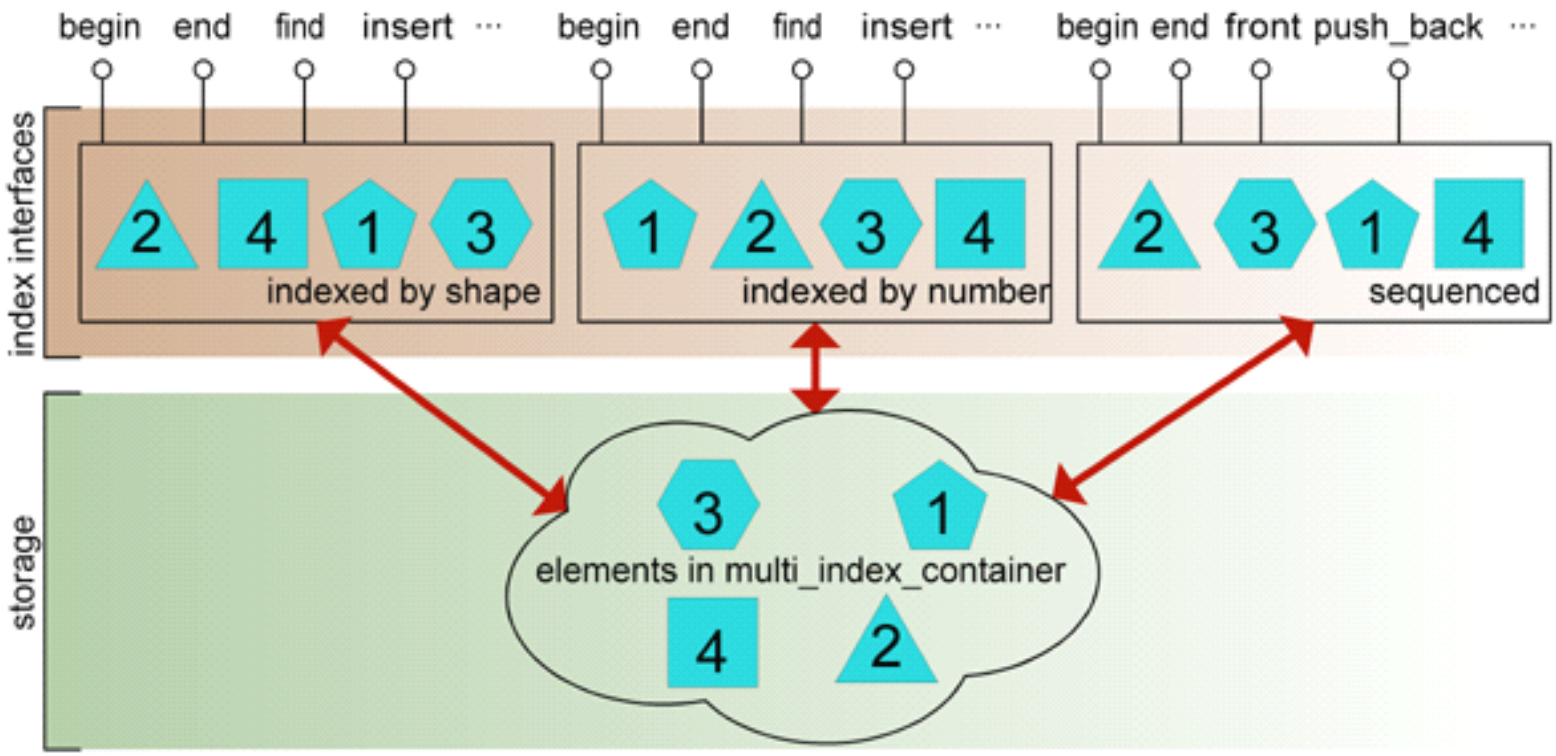
- FeedNews服务内部的FlyWeight



Index Cache的多条件查询

- 利用multi_index支持类似数据库的查寻方式，对同一个数据集，可以按不同的维度建立索引，方便支持不同条件的查询，同时对于排序结果，可以做到实时的更新

Boost Multi Index Container



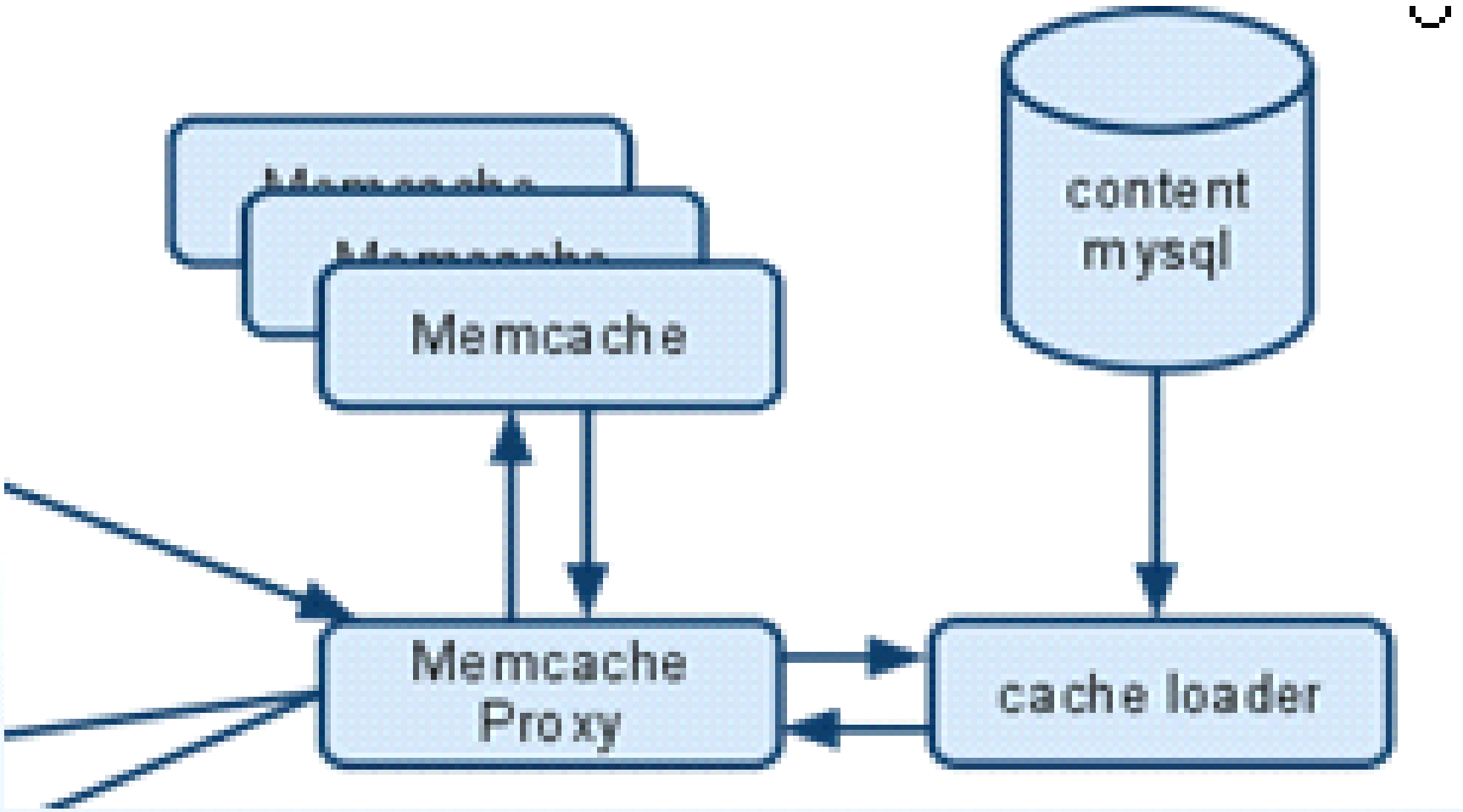
关于内存的压缩存储

- 各种压缩方法
  - zlib
  - lzo
  - fastlz
  - lzf
  - quicklz
- 对象序列化及压缩
  - Protobuf + quicklz

Library	Level	Compressed size	Compression Mbyte/s	Decompression Mbyte/s
QuickLZ C 1.4.0	1	47.9%	262	283
QuickLZ C 1.4.0	2	42.3%	111	203
QuickLZ C 1.4.0	3	40.0%	23	446
QuickLZ C# 1.4.0	1	47.9%	122	99
QuickLZ Java 1.4.0	1	47.9%	115	86
LZF 3.1	UF	54.9%	175	363
LZF 3.1	VF	51.9%	172	351
FastLZ 0.1.0	1	53.0%	179	392
FastLZ 0.1.0	2	50.7%	158	389
LZO 1X 2.02	1	48.3%	151	401
zlib 1.22	1	37.6%	43	152

