后端架构设计

### 目标

a）确保用户的内容数据能够很方便的被其他用户发现和获取.

b）确保内容推送是相关的，不仅在语义上，也是从用户设备的角度。

c）确保实时更新生成、推送和分析。

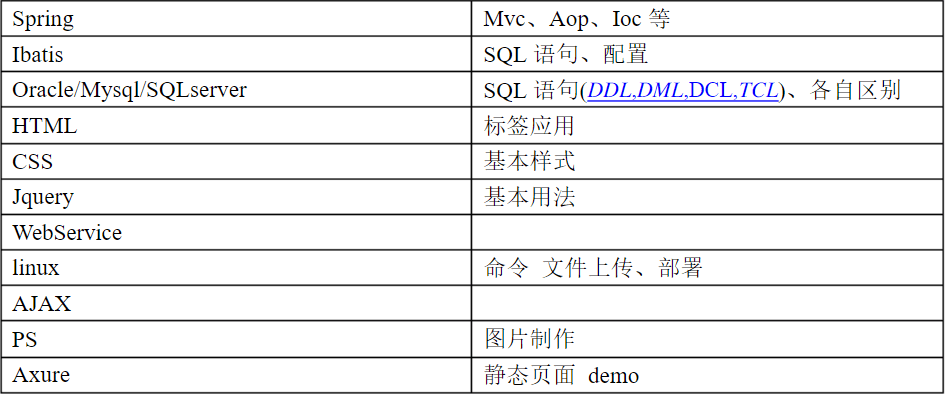
d）尽可能地节省用户的资源。

e）不论服务器负载变化如何，用户体验应保持不变。

f）确保应用整体上是安全的

总之，我们要处理一个相当大的挑战，我们必须处理不断扩大的海量用户生成的内容数据，不断增长的用户

### 我们会用到的技术列表：



### 数据存储

数据和数据模型的存储是一个好架构的关键设计之一。

第一步，我们要确定哪些数据是经常查询的热点数据，哪些不是经常需要的那些数据（如归档数据用于分析）。对于高频访问的数据，它必须总是可用，能够快速读写和水平可扩展。目前我们所有业务场景使用的都是MySQL。

数据分类：

a）主要的数据或静态形式的数据，如用户资料

b）语义数据

c）用户产生的内容数据

d）会话数据

**静态数据：**对于静态数据，最好是选择基于文档的存储方式，其中键和值都是可查询的。我们可以选择如MongoDB这种文档型数据库，选择MongoDB最大的优势是它提供了在文档级别的ACID。

MongoDB可以在多个分布式数据中心的范围内进行缩放。它将允许我们使用副本集来保持冗余，从而解决我们的可用性问题。

数据分片是一个重要的考虑因素，数据分片可以确保数据的扩展与查询速度。幸运的是，MongoDB透明的支持了数据分片。

**关联的或关系数据（核心数据）：**我们大部分数据本质上是关联的，我们使用图处理模型。我们应将这样的数据存储在图数据库，如Neo4j。图数据库支持ACID规则以及自动索引。

我们的要求是达到可用性和可扩展性。我们可能会有成百上千的并发事务，同时写入数据库，同时会有数百和数千查询请求。

Neo4j已经被设计为可水平扩展，并且有数据冗余功能来保证可用性。但到目前为止，它还不支持数据分片。我们可能需要更多的分析，才能做出抉择。其他可供选择的图数据库有FlockDB、AllegroGraph和InfiniteGraph。

**二进制数据（UGC）：**例如亚马逊S3，S3是非常流行的对象存储系统，具有可用性和弹性存储。

我们也可以考虑谷歌云存储或Rackspace的云文件等

S3已经支持数据分区。我们也可以尝试一些新的东西，如图像的自动维度识别，基于内容自动打标签等。

### Session数据

Session数据将帮助我们保持用户的状态。Session数据必须使用与服务器无关的方式，方便我们服务端可伸缩部署。确保session不会绑定到特定的节点或服务器。

