构建优质的推荐系统服务

原创 gongyouliu 大数据与人工智能 2019-04-17

作者丨gongyouliu

这是作者的第7篇文章，约1万字，阅读需60分钟左右

前言

作者在之前的文章《**推荐系统的工程实现**》第三节中，对推荐系统业务流和各个模块做了简单介绍，相信大家已有初步的了解。

文章中简单提到了**推荐Web服务模块**，这一模块也是直接与用户交互的部分，在整个推荐系统业务流中具有举足轻重的地位，因为Web服务模块的好坏直接影响用户体验。

本篇文章，作者会详细介绍**怎么构建优质的推荐交互模块，如何打造优质的推荐服务，更好地服务用户。**

-以下为正文-

任何一个优质的软件服务都必须考虑**高性能**、**高可用**(HighAvailability)、**可伸缩**、**可拓展**、**安全性**等5大核心要素，推荐系统也不例外。

所以，我们会围绕这5个点来说明，怎么构建高效的推荐服务。

本文会从推荐服务背景介绍、什么是优质的推荐服务、构建优质服务面临的挑战、一般指导原则、具体策略等5个部分来展开讲解。

希望读者读完本文后，对什么是优质的推荐服务能有初步了解。同时，我也试图为读者提供相应的方法和策略，期望本文可以作为大家的参考指南。

推荐服务背景介绍

推荐产品是通过推荐服务来为用户提供个性化推荐能力的，我们可以从**广义**和**狭义**两个角度来理解推荐服务。

从广义上讲，推荐服务是指整个推荐业务，包括数据收集、数据ETL、推荐模型构建、推荐推断、推荐web服务、推荐前端展示与交互等(见下面图1)。

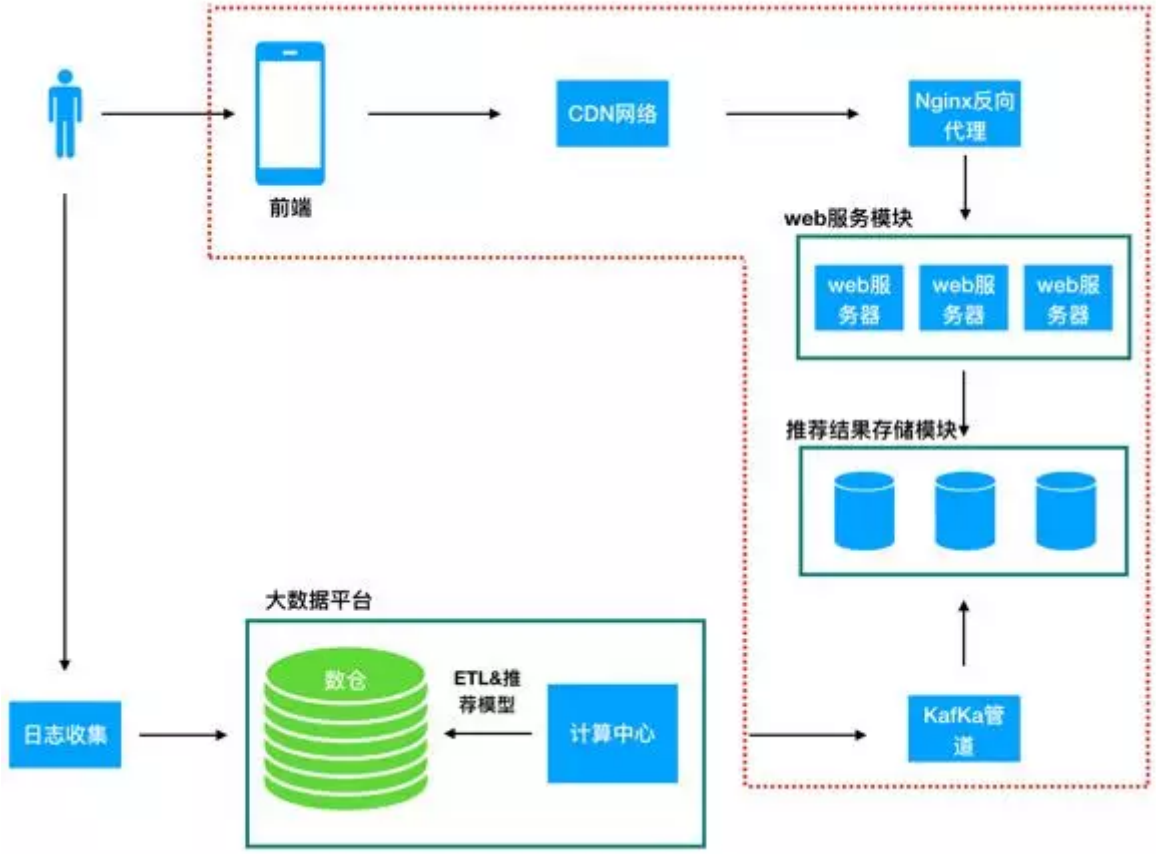

图1：推荐系统的业务流

图1中，大数据平台包含的数仓、计算平台等模块很多公司(特别是初创公司和中小型公司)都是基于开源的大数据平台(Hadoop、Spark、Hive等)来构建的，这些系统本身(或者通过增加一些组件)的设计是具备高可用、可拓展、可伸缩、安全等特性的。

