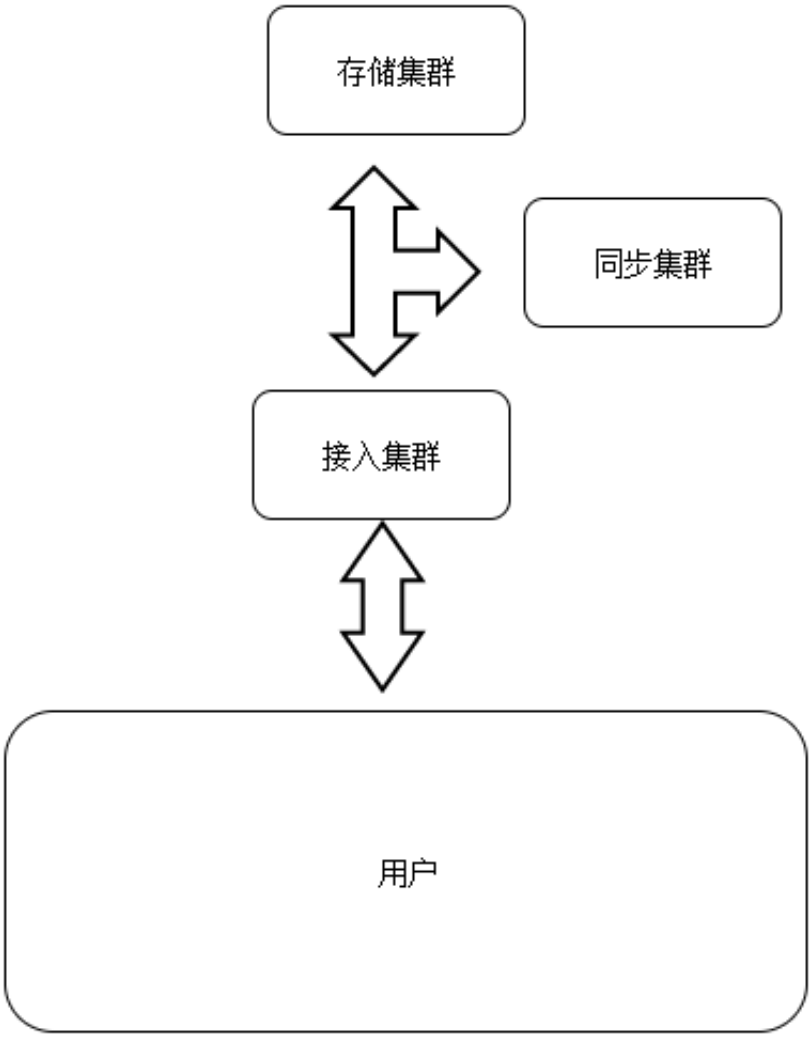
罗毅 1352388

整体架构：



分布式

目前而言，我们只是一个单机系统，设想一个用户需要8KB的内存占用，一个好点的PC机有8G的内存，则我们同时只能存在8*1024*1024/8≈100万个在线用户。如果用户达到了千万级别甚至亿级的时候，我们显然是不可能达到要求的。这里我参考了腾讯QQ的架构变化[8]，设计出了如上的架构。

分布式要解决的问题就是当你的客户端与服务端通信的时候，如何自动的分发请求到接入集群上，以及当其中某个节点挂掉之后，能够及时的切换到有效的服务器上。

服务器之间的数据同步问题，依据应用的不同，也会呈现不同的实现方案。比如，在处理用户登录这个问题上。我们首先可以向玩家开放一些默认的登录服务器（服务器的IP及PORT信息），当玩家连接到当前的登录服务器后，由该服务器首先判断自己同时连接的玩家是不是超过了自定义的上限，如果是，由向与该服务器连接着的“登

录服务器管理者”（一般是一个内部的服务器，不直接向玩家开放）申请仲裁，由“登录服务器管理者”根据当前各登录服务器的负载情况选择一个新的服务器IP和PORT信息传给客户端，客户端收到这个IP和PORT信息之后重定向连接到这个新 的登录服务器上去，完成后续的登录验证过程。

