通知系统的存储设计

街旁 2013.9

通知是什么

用户接收到的由另一用户或系统发送的消息提醒

通常和某一行为关联在一起

如:

评论了某人的状态

加某人为好友

收藏了某人的攻略

获得了徽章

数据特点

数据总量大，规模仅次于签到

单条数据很小

热点分布不均匀

查询维度很多

单个用户通知有上限

可以滞后更新

查询维度

查询一个用户某些类型的通知列表 to_user, type

查询一个用户某些类型的未读通知数 to_user, type, unread

删除一个用户对签到标记赞的通知 to_user, from_user, type, post

标记一个用户签到被回复的通知已读 to_user, from_user, type, unread, post

……

对象属性

One Notification

必须字段 **required**

主键 ID，类型 type，接收者 to_user，时间 create_on，已读 unread

选择字段 **optional**

发送者 from_user，状态 post_id

发送者 from_user，地点册 venuelist_id

徽章 badge_id

……

存储结构

通知数据

notif:{id} -> Hash

Example

```
redis> HGETALL notif:117527377
 1) "c"
 2) "1378106545"
 3) "fu"
 4) "836401721"
 5) "tu"
 6) "920809610"
 7) "ur"
 8) "0"
 9) "p"
10) "106903224"
11) "t"
12) "1"
```

用户通知列表

user:{user.id}:notif -> List

Example

```
redis> LRANGE user:920809610:notif -15 -1
 1) "117156684"
 2) "117298016"
 3) "117301030"
 4) "117303982"
 5) "117307963"
 6) "117309837"
 7) "117309855"
 8) "117321952"
 9) "117330242"
10) "117335231"
11) "117454717"
12) "117463356"
13) "117479951"
14) "117480342"
15) "117527377"
```

通知 ID 生成

notif:last.id -> String

Example

```
redis> INCR notif:last.id
(integer) 117527378
```

查询算法

Python Interface

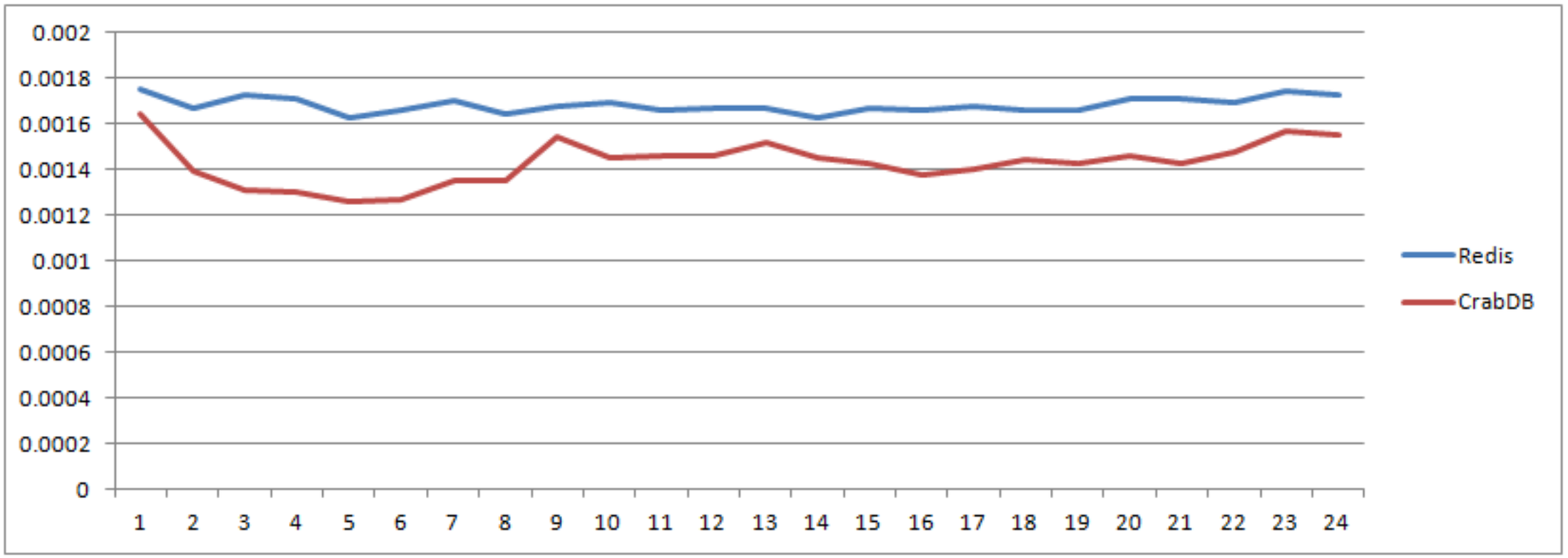
```
rds_notif.eval(LUA_SCRIPT, 1, key, types, cond, limit, max_id, since_id)
```