- 我们对内存Cache的要求
  - 支持高并发
  - 在内存容量不断增加的情况下，查询性能不会有大的降低
  - 易于扩容及高可用性（一致性哈希）
  - 统一的配置管理，使用简单

Memcache集群



索引的持久化系统

- 索引持久化的原因
  - 解决索引的内存Cache重启后无法快速恢复的问题
  - 利用相对便宜的存储介质为用户尽量保存多一些内容
- 需要解决的问题
  - 每天近60亿条索引的持久化存储(5w+ write/s)
- 传说中的解决方案
  - Mysql？（最高1K query/s）
  - Open Source key-value db？(还是不够快)
  - GFS？ (听说Google有，但是光盘没有卖的)

索引的持久化系统 —— 五花八门的key-value DB

项目名称	语言	容错性	持久性存储介质	客户端协议	数据模型	文档	赞助商/社区
Project Voldemort	Java	分区，复制，read-repair	Pluggable: BerkleyDB, Mysql	Java API	Structured / blob / text	A	Linkedin, no
Ringo	Erlang	分区，复制，immutable	Custom on-disk (append only log)	HTTP	blob	B	诺基亚, no
Scalaris	Erlang	分区，复制，paxos	In-memory only	Erlang, Java, HTTP	blob	B	OnScale, no
Kai	Erlang	分区，复制?	On-disk Dets file	Memcached	blob	C	no
Dynomite	Erlang	分区，复制	Pluggable: couch, dets	Custom ascii, Thrift	blob	D+	Powerset, no
ThruDB	C++	复制	Pluggable: BerkleyDB, Custom, Mysql, S3	Thrift	Document oriented	C+	Third rail, unsure
CouchDB	Erlang	复制，分区?	Custom on-disk	HTTP, json	Document oriented (json)	A	Apache, yes
HBase	Java	复制，分区	Custom on-disk	Custom API, Thrift, Rest	Bigtable	A	Apache, yes

索引的持久化系统 —— Feed Index DB

人人网 renren.com

需要解决的难题

- 数万级的每秒写入
- 每秒几千次的随机读
- 每天100G+的新增索引数据

索引的持久化系统 —— Feed Index DB

- 解决思路
  - 常规办法对于每秒几万次的写入，除了堆几十或上百台机器，别无它法。测试结果：做Raid5的机器，在完全随机写的情况下，IOPS也就能到800+
  - 如果我们将随机写改为顺序写文件，写入效率会高出很多
  - 需要充分的利用内存，在内存中将写入的随机索引进行整理和积攒，再顺序的写入硬盘
  - 由于使用了延迟写入内存的方式，需要在Log中记录所有操作，方便出问题时能找回内存中的数据
  - 使用异步Direct IO，不要让OS多管闲事，浪费内存
  - 选用更牛B的硬盘，我们用的是SSD

