BeansDB 设计与实现

davies.liu@gmail.com
Douban Inc.
2010-12-28

BeansDB是什么?

- Key-Value 数据库
- 分布式的，伸缩性比较好(P)
  - 性能和容量
- 最终一致的(C)
  - 可能出现短时间内的数据不一致
- 高可用的(A)
  - 部分节点出现故障不影响服务

BeansDB 不是什么?

- 事务型关系数据库
  - 有严格数据一致性需求的多个对象
- POSIX文件系统
  - 不要放过大的对象
  - 不能访问对象的部分内容
- 缓存系统
  - 性能不是那么高，与内存是否充足有关
- CDN等等

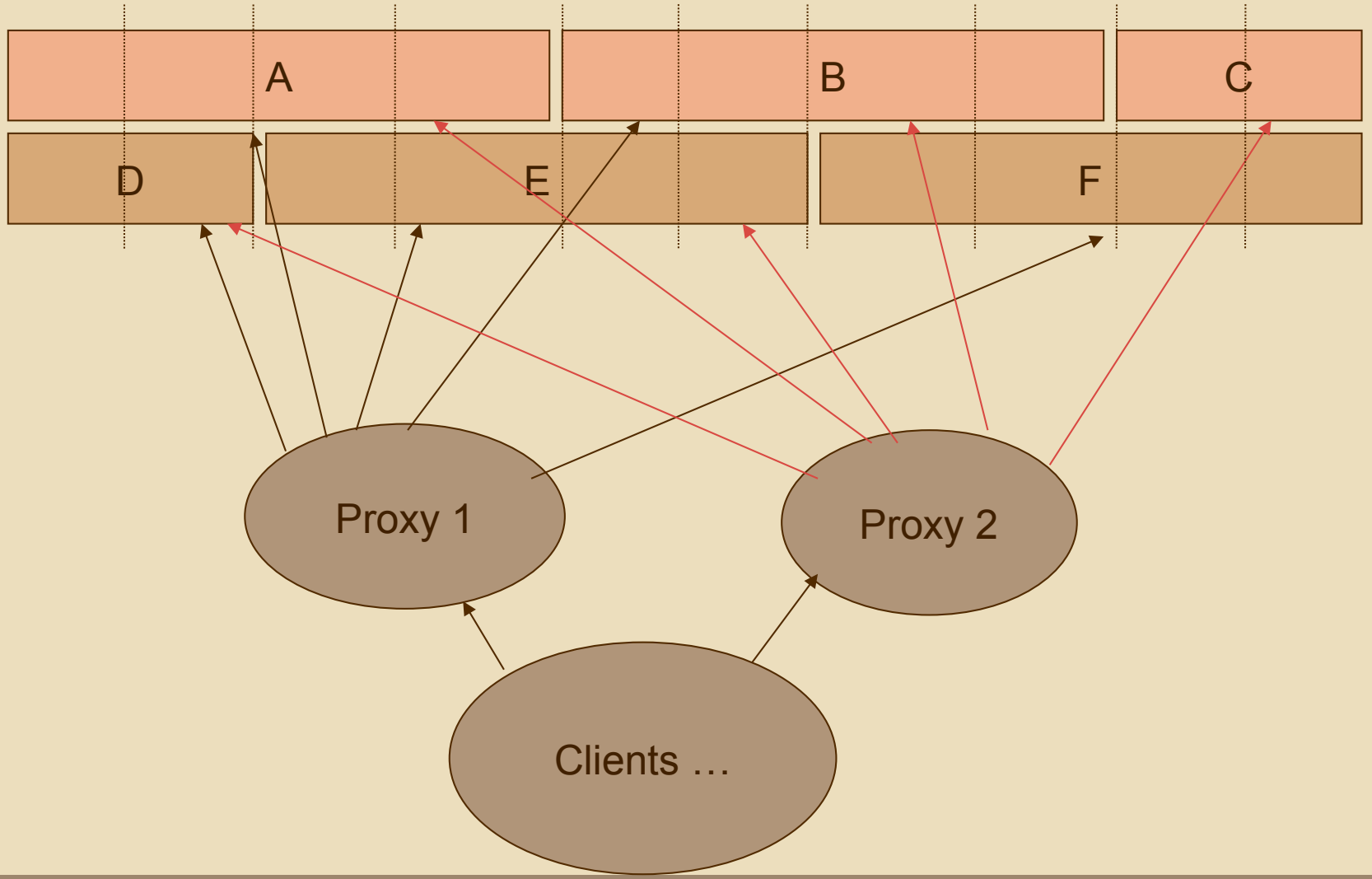
典型用法

- 图片文件，小媒体文件(mp3)
- 大文本字段(通常> 1kb)
- profile , properties
- 数据体系中的叶子部分
  - 不是其它数据依赖的关键

