

如何快速获得高并发编程经验？PCC性能挑战赛作品简介及源代码

Original 2017-05-12 高可用小助手 高可用架构

PCC 是 Performance Challenge Championship（性能挑战杯）的缩写，是高可用架构后花园会员在线上组织的一个活动，由于反响热烈，考虑到线下进行可以更好的加深对高并发编程的理解，于是高可用架构在 3 月组织了本次 PCC 活动。



对于工程师来说，参加 PCC 编程挑战赛的部分意义：

- 体验完成一个技术小目标。高性能系统如何实现应当是每个工程师需要走的路。
- 学习优秀的架构方法，隔壁老王用的设计思想，可能你坐在办公室永远也无法想到。
- 有经验评委的点评，了解真实环境的高并发系统的追求目标。

类似主题、有同样级别参赛队员及评委参加的编程活动，可能仅此一次。

比赛方法说明

实现类似 facebook 中的 like 功能，需要：

- 可以对一个对象（一条feed、文章、或者url）进行 like 操作，禁止 like 两次，第二次 like 返回错误码
- 有 isLike 接口，返回参数指定的对象有没有被当前用户 like 过
- 需要看到一个对象的 like 计数
- 可以看到一个对象的 like 用户列表（类似 QQ 空间）；
- 上述列表加分项：Like优先显示我的好友列表(social list)。

数据量：每天新增的 like 对象数为 1 千万，每秒 like 计数器查询量为 30 万次 / 秒。

比赛盛况

比赛题目在比赛前就发给了选手，实际上不少选手头一天晚上已经可以将功能跑通，第二天的时间主要用于优化。

第二天早上，选手陆续来到比赛现场，和一般的活动热闹现场的区别是，PCC 比赛的现场异常的安静，因为选手都在潜心优化及调试代码，1天的时间实际是非常短的，必须抓紧每一分一秒。中午吃饭的时间大家也都匆匆交流了十几分钟，又回到电脑前继续开发。

大赛组委提供的模拟数据

<https://github.com/archnotes/PCC/tree/master/data>

到傍晚，经过1天的角逐，代码都写得差不多了，就等压测了.....



PCC 评委领宇鹏（一乐）

组织方突然放出 200G 的测试数据，虽然是内网，但是几十人在云平台内部传输这么大数据也是一场风暴，考虑到已经没有选手使用纯内存方案，组织方简化了一下条件，测试数据被压缩到 40G，数据需要选手自行导入自己的系统中，不过导入速度依然慢得超出了大家自己的期望。各种新的问题开始冒出来了，真正体现高性能优化效果的时刻。

尽管大家还沉浸在优化的过程中不能自拔，到了晚上 8 点，评委宣布截止比赛，并根据比赛的规则宣布了入围名单。

优秀作品展示



图：入围奖选手及评委

参考实现：方圆

项目地址：<https://github.com/archnotes/PCC>

入围奖作品介绍

入围奖：覃冠日

我采用的是 OpenResty + Pika 的架构，OpenResty 能支持高并发的请求处理，使用 Lua 脚本完成业务逻辑处理，利用 OpenResty 的 sharedict 完成数据的缓存。存储层使用 Pika，利用有序集合、hash 等数据结构存储用户数据。

项目地址：https://github.com/qinguanri/demo_lua

入围奖：夏海峰

从架构的简单，可扩展，低耦合的角度考虑，我选择了微服务架构；同时从一个完整的业务系统的角度考虑，将整个系统分为了 article, user, action 三个微服务系统。三个微服务主要功能有：

- article - 处理 article 的存储，查询等功能
- user - 用户账户信息的存储，用户登录等功能
- action - 存储用户的操作行为，包括点赞，添加好友等功能（各种行为其实还可以再细分为不同的微服务）

微服务之间使用 gRPC 来获取信息；对于数据的写入，通过 NSQ 来传递到不同的微服务系统，实现数据的异步写入，达到数据的最终一致性。

数据存储上，采用了 SQL + NoSQL 缓存的模式。由于数据量的限制，使用内存缓存是不太可能的；所以这里选择了 SSDB(Leveldb + Redis 协议)作为数据缓存。为了减少不必要的存储空间占用，使用了 Protobuf 作为数据交换格式。缓存主要有两类东西：实体 + 列表。列表主要解决了“点赞”用户列表的查看，翻页问题。