Lua Script

```
if (max_id <= 0 or notif_id < max_id) and (notif_id > since_id) then
    local notif = bulk_to_table(redis.call('HGETALL', key))
    local type_selected = false
    for i = 1, #types do
        if notif['t'] == tostring(types[i]) then type_selected = true end
    end
    if type_selected or #types == 0 then
        local cond_selected = true
        for k, v in pairs(cond) do
            if notif[k] ~= tostring(v) then
                cond_selected = false
                break
            end
        end
        if cond_selected then table.insert(result, notif_ids[i]) end
    end
    if limit > 0 and #result >= limit then
        return result
    end
end
```

性能对比 I

取一天中二十四个时段耗时的中位数

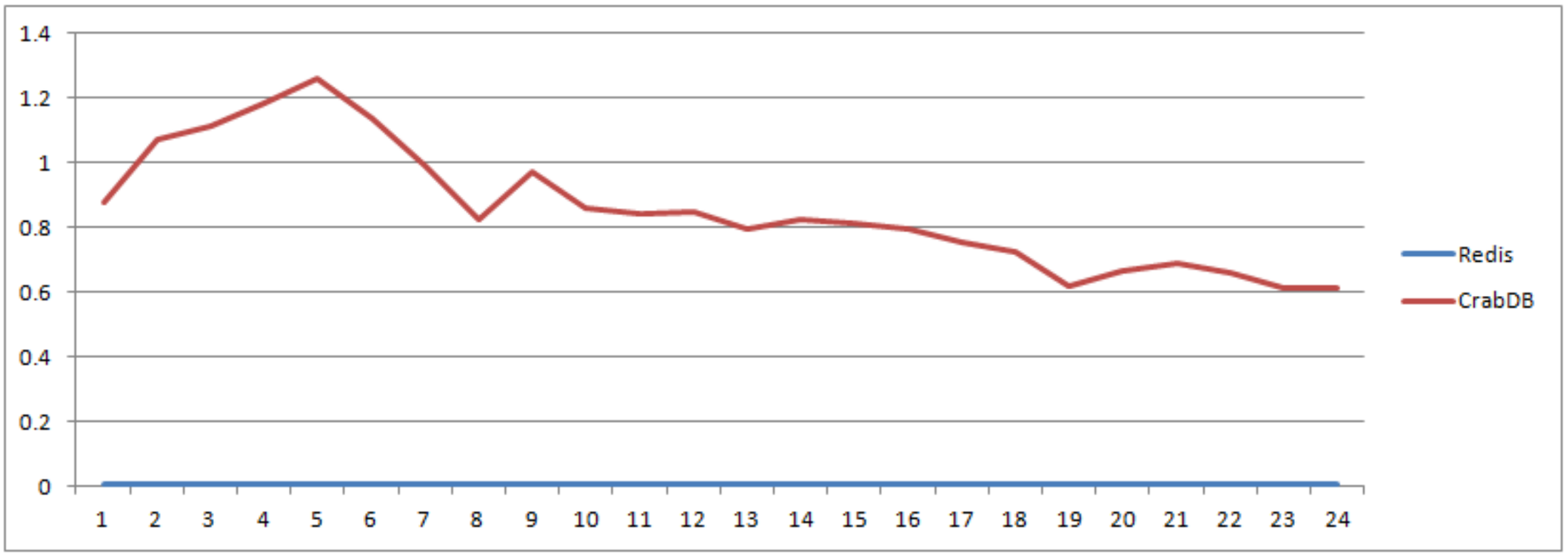


使用 Redis 比 使用 CrabDB 耗时时间上升了一些

Lua 与 C 执行效率上的差距

性能对比 II

取一天中二十四个时段耗时最大的10%



使用 Redis 比 CrabDB 耗时有两个数量级的下降

推断，CrabDB 在进行磁盘 IO 时带来数量级性能的下降

Redis 在内存管理上更合理，减少磁盘读写次数，带来更稳定的性能

存储引擎 I

MongoDB

方便的 Replication

直观的面向对象查询接口

糟糕的操作系统级内存管理 mmap

索引膨胀

CrabDB

压缩的数据集，节省空间将数据尽量装入内存

高效率的 C 查询表达式

数据存储的可靠性有待验证

发生磁盘读写带来的性能不稳定性

当我们没有 MongoDB 和 CrabDB 时，我们如何解决问题

存储引擎 II

Redis

良好的稳定性和扩展性

丰富的数据存储结构

完善的运维和监控工具

活跃的社区支持

MySQL 可以解决我们的问题吗?

Scale Out

主从架构，读写分离

按功能分库

水平分表

怎样设计一个更好的大规模数据存储服务

且听下回分解：分布式数据库的演变过程