同时，我们的数据ETL、推荐模型训练、推荐模型推断是基于数仓、计算平台基础之上构建的，也需要具备上面这些特征，这部分我们在这里不做介绍，在未来分享推荐算法时会单独讲解。

从狭义上讲，推荐服务是指用户通过终端(手机、Pad、电视等)与推荐系统的web模块的交互，即图1中红色虚线框中的部分(其实Kafka管道不属于直接参与Web服务的组件，但是我们是通过这个模块来跟更底层的数据处理算法组件解耦合，通过它来对接计算出的推荐结果，所以也包括进来了)。

本文我们将主要精力放到关注推荐系统Web服务上，即狭义上的推荐服务。

用户与终端交互的过程见下面图2，用户通过终端请求推荐服务，推荐服务模块通过返回相关的推荐结果给到终端，终端将推荐结果展示给用户。用户与终端的交互虽属于视觉及交互设计范畴，

与推荐工程师的工作无直接关系，但是会直接影响到用户的体验，也在我们讨论之列。绿色虚线框中是真正的推荐系统Web服务过程。

图2：用户与推荐系统交互的数据流向

后文所有关于构建优质服务策略的主题，都围绕这里所指的狭义的推荐服务来展开。

简单介绍完什么是我们本文要讨论的推荐服务，那么什么是优质的推荐服务呢？我们又可以从哪些维度来衡量推荐服务是否优质呢？

什么是优质的推荐服务

推荐服务作为一类软件服务，遵循通用的软件设计原则。

在复杂的软件设计中我们需要从**高性能、高可用、可伸缩、可拓展、安全性**等5个维度来衡量软件架构的质量，对于推荐系统也一样，推荐系统也属于一类非常偏业务的较复杂的软件系统，我们也会从这5个方面来说明什么是优质的推荐服务。

01 高性能

所谓高性能，是指推荐服务可以在较短的时间内给用户返回相关推荐结果，并且数据是准确可靠的，同时用户会感觉整个交互过程很流畅，不会感到非常慢或者卡顿。

一般用响应时间(用户触发推荐页面到返回推荐结果的时间)来衡量高性能，通常服务需要在200ms之内返回结果，否则用户肉眼就可以直观感受到慢了，好的系统可以做到50ms之内返回结果。这个时间当然是越短越好，相应技术实现成本和难度都会更大。

当然，网络会存在各种偶发情况，即使推荐服务性能很好，我们也没法保证每个用户请求都可以在这个时间内响应，所以一般可以采用百分之多少的请求可以在多少毫秒内返回(比如99%的请求可以在75毫秒内返回)来衡量高性能。

02 高可用

所谓高可用，从字面理解就是用户可以一直使用而不出现问题。

由于软件服务是基于现代芯片及硬件基础上构建的，硬件会产生故障宕机，软件也会由于bug或者偶发情况等出现问题，所以一般故障是几乎无法避免的，特别是对于大规模分布式服务，共同服务于同一服务的计算机集群越大，出现故障的可能性也会越大。

这里举个例子：比如飞机是最安全的交通工具，但是一两年基本都有一些飞机相关的事故，主要是全球每天有大量的航班飞行，虽然单次飞行出问题概率非常小，但一两年累计下来至少一次飞行出问题的概率就很大了，学过概率统计的读者应该很好理解。

当这些故障出现时，软件系统将无法响应用户请求，导致提供的服务不及时、不稳定、不可靠，甚至不可用。

计算机行业的高可用一般是通过故障出现后的影响时长、等级及故障恢复的快慢来衡量一个软件系统是否高可用。如果故障不频繁、故障影响面不大、在很短的时间就恢复正常了就是高可用的系统，否则就不是高可用的系统。

很多大型网站，比如淘宝，百度基本达到了99.99%的高可用了，算下来一年大约只有0.88小时不可用。

推荐系统本身就是一项软件服务，对于推荐系统来说，高可用就是推荐服务是否稳定高效的为用户提供服务。

03 可伸缩

我们可以这样来理解伸缩性，将一个模块或者系统类比为一条生产线（如富士康中苹果手机生产线），当有大量的订单需求时，可以通过扩充生产线来应对大规模的业务需求，这就是生产线的伸缩性。