我的项目中，对 优先显示点赞用户中我的好友 这个需求理解错误；这个需求，将我的好友和点赞用户做个交集是一个比较简单清晰的方案，这在微服务架构中很好扩展。

主要的技术栈：Golang, Postgresql, SSDB, NSQ, Protobuf, gRPC, Microservices, RESTful API

项目地址：<https://github.com/chideat/pcc>

入围奖：陈刚



缓存设计

- feedLike 计数器 : key : like_count:feed_id ; value: 存储 String , like, unlike 操作时, 分别 incr,decr。
- likeList 列表 : key: like_list:feed_id;value:lists , like 时 , rpush 插入用户 id , unlike 时 , srem 删除对应的用户 id。
- friends 列表 : key: friends:uid ; value:lists , 存储所有好友的 uid 列表。
- 点赞的好友列表 : key:like_friends:feed_id;value:lists,likeList 列表与 friends 列表求交集, 得到好友 list。
- 点赞的其他人列表 : key:like_others:feed_id;value:lists , likeList 列表与friends 列表求差集, 得到非好友 list。

注：由于数据量大，所以要采用 redis cluster 来存储。同时，由于要求交集，是否要求两个 list 必须在同一 shard 上，这个在 redis cluster 尚未验证。

性能优化

依据场景可以做一些裁剪，真实情况下，赞列表很少翻到后面，可以在缓存中只存储前 100 条数据，减少存储量。当翻页到 100 条后，再通过数据库，获取后面的数据。

如果好友关系变化不影响历史的话，每个 feed 可以设计两个赞列表，一个好友的，一个非好友的。在数据写入时，就计算好，分别写入，提升读性能。（如果好友关系实时变化，需要重新计算数据）

实现

关系数据库：MySQL 5.7

计数、查询缓存：Redis

接口实现：spring-boot

项目地址：<https://github.com/iqinghe/pcc-like>

二等奖

最后，经过评委对架构打分集体商议评比后，产生了 PCC 的二等奖。



图：二等奖选手及评委

二等奖 黄东旭



项目地址：<https://github.com/c4pt0r/pcc>

二等奖 唐福林



作为一个 local cache 的坚定拥护者，在第一眼看到这次比赛题目的时候，就已经决定了要用 local cache 来做。

唯一的问题是，Java 技术圈里，local cache 不少，但真正适合大量数据的却不多。曾经在线上环境用过 ehcache，也用过 hazelcast，非线上环境尝试过 mapdb。这一次，想试试号称“高频交易”而生的 Chronicle-Map。

选中 Chronicle-Map 是因为：

- Map 接口，使用简单
- off heap，无 gc 压力
- mmap 文件，支持重启不丢失数据

为了解决 value 长度差别过大，导致写入文件性能低下的问题，我在原生的 Chronicle-Map 外面包了一层 ListmapService，用多个不同的 map 来存储不同 value 长度的数据。于是这个方案的重点就变成了如何根据数据的分布选择合适的 map size 的问题了。

用 Springboot 写微服务如行云流水，半天时间连 test case 都写好了。但写到 cursor 翻页的地方，我才反应过来，简单粗暴的的数组并不是一个很适合存储 like 列表的数据结构。果然，在后面的导入数据环节，因为数据结构不够高效，导致导入速度非常缓慢，简单的做了一下并发导入的优化，但效果依然不够理想。

比赛结束后，回过头来想想，这样的比赛对于码农来说确实非常有帮助，既锻炼了写码速度，又开拓了架构眼界。唯一不足的是，很多参赛方案最后都演变成了开源组件选择比赛，选 nginx，选 redis，选 leveldb，选来选去，最终也没有选出一个因为所以来。

项目地址：<https://github.com/tangfl/chestnut>

压测程序说明

本次性能挑战赛使用的压测程序是 Tsung。Tsung 是一个开源的性能测试工具，能用来压测 HTTP, WebDAV, SOAP, PostgreSQL, MySQL, LDAP 和 Jabber/XMPP 等服务。它支持分布式压测，将压力分布在多个测试机，模拟数十万甚至更多的虚拟用户数并发产生压力。



感谢 @left2right 贡献压测程序

一等奖

那有看官要问了，PCC 一等奖花落谁家呢？由于本次参赛时间比较短，测试数据集也比较大，无法在有限时间内完全决出跑分胜负，我们期待上面的选手能够继续优化工程，能够在代码优雅方面具有广泛的借鉴参考价值，并且跑分持续领先的话，PCC 一等奖的大门是一直打开的。

感谢

本次挑战赛活动的高性能云服务平台由青云提供支持。

感谢青云的场地以及在许多在背后默默支持活动的小伙伴们。感谢大赛评委梁宇鹏、刘奇、王渊命的热心支持。



想进一步了解 PCC 代码, 请访问 PCC 项目仓库

<https://github.com/archnotes/PCC>

推荐阅读

- [首届高可用架构PCC性能挑战赛于3月在北京举行](#)
- [获得PCC性能大赛背后的RocksDB引擎:5分钟全面了解其原理](#)

高可用架构
改变互联网的构建方式



长按二维码 关注「高可用架构」公众号