我们得用一种新的方式来更新用户的实际session，如果用户的session终止，我们仍然可以帮助用户从一个地方，他离开的地方重新恢复信息。

使用MongoDB本身来保存数据。后来，我们想转移到纯粹的键值存储，如Redis。

注：所有推荐和离线作业都应该只运行在非服务节点上。

### 索引

必须设计一个实时索引，针对一段时间窗口的实时数据进行处理。首先，我们可以写一个非常简单的系统，对产生的内容数据做倒排索引。随着输入数据的增加，我们可以方便地用实时数据处理引擎取代它，如Apache的Storm，这是一个分布式的，容错和高度可扩展的系统。它可以负责生成索引的逻辑。

我们可以使用SolrCloud。它已经透明的支持分片，复制和读写方面的容错。

### 队列&消息推送

可以使用ActiveMQ，这是最可靠的队列软件。它支持集群的高可用性，支持分布式队列。

消息推送是另一个领域，要把通知发送给我们的用户。在这里我们需要估计一下规模。应该准备好支持像nps这样上亿的规模。

我们需要自己管理一些事情，特别是要保证消息传递可靠性，即使用户的设备处于离线状态。建议我们实现一个双向的系统，保持状态的通知，并在后台持久化到磁盘。所以每次一个通知失败时，它的状态都被处理并标上状态码，添加到重试队列中。最后，当通知被送达，移出重试出列。

### 缓存策略

我们的业务逻辑会在多层缓存中，并且能够智能的清除失效缓存。

应用层缓存（内容缓存）：

a）原内容是非规格化形式保存在缓存。为了安全起见，我们将永远设置一个有效的期限。

b）原内容也写在我们的数据存储区中。

使用Redis来做这个缓存，保存任何格式的数据，这使得它很容易做增量写，并支持内容feeds。

缓存代理：正确设置HTTP响应头。代理服务器有很多种，但最受欢迎的是nginx和ATS。

二级缓存（代码级缓存）：一个实体数据的本地存储，用于提高应用程序的性能。它有助于通过减少昂贵的数据库调用以提高性能，保持实体数据的本地化。如EhCache

客户端缓存：尽可能缓存其他内容，可以使用设备自己的内存，或使用SQLite。所有昂贵的对象都应该缓存。例如NSDateFormatter和NSCalendar。

### 数据压缩

可以尝试帧内压缩和帧间压缩技术。但总的来说我们可以采用zpaq和fp8来应对所有压缩需求。我们也可以尝试非常适合我们业务场景的WebP。一般情况下，我们的API会使用gzip，我们API response总是经过gzip压缩过的。

### 安全问题

1.用户数据必须加密。MongoDB和Neo4j已经支持存储加密。在这基础上，我们可以决定加密哪些用户关键信息。所有与数据库相关的传输调用必须启用加密。

2.安全套接字层：所有代理服务器的访问都应该使用SSLed。代理服务器可以充当SSL终止点。

3.API端点应该运行在非默认端口，并且必须实现OAuth。

4. DB读取都应该通过Rest endpoints。

5.有关密码的配置必须特殊处理。密码必须hashed，文件应该被限制只能在应用启动时读取。这允许我们通过文件系统权限来控制应用程序身份实例。只有应用程序用户可以读，但不能写，其他用户不可以读取。所有类似的配置都要用keydb打包并需要密码。

### 组件

以下是我们架构用到的组件:

1.负载均衡器：这层是用来转发所有对代理服务器的请求

2.代理服务器：所有即将到来的调用都必须以这里为入口。这也是我们SSL的终止点。它缓存所有基于策略定义的HTTP请求。FE层：该层运行一个node服务器。

3.数据输入引擎：这个组件涉及所有内容的输入，它做了一系列的工作:非规范化模型，转码，缓存等。

4.Rest服务：这层负责与所有DB交互，并返回数据。它的访问是受OAuth保护的。这可以用Tomcat容器以及edge缓存来实现。

5.事件处理：这层处理所有的事件，主要负责分发的功能。它读取ActiveMQ并使用通知引擎生成通知。

6.推荐引擎：这个组件通过分析所有收集到的用户动态来做推荐。根据实际收集到的动态，我们可以部署各种基于亲和力的算法。我们可以使用Apache Mahout提供的各种算法接口