这种方案的一个特点是，在面向玩家的一侧，会提供一个外部访问接口，而在服务器集群的内部，会提供一个“服务器管理者”及时记录各登录服务器的负载情况以便客户端需要重定向时根据策略选择一个新的登录接口给客户端。

采用分布式结构的好处是可以有效分摊整个系统的压力，但是，不足点就是对于全局信息的索引将会变得比较困难，因为每个单独的底层逻辑服务器上都只是存放 了自己这一个服务器上的用户数据，它没有办法查找到其它服务器上的用户数据。解决这个问题，简单一点的作法，就是在集群内部，由一个中介者，提供一个全局 的玩家列表。这个全局列表，根据需要，可以直接放在“服务器管理者”上，也可以存放在数据库中。

负载均衡

各大云服务提供商均提供了负载均衡的功能，例如腾讯云[9]和阿里云[10]，我们可以直接使用其提供的方案，这里不再赘述。

消息一致性

根据Google App Engine的co-founder Ryan Barrett在2009年的google i/o上的演讲《Transaction Across DataCenter》（视频：http://www.youtube.com/watch?v=srOgpXECblk）

要想让数据有高可用性，就需要冗余数据写多份。写多份的问题会带来一致性的问题，而一致性的问题又会带来性能问题。从上图我们可以看到，我们基本上来说不可以让所有的项都绿起来，这就是著名的CAP理论：一致性，可用性，分区容忍性，你只可能要其中的两个。

Consistency：一致性，这个和数据库ACID的一致性类似，但这里关注的所有数据节点上的数据一致性和正确性，而数据库的ACID关注的是在在一个事务内，对数据的一些约束。

Availability：可用性，关注的在某个结点的数据是否可用，可以认为某一个节点的系统是否可用，通信故障除外。

Partition Tolerance：分区容忍性，是否可以对数据进行分区。这是考虑到性能和可伸缩性。

对于大数据量，高并发的聊天室应用来说，数据分区是必不可少的，同时可用性对于用户体验来说也是十分重要的，故而我们可以牺牲一部分的一致性来达到我们的要求。

对于牺牲一致性的情况最多的就是缓存和数据库的数据同步问题，我们把缓存看做一个数据分区节点，数据库看作另外一个节点，这两个节点之间的数据在任何时刻都无法保证一致性的。在web2.0这样的业务，开心网来举例子，访问一个用户的信息的时候，可以先访问缓存的数据，但是如果用户修改了自己的一些信息，首先修改的是数据库，然后在通知缓存进行更新，这段期间内就会导致的数据不一致，用户可能访问的是一个过期的缓存，而不是最新的数据。但是由于这些业务对一致性的要求比较高，不会带来太大的影响。

一个可用方案是两阶段提交协议。但是在第一阶段和第二阶段之间，数据也不可能是一致的，可能导致异常，我们也需要做相关处理。

国外有的架构师有两种方案去解决CAP的限制，但是也是比较适合特定的业务，而没有通用的解决方案。

垃圾消息过滤

一句话，通过机器学习。

首先一个问题，如何定义这是一条垃圾信息？通过机器进行学习，简单的搜索一下，可以发现有人尝试过逻辑回归模型[1]、支持向量机[2]等模型。至于真正工程上的应用，我了解了一下facebook的做法[3]，描述如下：

Facebook 使用了一个叫做 sigma 的系统来抵制这些垃圾信息。这个系统安装在 2000 多台机器上面，Facebook 用户做的任何事情，都会经过 sigma 系统分析处理，比如评论、链接、朋友请求，都会被这个系统进行判断，是正常行为、滥用行为还是有问题的行为。

举个例子说，比如发送朋友请求，Facebook 的系统会自动判断一下：如果这个人的朋友请求都被别人拒绝了，他再发送朋友请求是不会被批准的。如果一个人发送的朋友请求十个有九个都被拒绝了，那么他下一次的朋友请求就会被系统拒绝。当然这个系统还有其他的判断信号。

它是一个机器学习系统，通过你之前发的朋友请求拒绝概率高低来判断你被拒绝的概率有多高。如果这个比率很高，Facebook 会让你进行手机短信或其他方式认证，来验证是软件还是真人发送的，以此判断你是不是真的要发送朋友请求，比如你发出的朋友请求对象与你没有任何共同好友，那就可能是一个不合理的请求。