非典型使用

- 大量过于零碎的小内容(<100 byte)
  - 收藏数据
  - 广播数据
  - 通常需要批量查询
- 非在线使用数据
  - 用户上传的原始图片
  - 日志文件
- 中间结果
  - 矩阵计算的中间结果

系统结构



数据分布方式

- 手动指定
  - 按照bucket分配(bucket数容易调整)
  - Proxy 自动路由, 根据存储节点的数据信息
- 为什么不用Consistant Hash?
  - 数据迁移成本比较大
  - 数据所在位置不清晰，不方便迁移
  - 不方便扩容
  - 等节点数更多时再采用

数据分布

All records in buckets																	
server	buckets																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	all
chubb1:7900					28135608	28159580								28165417	28164878	28133234	140758717
chubb2:7900	28133235	28135997	28156326		28135608	28159580											140720746
chubb3:7900					28135608	28159580								28165417	28164878	28133234	140758717
cirdan:7900							28173589	28174891	28161902	28141653	28147586	28138986					168938607
elrond:7900							28173589	28174891					28151291	28165417	28164878	28133234	168963300
elros:7900																	0
leaflock:7900	28133235	28135997	28156326	28144805					28161902	28141653			28151291				197025209
skinbark:7900	28133235	28135997	28156326	28144805							28147586	28138986	28151291				197008226
treebeard:7900				28144805			28173589	28174891	28161902	28141653	28147586	28138986					197083412

同步

- Hash Tree
  - 16进制
- @
  - @0
    - @00
  - @a
    - @a0
    - @aa
  - @f
    - @f0
    - @f1

memcat --servers=localhost:7900 @

- 0/ 0 0
- 1/ 0 0
- 2/ 0 0
- 3/ 0 0
- 4/ 30407 10897413
- 5/ 985 10904490
- 6/ 15288 10916003
- 7/ 30051 10911943
- 8/ 3984 1
- 9/ 0 0
- a/ 23376 1
- b/ 0 0
- c/ 50546 10893019
- d/ 15953 10899763
- e/ 36641 10903396
- f/ 34832 10886637

冲突解决

- 版本号优先
  - 用高版本覆盖低版本
- 修改时间
  - 用新数据覆盖旧数据

数据迁移及故障恢复

- 直接拷贝数据及Hint文件
- 不需要关闭正常数据节点

协议

- memcached 兼容协议
- set
  - 过期时间作为版本号
- get @xxx
  - 返回 Hash Tree 状态
- get ?xxxx
  - 返回对应key的meta信息
- flush_all
  - Merge, 压缩数据文件大小

协议实现

- 使用memcached的代码
- 异步网络IO
- 多worker线程, leader/follower 模式

数据存储实现

- Bitcask
  - 日志结构 + 全内存索引
  - 简单, 可靠, 高性能
- 多目录
  - 便于迁移和恢复
  - 提高并发性
- 支持数据压缩
  - 自动选择性压缩

```
testdb
|-- 0
|   |-- 000.data
|   |-- 000.hint.qlz
|   |-- 001.data
|   `-- 001.hint.qlz
|-- 1
|   |-- 000.data
|   |-- 000.hint.qlz
|   |-- 001.data
|   `-- 001.hint.qlz
```

HashTree 实现

- 需求
  - 读取, 保护在数据文件中的偏移
  - 修改时的版本管理, 版本号
  - 节点间的同步, 内容的hash
  - 能快速建立, 降低启动时间
- 空间效率
  - 索引全部在内存, 决定了单机容量
  - 平均 20 byte / record (包括key在内)
- 时间效率
  - 每秒插入近百万条