推荐系统需要面对海量用户的推荐请求，同时也要为每个用户存储相关的推荐结果。可伸缩性是指是否可以通过不断增加服务器(在该服务器上部署相关的推荐服务)的手段来应对不断新增的用户及在服务高峰期暴增的请求。这种增加服务器来提供无差别的服务，必须是对用户无感知的，不会影响用户体验。

互联网产品(特别是toC互联网产品)是基于规模效应的一种生意，发展用户是公司最重要的事情，在用户发展阶段，用户是爆发增长的，这时原有的推荐服务是无法满足快速增长的用户需求的，所以要求推荐服务具备伸缩能力是必然的。

由于推荐系统需要存储用户推荐结果，因此相应的存储数据库也需要具备可伸缩的能力，当前很多NoSQL数据库都是具备可伸缩能力的。

04 可拓展

互联网产品是需要快速响应用户需求变化的，所以对产品做调整，或者增加新的产品形态是常有的事情。

可拓展性指的就是推荐服务可以快速响应业务需求变化，非常容易对服务做调整修改，可以非常方便地增加新的推荐业务。

比如，公司在前期没有接入广告，等做商业变现时，需要在信息流推荐中插入广告，这时就需要对信息流推荐产品做调整，整合广告投放能力。

05 安全性

互联网是一个开放的服务体系，我们需要采用技术手段确保网站数据不会轻易被恶意攻击，防止数据被盗。

衡量推荐服务安全性的主要指标是针对各种恶意攻击及窃密手段是否有有效的应对方案，同时是否可以很好的保护用户隐私，特别是今年315曝光了很多数据黑产的利益链，用户数据安全性只会越来越重要，相信不久的将来，就会有更完善的法律保护措施出台。

我们已经介绍完了好的服务设计需要具备的5大要素，这些要素是任何一个互联网服务都必须关注的，更需要我们基于已有的人力资源、经验、投入成本、业务特性等做好平衡。构建优质的推荐服务，也需要关注上面的5点，需要在这5大要素之间做好取舍和平衡。

相对于后台服务，推荐服务是一种较特殊的软件服务，那么对于推荐服务是否可以很容易做到上面5点呢？会面临哪些挑战呢？

设计推荐服务面临的挑战

相对于其他后台系统来说，推荐系统有很多不一样的地方。

对于个性化推荐来说，给每个用户的推荐都是个性化的，所以生成的推荐结果都是不一样的，这些推荐结果需要事先存储下来，方便用户请求时快速反馈给用户，因此需要大规模的数据存储系统来支撑。

特别是随着短视频、新闻APP的火爆，在这些产品中用户消耗单个标的物的时长较短，因此为用户提供近实时的推荐服务，并跟紧用户兴趣的变化，试图占用用户的碎片化时间是这类产品设计中非常关键的要素，也是产品是否具备核心竞争力的先决条件。

具体来说，构建优质的推荐服务，会面临如下挑战：

01 需要存储的数据量大

个性化推荐为每个用户存一份推荐数据，数据量随着用户线性增长。

一般toC互联网产品都是通过规模效应盈利的，所以发展用户是互联网公司最重要的事情之一，做得好的产品用户规模一定会在一定时期内爆发增长，因此数据存储也会急速增长，需要更多的软硬件资源来容纳新增的大量数据。

当用户量大到一定程度时，一台服务器无法装下所有用户的推荐结果，一台服务器也无法为用户提供web接口服务，这时就需要采用分布式技术，需要数据库及web服务系统具备很好的伸缩能力。

02 需要快速响应用户请求

随着新闻、短视频等消费用户碎片化时间的应用层出不穷，越来越多的推荐系统采用近实时的推荐策略，以提升用户体验，同时让用户沉浸其中，增加自己产品的使用时长，方便更好地拉投资或者做变现。

实时给用户提供个性化推荐，这个过程中需要实时学习用户的短期兴趣，并基于用户的短期兴趣实时更新用户的推荐列表，这为整个推荐系统业务设计开发带来极大压力和挑战。

03 接口访问并发量大

个性化推荐由于每个用户推荐结果都不一样，很难利用现代CDN技术来对推荐结果加速(主要是命中率太低)，用户的请求一般都会回源，对后端系统产生较大的访问压力。

总的说来，有可能在极短的时间产生流量风暴。特别是对有些产品，由于产品自身的属性，在特定时段访问流量极大，比如视频类应用，一般是晚上6-9点是访问高峰，这时的流量可能会暴涨50%以上。

04 业务相对复杂

推荐业务为了给用户提供好的体验，需要涉及到很多方面。

比如，需要具备根据一定业务规则做运营的能力。需要为用户过滤掉已经看过的或者曝光过的内容，需要对在推荐结果中下线某个标的物(如视频中某个节目下线，电商中某个商品下线)，需要实时根据用户行为更新用户兴趣推荐。这些较复杂的逻辑，对设计优质服务也是一种挑战。