基本上，你在 Facebook 上做的任何事情，都会经过这个系统来分析、预测、决定是否允许你发出信息，借此希望会减少生态圈中的骚扰行为。当时 Facebook 每天有上百亿次的信息发生要通过这个系统进行判断。

这个系统中有些是人为规则也有机器算法，请求通过和拒绝就是一个迅捷数据组（Scrum）。任务通过，则说明这个任务是一个对机器学习来说的正样本，被拒绝则是一个负样本，很像 0 和 1。比如发送朋友请求如果被接受，y 值是 1，如果被拒绝就是 0。如果是评论和点赞，系统就能寻找 y 值，用户发送的不当信息就会被删除。

另外一个方法是通过一些异常行为的分析、数据挖掘的方法来分析用户的异常行为。比如一个人发的同样类型评论非常多，所有评论里都有一个相似链接，这就非常有问题。正常操作不会在不同人的主页上留同样的评论，这显然属于异常行为，我们不会允许。

机器学习的东西过于复杂，如果要再继续说下去的话，估计足够写一篇研究生论文了，限于篇幅，这里点到为止，但是总结来说就是通过机器学习的方法来过滤垃圾信息。

ID 分配

在一个分布式系统中，如何生成全局唯一的ID？
在单机系统中（例如MySQL实例），unique ID的生成是非常简单的，直接利用MYSQL自带的自增ID功能就可以实现。
但在一个存在多个Shareds的分布式系统（例如多个MYSQL实例组成一个集群，在这个集群中插入数据），这个问题会变得复杂，所生成的全局的unique ID要满足以下需

求：
1. 保证生成的ID全局唯一
2. 今后数据在多个Shards之间迁移不会受到ID生成方式的限制
3. 生成的ID中最好能带上时间信息，例如ID的前K位是timestamp，这样能够直接通过对ID的前k位的排序来对数据按时间排序
4. 生成的ID最好不大于64bits
5. 生成ID的速度有要求。例如，在一个高吞吐量的场景中，需要每秒生成几万个ID（Twitter最新的峰值到达了143199 Tweets/s，也就是10万+/s）
6. 整个服务最好没有单点

一种简单的做法是利用一个中心服务器来统一生成 unique ID.。但这种方案可能存在单点问题；另外，要支持高吞吐率的系统，这个方案还要做很多改进工作（例如，每次从中心服务器批量获取一批 IDs，提升 ID 产生的吞吐率）。

目前而言，已经有一个比较成熟的解决方案，叫做Flake Ids[4]，至于其具体开源实现有以下几种：
- boundary/flake
- jamesgolick/lexical_uuid.erl
- jamesgolick/lexical_uuid
- simpleflake
- noeqd
- snowflake
- simple_uuid
- skuld's internal implementation
- datomic's implementation

其中SnowFlake是Facebook所采用的方法，其算法在文章http://darktea.github.io/notes/2013/12/08/Unique-ID中有描述，由于本门课是软件复用，我就不再具体解释算法。其开源地址在https://github.com/twitter/snowflake/tree/b3f6a3c6ca8e1b6847baa6ff42bf72201e2c2231，我们可以直接拿来使用

安全

可以使用Apache Eagle[5]。

Eagle 是开源分布式实时Hadoop数据安全方案，支持数据行为实时监控，能立即监测出对敏感数据的访问或恶意的操作，并立即采取应对的措施。

[1]: 基于逻辑回归模型的中文垃圾信息过滤

[2]: 基于支持向量机的垃圾信息过滤方法
[3]: 占领微博微信们的信息垃圾，Facebook 是怎么处理的？
[4]: Distributed System Building Block: Flake Ids
[5]: Apache Eagle——eBay开源分布式实时Hadoop数据安全方案
[6]: QQ与微信架构的惊天秘密
[7]: 微信后台存储架构
[8]: 1.4亿在线背后-QQ IM后台架构的演化与启示
[9]: 负载均衡——腾讯云平台
[10]: 负载均衡——阿里云平台