索引的持久化系统 —— Feed Index DB

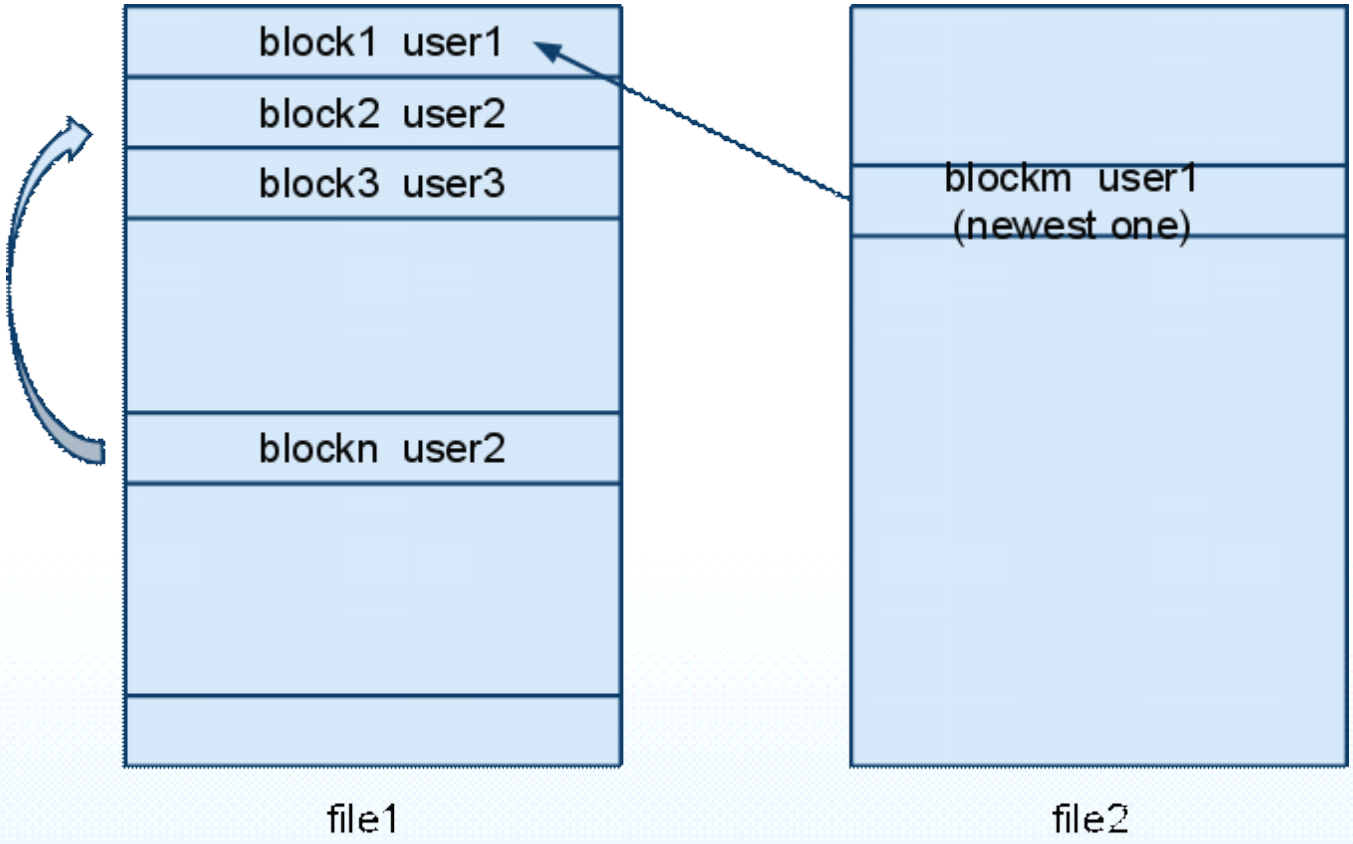
人人网 renren.com

- 解决方案
  - 合并写操作
  - 通过Log保证Down机后数据恢复
  - 使用TT保存索引
  - 使用异步IO读写文件
  - 使用Direct IO屏蔽OS的Cache策略
  - 使用SSD解决大量的并发读取

索引的持久化系统 —— Feed Index DB结构

- Index Node
  - 责任存储 UserID 到 最新一块Data Block的位置信息
  - 使用Tokyo Tyrant保存key-value对应关系
  - 因为数据量很小，所以TT很好用
- Data Node
  - 异步的Direct IO
  - 每个用户可以分配N个Block，每个Block占2K大小，N个Block首尾相连，很像一个单向链表

索引的持久化系统 —— Data File 结构



模板的渲染引擎及展示

人人网 renren.com

- 数据格式的一致性
  - 由于Feed的输入很多，自来各个不同业务，需要保证数据格式的一致性，输出时，通过渲染引擎将数据转化为不同的View，提供给各业务
- 技术方案
  - Ctemplate提供高效的模板渲染能力
  - Nginx+FastCgi提供高并发的Web服务

想了解更多有趣内容，欢迎加入我们的开发团队

人人网 renren.com

- 我们正在努力做好的事情
  - Feed System（新鲜事系统）
  - IM Server （人人桌面服务器端）
  - Ad Engine（广告引擎系统）
- 感兴趣的快来报名
  - Linux C++开发工程师（3~5人）
- Email：
  - antonio200@gmail.com
  - jobs@opi-corp.com

End

thanks