HashTree实现

- 类似于Heap实现
- Item
  - [verion, hash, pos, length, name[…]]
- Node
  - 基于数组，16进制

- Leaf
  - Item 连续存储, 最多 128 个
  - [size, count, used, Item[0], Item[1], …]

线程模型

- N个 worker 线程
- Leader / follower 模型
  - 得到事件后立即处理, 无线程切换
  - 释放leader锁, IO操作时不阻塞其它事件
- 写缓存，后台定时/定量写入到磁盘
- 读的时候才打开数据文件
  - 减少fd占用, 读并发好

案例1：图片和音频文件

- 数据量：
  - 每个 1k到20M
  - 460G x 16 x 3 = 22T
  - 16M x 16 x 3 = 768 M
  - 17 个节点, 约 50 块SATA硬盘
- 访问量：
  - 150 qps左右
  - 有CDN作为缓存
- 性能
  - Med/Avg/90%/99%：18/35/86/266 ms

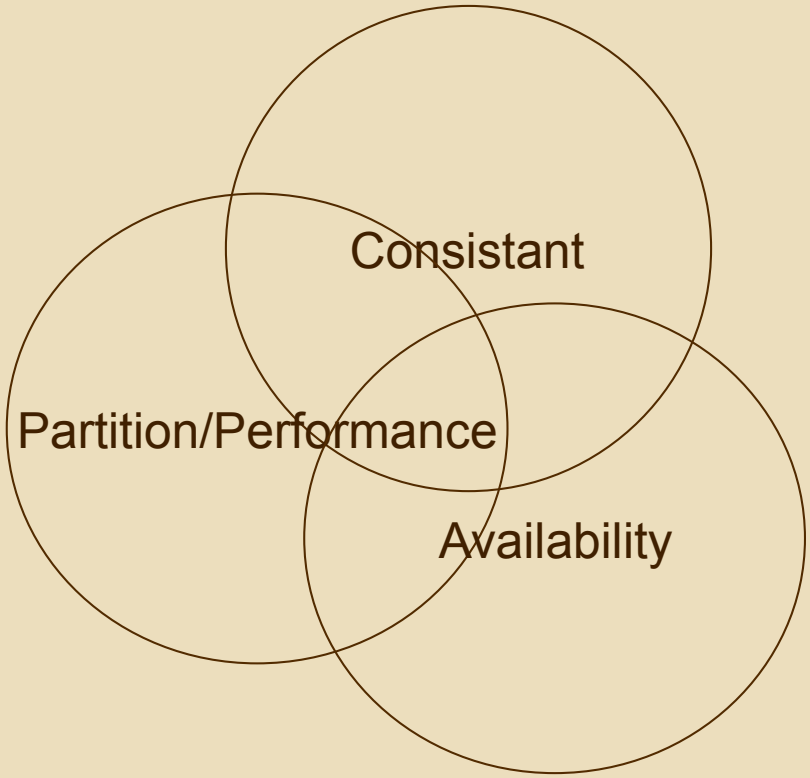
案例2：文本数据

- 数据量：
  - 每个 10b 到 100k
  - 70G x 16 x 3 = 3.3T
  - 28M x 16 x 3 = 1.3B
  - 8个节点, 9 块SATA硬盘
- 访问量：
  - 180 qps左右
  - 有 memcached 作为缓存
- 性能
  - Med/Avg/90%/99%: 3/10/21/65 ms

为什么做BeansDB?

- 其它Key-Value选择
  - memcachedb, tokyotrant, 主从复制
  - 不便于Fail-Over
- 其它类Dynamo选择
  - Dynomite: erlang完整实现，内存效率可能有问题
  - Voldemort: Java部分实现
- 最初的动机:
  - 存储大量图片，MogileFS比较慢

CAP理论



大型Web 2.0应用中的问题

- 数据量随着用户量激增 (P++)
- 需要 7 x 24h 在线 (A+)
- 用户对数据一致性要求不苛刻(C)
- 大量一次写数据（图片，文本）（C-）

BeansDB相对Dynamo的简化

- 简化服务器端，在客户端实现Routing
- 简化版本管理，用单版本号+时间戳
- 简化数据拆分，手动管理数据分布
- 简化协议，使用mc兼容协议
- 简化server端实现，bitcask + memcached
- 简化同步，外部定时任务实现