豆瓣的基础架构

洪强宁 软件架构师 2014-04-16

嘉宾介绍

洪强宁，豆瓣首席架构师。豆瓣第一位全职员工。清华毕业后，洪强宁一直做嵌入式系统。在2002年开始接触Python语言，从硬件工程师变为软件工程师，对一种语言在计算机底层如何工作有深入的理解。

架构

豆瓣整个基础架构可以粗略的分为在线和离线两大块。在线的部分和大部分网站类似：前面用LVS做HA，用Nginx做反向代理，形成负载均衡的一 层；应用层主要是做运算，将运算结果返回给前面的用户，DAE平台是这两年建起来的，现在大部分豆瓣的应用基本都跑在DAE上面了；应用后面的基础服务也 跟其他网站差不多，MySQL、memcached、redis、beanstalkd，不一样的是NoSQL的选择——BeansDB，这是我们在几年前开源的KV数据库，也是国内比较早开源的KV数据库。

BeansDB项目可以说是一个简化版的AWS DynamoDB，该项目在2008年启动，2009年开源，第一版使用tokyo cabinet作为存储引擎，2010年使用bitcask存储格式重写了存储引擎，性能更好。BeansDB对key做哈希运算找到节点来实现分布和冗 余， 一个写操作

会写好几个节点，而现在的配置是写三份读一份。BeansDB主要的特点是支持海量KV数据库——相比Redis这种支持几十个G到几百个G的 内存KV数据库，BeansDB可以支持到上百T的数据。另外BeansDB最大的好处就是运维很简单，性能、可用性、扩容都很好，也实现了最终一致性。

BeansDB中间的Proxy是用Go语言写的，也是一个开源的组件。整体来说BeansDB的设计结构比较简单，相比Redis那种有多种value 类型的方式，BeansDB的Value比较简单一些。

在豆瓣内部建立了两个不同的BeansDB集群，一个是doubandb，一个是doubanfs，分别针对不同的场景。doubandb主要存储 小型文本数据，如影评、用户个人介绍、帖子内容等，这样的好处是可以大大降低我们对MySQL的性能依赖，算是给MySQL减负；doubanfs主要存 放图片和音频等中型数据。

DAE可以说是基于很多以前积累的、旧的组件做起来的。我们做的这种对内的PaaS，相比对外的PaaS而言做了很多简化，尤其是安全方面如应用间 隔离、权限管理方面，我们都不用像公有云那样花大量精力去做，所以工作量其实还好。DAE现在在计划开源，当然它现在只支持Python应用。以后我们也 许会让DAE支持Go语言。

上面是在线的部分，对高可用性和低时延有较大要求。离线部分则包括数据挖掘、数据分析等，技术组件分别是海量分布式文件系统MooseFS，这个文 件系统的结构类似HDFS，用C语言编写，其好处在于FUSE模块实现的比较好，用文件系统就可以直接进行操作，而不需要专门的命令，可以支持的数据量也 很大。另外就是自己开发的分布式计算平台DPark。

DPark顾名思义是Spark的Python实现，不过现在已经跟Spark越来越不一样了。和Hadoop 相比，Spark可以使用内存做为缓存加速分布式计算，DPark继承了这个优点，这对于大规模数据的迭代计算非常有用。在豆瓣的应用场景下，因为我们的 离线计算很多是推荐算法计算，这种计算涉及大量的迭代算法，如果每次计算的结果都入磁盘再在下一轮计算加载，那性能是很差的，所以DPark能够大幅提升 性能。另外，因为DPark的编写使用了函数式语言的特点，所以可以写的非常简洁：

到目前（2014年3月），DPark的集群规模和处理数据量已经比去年多了一倍左右，一天要处理60~100TB左右的数据。

团队

当前，我所负责的豆瓣平台部一共包括四个部分：核心系统，这块也是由我直接带领的，共6名工程师；DAE，现在是彭宇负责，共4名工程师；DBA两人；SA两人。

平台部负责的项目大多是跟业务无关的东西，贴近应用层的主要在产品线团队做，这个分工跟豆瓣工程团队的发展历史有关。早期豆瓣工程师还不多的时候， 就已经分为两种倾向，一种是偏业务的，就是去做用户能看得见的东西；另一种是支持性的，运行在业务层下面、不被用户所感知的东西。下面这一层就衍变成了平 台部门。

在豆瓣，不管是做产品还是做平台的工程师，技术实力都比较强，一个项目应该从哪个部门发起，并不是看这个任务的难度，而是看它是公共的还是业务特有 的。有些项目即使未来可能会成为公共的，但一开始只是一个产品线需要，那么它也会从产品线发起。比如豆瓣的短信服务，最开始是产品线有需求，所以这些服务 都是由他们发起完成的，平台这边主要负责提供建设服务的架构，比如DoubanService，告诉他们一个服务怎样去写、怎样去部署、怎样去对用户开 放。短信服务后来成为很多产品线都在使用的服务，同时这个系统本身也越来越成熟，那么它逐渐就被转移到SA团队来进行维护。

核心系统组做过的项目，包括刚才提到的DPark、BeansDB，还有MooseFS这些二次开发的，还有搜索服务、信息推送的长连接服务等，大大小小差不多有十几个。有些项目处于维护状态，所以需要的人不是那么多。

跟豆瓣其他工程团队一样，平台部也强制大家做code review。这对于核心系统来说很重要的一点在于，code review是一个知识共享的过程：我们人少项目多，所以很多项目都是一个人做主力，很容易就变成其他人不知道你这个项目具体是什么情况，而强制code review就可以实现一种公开透明的状态，让大家都了解每个项目在做什么。

在平台部，因为你做的所有东西都会影响到全公司，测试显然很重要，我们还做了另一件事来进行质量保证，那就是一个项目由谁来主导上线，谁就要负责这 个项目的故障响应——所有运维、调整系统等SA的工作，你这个第一负责人都要参与。你做的东西的好坏会影响到自己晚上能不能睡好觉，所以大家就会比较谨 慎。灰度上线也是我们这边的通用做法。

平台部还有一点跟产品线不一样的是，平台部没有产品经理，所以你的工作方向更多是自己去找的，每个人自己发现问题的能力更重要。我们每个月都会问大家，你这个月想要解决什么问题？如果方向大家一致认可，那就去做。

最后，对于新技术的引入上，豆瓣整体是比较偏激进的，我们鼓励大家去看看新的技术。当然我们也不会看到新的就上，这里面有一些限制：一个是比较重要 的服务如果要上新的技术，一定要有成功案例，且成功案例有跟我们量级差不多的规模，这样可以降低风险；另一个是对于引入的新技术一定要吃透——大部分引入 的技术肯定是要做二次开发的，所以拿进来的技术你必须保证能完全理解它的代码结构，出了问题能修，能去掉自己无法掌控的东西。这也是为什么豆瓣不太可能在 重要的地方引入Java的原因，除非别无选择，我们一般都是Python、C和Go。

----------华丽分割线----------

欢迎【订阅】微信公众号：ArchitectClub，为您的软件架构师之路添砖加瓦。

阅读原文