既然推荐服务的设计有上面这么多挑战，那我们要怎么设计好的推荐服务呢？是否有一些一般的原则可借鉴呢？回答是肯定的。

构建优质服务的一般原则

在讲具体的方法和策略之前，我们先简单介绍一下做到优质服务需要了解的一般思路和原则，这些原则是帮助我们构建优质服务的指导思想。

01 模块化（SOA）

SOA(Service Oriented Architecture)即面向服务的架构，主要目的在于服务重用，通过将服务解耦，提升整个系统的可维护性。

在设计系统时，尽量减少系统的耦合，将功能相对独立的部分抽提出来，通过数据交互协议或者接口与外界交互。这样设计的主要目的是减少系统的复杂度，方便独立对某个模块优化和升级，同时，当系统出现问题时也可以快速定位。

最近几年很火的微服务是对SOA思想的延伸，是一种轻量级的SOA解决方案，将服务拆解为更细粒度的单元，更易于系统维护和拓展。

02 数据存储

互联网行业有所谓空间换时间的说法，意思是通过将需要的结果预先计算好并存储下来，等用户请求时就可以直接返回给用户而不需要再去计算，虽然占用了存储空间，但是大大加快了查询速度。

而数据缓存就是一种空间换时间的做法，先将用户需要的数据(对推荐系统来说，就是返回给用户的最终推荐结果)事先计算好在数据库中存起来。当用户请求时，可以直接给到用户。

涉及到缓存，缓存命中率就必须要关注了，如果一个查询不会经常查到，缓存下来其实是没有太多好处的，因为以后也不会经常用到了。

个性化推荐产品每个用户的推荐结果都不一样，做缓存的价值是没有那么大的。但是对于关联推荐，每个标的物关联的标的物列表在短期(可能是一天)是不变的，这时就可以充分利用缓存的优势了。

03 负载均衡

负载均衡(Load Balance)，就是将请求均匀分担到多个节点上执行，每个节点分担一部分任务，整个系统的处理能力跟节点的数量成线性相关，通过增加节点可以大大提升整个系统的处理能力。推荐接口会大量采用负载均衡技术。

04 异步调用

举个简单的例子，你去银行办业务，拿到号后需要排队，如果你一直看着屏幕等待你的号出现，这就是同步。如果你在等待的同时用手机处理工作邮件，等轮到你的号了银行语音提示你去办理业务就是异步。

从这个简单例子可以看到，异步可以提升系统(这个例子就是你的大脑)的处理效率，而不必在一件事情上浪费时间。

在推荐服务中可以大量采用异步的思路，比如将推荐结果插入数据库时，可以采用异步插入，提升插入的效率，响应接口请求时也可以采用异步处理。

由于异步不需要双向确认，大大提升了效率，但是也可能由于没有确认，导致部分处理请求失败(比如某个用户的推荐结果由于各种未知原因未插入数据库)。

后面会讲到推荐业务是可以容忍一定的错误的(不像涉及钱的会员等业务必须准确无误)，同时推荐业务需要处理大规模的数据(如T+1的个性化推荐，在一两个小时内需要为每个活跃用户更新推荐

结果，如果用户规模很大，这个过程是很耗时的)， 所以采用异步可以大大提升效率。

05 分布式及去中心化

分布式网络存储技术是将数据分散地存储于多台独立的机器上。

分布式网络存储系统采用可扩展的系统架构，利用多台存储服务器分担存储负荷，利用位置服务器定位存储信息，不但解决了传统集中式存储系统中单存储服务器的瓶颈问题，还提高了系统的可靠性、可用性和扩展性，这种组织方式能有效提升信息的传递效率。

通过将系统、数据或者服务分布于多台机器上，首先可以增强整个系统的处理能力，同时也可以降低整个系统的风险。

去中心化是互联网发展过程中形成的一种内容或服务组织形态, 是相对于“中心化”而言的新型网络内容的生产过程。在计算机技术领域，去中心化结构使用分布式核算和存储，不存在中心化的节点，任意节点的权利和义务都是均等的, 系统中的数据块由整个系统中具有维护功能的节点来共同维护，任一节点停止工作都不会影响系统整体的运作。

推荐系统的web服务和数据存储都可以采用分布式和去中心化的思想利用相关开源系统构建，如CouchBase数据库就是分布式去中心化的数据库。

06 分层思想

分层跟模块化思想类似，最大的区别是各个层之间是有直接的依赖关系的，分层一般也是根据逻辑结构、数据流、业务流等来分，即使是同一层内，也是可以做更细粒度模块化的。

分层的目的是让系统逻辑结构更清晰，便于理解、排查问题。推荐系统根据数据流就可以简单分为数据生成层、数据存储层、数据服务层，后面会详细介绍。

讲完了设计优质服务的一般思想，那我们就来详细讲解一下具体有哪些策略可以帮助我们设计优质的推荐服务。

可行策略

我们在第一节中对推荐服务的范围做了简单限定，在第二节对优质服务的5个维度做了简要说明，结合第四节的基本原则，我们在本节来详细说明怎么设计优质的推荐服务，有哪些具体的策略和方法。

设计优质推荐服务的目的是希望更好的服务于用户，提升整个系统的效能，最终提升用户体验。我们还是从**高性能、高可用、可伸缩、可拓展、安全性**5个维度来展开介绍。

高性能

为了能够提供高性能推荐服务，我们可以从如下维度来优化推荐服务模块，以提升推荐服务的响应速度，给用户更好的交互体验。

01 CDN缓存

CDN(Content Delivery Network，即内容分发网络)是一个非常成熟的技术，通过部署在各地的边缘服务器来对内容进行加速。我们也可以利用该技术来加速推荐服务。

对于非个性化推荐(如排行榜、关联推荐等)，每个用户返回结果都一样，所以命中率极高，完全可以采用CDN来加速，以提升推荐接口的性能。

对于首页上的T+1个性化推荐，由于用户进入(是必经路径，可能会经常回退到首页)的概率较大，特别是很多APP，用户一天多次登录，也可以采用CDN做缓存(命中率可能没有非个性化推荐大)。但是对于实时个性化推荐，每次刷新，推荐结果都不一样，基本无法利用CDN的缓存能力。

CDN缓存虽然可以加速，但是利用CDN缓存也需要注意，如果某个请求出错了，刚好被CDN缓存了，会对后来访问的用户产生负面影响(后来的用户会返回这个被CDN缓存了的出错的结果)。我们需要定期清理缓存，或者跟CDN厂商沟通，采用特殊的缓存策略(如返回的接口为空或者不合法时不做缓存)，最大利用CDN的优势，避免不必要的问题。

02 Nginx层或接口层的缓存

除了CDN层的缓存，我们可以在Nginx层及接口web服务层增加缓存，采用多级缓存的策略能够更好的避免请求击穿缓存，从而更快速的为用户提供推荐服务。

03 数据压缩

如果某个推荐产品形态给用户推荐的数据量比较大(比如，我们公司在做个性化重排序时，可能有几百上千个视频，用户是通过分页来请求的，数据量大，见下面图3战争风云这个tab，会根据用户的兴趣做个性化重排，用户通过下滑遥控器按键分页请求数据)，可以对存储于数据库中的推荐结果进行压缩(比如采用protobuf + base64进行编码)，这样数据量就会少很多，减少网络数据传输，提升接口性能。

图3：基于用户兴趣的列表个性化重排序

04 接口做压力测试

我们不光要验证接口的功能是否正确(功能测试)，还需要事先对接口的性能有所了解，知道接口的性能极限，这样才可以知道在高峰期间，所有推荐接口服务器是否能够抗住压力。

了解接口性能的最好方式是通过压力测试。

通过压力测试就可以知道接口在一定并发量下的吞吐率、响应速度、能够承受多大的QPS。特别是个性化推荐接口，访问量非常大，每次接口做升级或者开发新的推荐产品形态时，都需要做打压测试。

我们基于打压测试及在高峰时段用户访问情况，才可以确定到底需要多少台接口服务器可以支撑现有的服务。

05 服务质量评估

推荐接口性能怎么样？是否有延迟，我们需要收集相关的数据来评估接口响应情况，总响应时间分为两个部分(见下面图4)T1和T2，用户的总响应时间T等于这两部分之和(T=T1+T2)。

其中T1是网络传输时间，衡量网络情况，这部分时间基本是我们很难控制的(当然可以通过CDN加速，提升出口带宽来适当缓解)。

T2即是我们推荐接口响应时长，这部分时间包括从推荐库中获取用户的推荐结果，并将结果组装成前端展示需要的形式(拿视频推荐来说，我们需要组装出节目标题、演职员、详情、评分、海报等前端展示时必要的信息)。

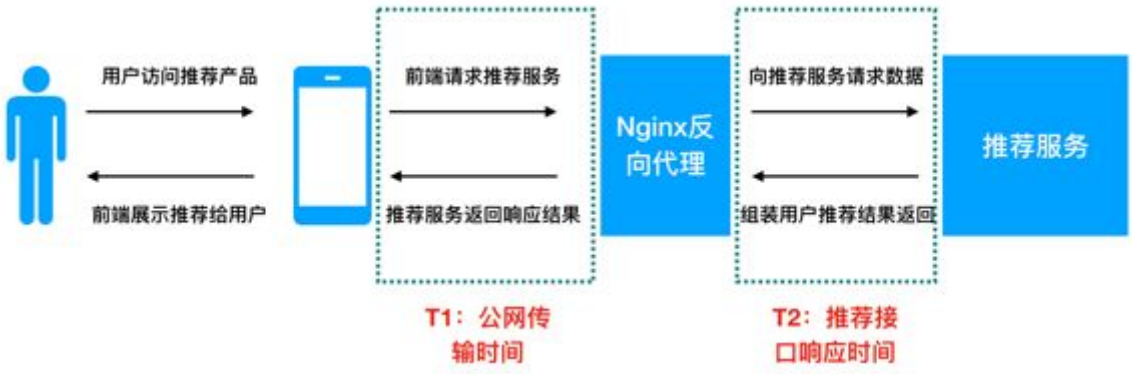

图4：推荐服务响应用户请求链路及时间花费

对于T2，我们可以在Nginx侧记录每次请求的响应时间，并将相关日志收集到数据中心做分析，这样就知道各个推荐业务接口响应情况。

下面图5是我们自己的推荐业务相关接口性能统计情况(为了隐私，隐藏了具体业务名称、QPS及请求次数)。

从下图可以看到很多接口99%的调用响应时长低于50ms，性能是很不错的，但有些性能不是很好，如第四行的，只有81%的请求控制在200ms之内，这些业务都是非常老的版本的业务了，基本不再维护了。

从这张图中，我们可以非常清楚地看到各个业务接口的性能情况，这样我们可以针对业务的重要性和当前性能情况做接口优化。

biz	统计日期	QPS	请求次数	平均响应时长	50毫秒占比	100毫秒占比	200毫秒占比
	20190408			0.00	100.00%	100.00%	100.00%
	20190408			0.01	100.00%	100.00%	100.00%
	20190408			0.01	99.37%	99.72%	99.88%
	20190408			0.13	27.10%	71.65%	81.62%
	20190408			0.01	99.99%	99.99%	99.99%
	20190408			0.03	99.80%	99.80%	99.80%
	20190408			0.01	100.00%	100.00%	100.00%
	20190408			0.01	99.96%	99.98%	99.98%
	20190408			0.01	99.90%	99.95%	99.97%
	20190408			0.01	99.88%	99.95%	99.97%
	20190408			0.00	99.99%	99.99%	99.99%
	20190408			0.01	99.99%	99.99%	99.99%
	20190408			0.01	99.99%	99.99%	99.99%
	20190408			0.01	99.99%	99.99%	99.99%
	20190408			0.01	99.55%	99.78%	99.89%
	20190408			0.14	73.84%	81.27%	89.47%
	20190408			0.00	100.00%	100.00%	100.00%
	20190408			0.00	99.95%	99.95%	99.95%

图5：推荐接口性能统计

对于总时长T，我们也可以在前端通过日志埋点记录下来，同样通过数据分析可以知道一个推荐业务平均耗时多少，总时间减去T2，就是T1的平均耗时，即网络传输时间。

通过对服务质量评估，就可以有针对性的对上述的T1，T2做优化，从而提升接口性能。

06 采用高性能的web服务器

采用高性能的web服务器可以极大提升推荐服务的性能，推荐服务业务逻辑相对简单，可以采用轻量级的web服务器，比如Vert.x(基于java语言的高性能web服务器)、Spray(基于Scala语言的高性能web服务器)、gin(基于Go语言的高性能web服务器)、cowboy(基于Erlang语言的高性能

web服务器)等，这样不仅可以满足开发推荐接口的需求，开发速度快，并且性能也很好。传统的web服务器如Tomcat等太重了，不太适合推荐api接口的开发。

07 采用基于内存的NoSQL数据库

一般来说内存的访问速度比磁盘快好几个数量级，采用基于内存的数据库来存储推荐结果会提升整个接口获取推荐结果的速度，现在有很多开源的这类数据库可供我们选择，比如Redis、CouchBase等。

即使不用基于内存的数据库，也要将数据存放到SSD中，获取速度也会快很多。

高可用

构建高可用系统是一个比较有挑战的事情，具体可以从如下方面来考虑：

01 接口层保护

即使有很多的防护策略，**我们也不能保证推荐接口永远也不出错。**

为了应对这种在极端情况下可能存在的问题，给用户更好的体验，我们可以在前端(即APP侧)做一层接口保护。

具体做法可以是提供一组默认推荐接口，前端在启动时加载该接口，将数据存储在终端，当推荐服务无响应或者响应超时时，可以用默认推荐结果顶替。默认推荐虽然推荐的标的物没有原来的精准，但是不至于“开天窗”，对用户体验也算是一个不错的补救措施。

02 多可用区（多活）

对于创业中期或者成熟的公司，最好需要在多个可用区(同城多活，异地多活)部署推荐服务，避免由于自然灾害(如工程建造挖断光缆、爆炸、水灾、火灾、地震等)等导致服务无法使用。

构建多可用区需要投入非常多的资源，成本较大，对于初创公司建议不要考虑采用这种方式。

03 服务监控与自动拉起

服务监控的目的是保证在服务出现异常的时候第一时间通知运维或者相关负责人，在问题还没有引起灾难时尽快扩容服务器，或者有重大问题时，相关人员可以第一时间知道，快速解决问题。

有了自动监控，当服务出问题或者挂掉后，可以通过监控脚本自动将服务拉起。一般来说，重启可以解决80%的故障问题。

04 灰度发布

灰度发布是互联网公司常用的发布策略，目的是通过先发布少量的用户，看新功能点是否异常，如果有异常及时修复，不至于对所有用户产生不好的影响。

对于推荐服务，我们也建议采用灰度发布的方式，减少由于未发现的未知问题对用户产生的伤害。

05 超时、限流、降级与熔断

当推荐接口服务在一定时间(比如2s)无返回时，可以告知用户访问超时，避免一直等待导致的资源紧缺。

在极端情况下，当接口并发请求太大时(比如今年的春晚百度红包)，可以对访问请求做限制，让部分请求立即执行，其他请求在队列中等待。同时可以对同一IP的多次请求(可能是正常请求，也可能是恶意攻击)做限制，减缓对接口的冲击。还可以限制并发数、网络连接数、网络流量、CPU负载等各种限制措施来对访问进行控制。

熔断可以类比为电表的保险丝，当电流过大时(家里太多电器同时用或者短路)保险丝熔断，停止供电，避免出现意外事故。当请求推荐的服务有大量超时，这时新来的请求无法获得响应，只会无谓的消耗系统资源，这时整个服务可能出现了异常，熔断是较好的策略。

所谓降级，就是当服务不可用(比如熔断后)时，采用效果更差的服务替代，虽然效果没那么好，但是至少比什么都没有强。上面提到的接口层保护就是一种降级策略。

采用超时、限流、降级、甚至是熔断策略，主要是从系统高可用性角度考虑而采取的策略，目的是为了防止系统整体缓慢甚至崩溃。

可伸缩

构建可伸缩的推荐服务，对于应对大规模的用户请求非常必要，我们可以从如下方面来增强系统的可伸缩性。

01 利用NoSQL数据库作为数据存储

由于推荐系统产生的数据量线性依赖于活跃用户量，而互联网产品DAU一般会很大(百万级、千万级、甚至亿级)，所以需要存储大量的用户推荐数据，并且这些数据是会频繁更新的(对于T+1推荐每天更新一次，对于近实时推荐，可能会秒级更新)，所以采用一般的关系型数据库是很不合适的。推荐的结果一般是为一个用户推荐一个标的物的列表，用关系型数据库也不是特别合适，推荐的数据结构一般可以采用list，json等格式存储。

基于上面的说明，这非常适合用现在的NoSQL数据库做推荐结果存储，现在很多NoSQL数据支持Json等复杂的数据格式，并且具备横向扩容的能力。如常用的Redis，就支持String，Hash，List，Set，Sorted_Set等多种数据格式。

在我们公司的业务中，我们主要采用了CouchBase和Redis两种NoSQL数据库，CouchBase是一个文档型分布式数据库，热数据会放到内存中，冷数据会放到磁盘中，并且在水平拓展、监控、稳定性等方面做的非常好。我们将个性化推荐存储在CouchBase中，非个性化推荐(如排行榜、关联推荐等)存储在Redis中。据我所知，在爱奇艺的推荐业务中也大量采用CouchBase。

02 接口web服务可横向拓展

现在一般互联网公司会利用Nginx的高性能特性做反向代理，通过Nginx代理推荐的web服务。

接口web服务最好做到无状态，这样就方便做横向扩展。在我们公司实践中，我们用Go语言的Beego框架和Gin框架来开发推荐接口，开发效率高，稳定，并且性能相当不错，目前Go的生态圈非常完善，是一个不错的选择。

03 自动伸缩

推荐服务的可伸缩性要求我们可以非常容易地在负载高的时候做服务的扩容，结合现在的Docker容器技术及K8S编排系统及对接口服务的监控，制定一些伸缩的规则是可以做到自动伸缩的，当负载高时自动扩容服务器，当负载低时自动缩容。

这样的好处是减少人工干预的时间，及时伸缩也能更好的节省开支，让资源得到充分利用。当然，要想基于开源技术自己构建一套好用稳定的可自动伸缩的服务体系还是很有挑战的，幸好现在很多云计算厂商可以直接提供基于k8s、docker的云服务，让构建这样一套系统变得容易起来。

可拓展

可拓展性衡量的是推荐系统应对需求变化的能力，我们可以通过如下一些策略和思路让推荐服务可以更好的拓展。

01 利用消息列队减少系统耦合

在上面图1，我们通过一个Kafka管道的模块来将推荐算法平台与推荐数据存储解耦合，而不是在推荐系统推断阶段直接将推荐结果插入推荐数据库。这样做的好处是减少系统依赖，便于问题排查。同时Kafka起到了对大规模推荐数据做备份和缓冲的作用。

02 利用解耦及庸才数据交互协议

将推荐系统服务尽量解耦，采用微服务架构，Nginx层、接口Web层、数据层等尽量独立，采用符合业务规范(基于公司自己的业务特性及技术选型)的方式交互(比如利用http，thrift，protobuf等协议做数据交换)。这样，对系统进行升级、维护、功能拓展、或者排查问题都非常方便。

现在业内有很多开源的微服务框架供大家选择，如dubbo、Spring cloud等。也可以根据自己公司需要，自行开发满足自己业务需求的微服务组件。

03 分层思想

我们可以简单将推荐系统分为三层，接口服务层处理用户的请求，数据层存储用户的推荐结果，算法模型层构建推荐模型并为用户生成推荐结果(见下面图6)。通过分层，让整个系统更有层次感，更易于理解、升级、维护，也更方便排查问题。

图6：推荐业务的分层模型

04 可适当容错及服务降级

推荐服务跟涉及到钱的业务(如会员购买，广告投放等)是不一样的，推荐结果不够精准最多是用户体验不好，不会有非常严重的投诉问题或者法律风险，所以推荐系统的容错性相对要大一些。

基于推荐系统可容错的特性及CAP理论(指的是在一个分布式系统中，Consistency（一致性）、 Availability（可用性）、Partition tolerance（分区容错性），三者不可兼得)，推荐服务对一致性的要求也没有这么高，对于推荐系统选择的分布式存储数据库，不需要强一致性，往往达到最终一致性就足够了，但是我们最好需要保证系统是满足可用性的，这样才可以时时刻刻为用户提供推荐服务。

随着产品的迭代，极大部分用户可能会升级到相对较新的版本中，很老的版本用户数肯定是较少的(相对于总用户)，对于这部分用户，我们建议产品通过各种运营或者技术手段让用户升级上去，对于不升级的用户，我们可以采用有损服务的形式为它们提供推荐服务。具体方法主要有对这部分用户关闭推荐服务和只为这部分用户提供默认推荐服务两种方式，这样做的目的主要是减少对推荐产品的维护成本。

所以，针对推荐系统可适当容错及对低版本用户可提供有损服务的特点，可以优化整个推荐系统的服务，让部分服务简化，间接提升了系统的可拓展性。

安全性

对于企业级服务，安全无小事，对于推荐系统同样存在安全隐患，提升推荐服务的安全性可以从如下几个维度来考虑。

01 接口安全

推荐服务可能由于受到攻击或者可能存在的软件bug导致对某个推荐服务的大规模请求。我们需要对推荐接口做保护，可以对同一IP地址的频繁访问做限制，或者对用户鉴权，防止系统受到恶意攻击。

对接口中涉及的隐私或者机密信息需要做加密处理。

同时，接口设计也要具备鲁棒性，对获取的推荐数据中可能存在的错误做异常保护，避免开发插入不符合规范的数据格式、数据类型等错误导致接口挂掉。

02 域名分流

对于用户量较大的APP，我们可以通过域名分流的形式对推荐接口分流，当某个域名出问题，可以快速切换到另外的域名，提供对接口更好的保护功能。

03 https

采用https协议而不是http，可以大大提升整个推荐接口的安全性，防止用户信息泄露。https性能可能会有一定损失，但是相对安全性的提升是可以忽略的。但是采用https对开发及资金成本都有更高的要求。

04 现网验证

当一个已有推荐业务做调整(接口调整、算法逻辑调整)或者新的业务上线后，一定要创造条件在现网验证一下是否正常，确保接口可以正常返回数据，并且前端看到的数据跟接口返回的数据及数据库中推荐的数据要保持一致。我们曾经出现过升级后未做验证，发现前端数据不正常的情况。

写在最后

本文从高性能、高可用、可伸缩、可拓展、安全性等5个方面对怎么设计优质的推荐服务做了详细讲解，提供了一些思路和策略，希望为设计推荐服务的读者提供一些指导。

由于本人在软件架构设计上实践经验有限，不当之处甚至错误在所难免，欢迎大家批评指正！

-end-

相关内容推荐

1.推荐系统冷启动

2.推荐算法工程师的成长之道

3.推荐系统的商业价值

4.推荐系统评估

5.推荐系统的工程实现