**租约机制简介**

内容来自http://blog.csdn.net/kevinfankai/article/details/4024937

# 背景和介绍

缓存是计算机里广泛使用的一种技术，对降低读取延迟、网络流量和服务器负载都非常有效，但也带来了一致性(Consistency)的问题。所谓一致就是客户端总能读到最新的数据，使用缓存后有可能服务器端的数据已经被修改，但客户端仍然从缓存中读取陈旧的数据。为了保证一致性，有两种常见的解决办法，第一种是轮询（Polling），即每次读取数据时都先询问服务器数据是不是最新的，如果不是就从服务器传输新数据，这种方法需要每次读取数据时都与服务器通信。另一种方法就是回调（Callback）或者无效化（Invalidation），就是由服务器记住有哪些客户端读取了数据，对数据做修改时首先通知所有这些客户端数据已经失效，这种方法的问题在于服务器需要记住所有读取过数据的客户端，这是很大的负担，更严重的是，一旦有客户端联系不上或者丢失了客户端的信息，修改操作就无法继续。

1989年斯坦福大学的Cary G. Gray和David R. Cheriton提出了利用租约来维护缓存一致性的方法。所谓租约，其实就是一个合同，即服务器给予客户端在一定期限内可以控制修改操作的 权力。如果服务器要修改数据，首先要征求拥有这块数据的租约的客户端的同意，之后才可以修改。客户端从服务器读取数据时往往就同时获取租约，在租约期限内，如果没有收到服务器的修改请求，就可以保证当前缓存中的内容就是最新的。如果在租约期限内收到了修改数据的请求并且同意了，就需要清空缓存。在租约过 期以后，客户端如果还要从缓存读取数据，就必须重新获取租约，我们称这个操作为“续约”。

在租约期限内，客户端可以保证其缓存中的数据是最新的。同时，租约可以容忍各种非拜占庭式失效（机器崩溃、网络分割等）。如果客户端崩溃或者网络中断，服务器只需要等待其租约过期就可以进行修改操作。如果服务器出错丢失了所有客户端的信息，它只需要知道租约的最长期限，就可以在这个期限之后安全的修改数据。与回调方式相比，服务器只需记住还拥有租约的客户端即可。

租约与带期限的锁非常相似，但更加灵活，因为租约还提供了“寻求同意”的机制（我觉得可以称为“带期限可妥协的锁”）。服务器还可以实现多种租约，比如“写租约”和“读租约”，并保证一个时间段内只有一个写租约或者多个读租约，这就相当于是单写者多读者的锁协议。

因为租约是基于时间的，因此其有效性需要系统时间来保证。如果服务器的时钟快而客户端时钟慢，那么有可能服务器认为一个租约已经过期而客户端仍然认为其有 效，就可能导致错误。对这种情况就必须通过时钟同步协议来解决了，不过这种情况很少见。一般情况下，我们可以认为一个分布式系统的时间是同步在一个很小的时间差e之内，只需把这个e考虑到租约期限内即可。

# 租约属性和管理

租约的属性和管理有多种选择，首先要考虑的就是租约期限的长短。

一般情况下，应当选择较短的租约期限。与长租约相比，短租约有三个优点。首先，在失效情况下修改操作往往需要等待租约过期，因此短租约就意味着更短的失效延迟。其次，就算一个客户端已经不再需要读取数据，但在其租约过期前，任何的修改操作仍然需要征求它的同意，这种情况叫做“假共享”，显然租约期限越长，这个问题就越严重。最后，短租约也使得服务器要维护的客户端信息更少。然而短租约也意味着更大的续约开销， 因此对于要反复读取却很少修改的数据，长租约会更有效。因此，对租约期的选择要权衡失效延迟、假共享开销和续约开销等多个因素，服务器可以根据数据访问特 性和客户端的性质灵活设置期限。事实上，如果我们把租约期限设为零，就相当与轮询，此时修改操作随时可以进行，而读取数据总是要联系服务器。如果把租约期 限设为无限长，就相当于回调。

除了期限的选择，还有很多管理选项。对客户端来说，可以选择 是否续约、何时续约以及是否同意修改等。比如为了减少读取延迟，客户端可以在租约过期前就续约，不过这样可能加重服务器的负担。对服务器来说，可以选择是否发放租约、租约覆盖粒度以及对如何进行修改操作。比如在收到修改请求后，服务器可以不征求客户端同意，而是简单的等待所有租约过期（等待时不再发放新租 约以避免无限期的延迟）。对于“安装文件”，也就是修改极少的文件（比如头文件、库文件），服务器可以用一个租约来覆盖一批文件，同时定期广播续约通知来节省开销，如果需要修改数据，就停止广播并等待租约过期即可。

# 互联网的一致性问题

在互联网环境下，一致性问题更加复杂，因为网络延迟比局域网要大的多，客户端失效和网络分割都很常见。因此很多情况下，我们都只保证缓存的弱一致性，也就是不保证客户端总能读到最新的数据，只是尽量保证其读到的数据还不是非常滞后。相应的，我们把前面使用的一致性称为强一致性。目前最常用的保证弱一致性的方法就是生存期（TTL)， 即读取数据的时候会指定生存期，在生存期内客户端直接从缓存中读取数据，之后必须与服务器通信验证缓存有效性或者获取最新数据。很显然，我们可以给变化较多的数据分配较短的生存期来尽量减少客户端读取过期数据的几率，而给变化较少的数据分配较长的生存期来减少读取延迟和服务器负载。

1992年CMU的Vincent Gate在Alex项目中实现了可变生存期（Adaptive TTL)，这个技术基于这样的观察，就是新数据比旧数据更容易被修改。 在Alex中采用了更新阈值（update threshold）的概念，把生存期设定为其缓存时间的一个百分比。假设一个刚刚向服务器验证过的数据已经在缓存中存在了10个钟头，其更新阈值为 20%，那么其生存期就是2个钟头。采用可变生存期技术并不能减少网络流量，但是可以比普通的生存期技术减少服务器负荷。

虽然弱一致性模型已经满足我们日常浏览网页的需求，但还是有一些应用会要求强一致性。局域网情况下的方法很难直接扩展到互联网环境。轮询方法的读取延迟太 大。回调方法不但记录所有客户会使得服务器难以承受，经常出现的网络分割更使得修改操作无法继续。短租约导致的网络通信和服务器负荷都太大，而且如果租约期限小于网络延迟的话，那么除了增加服务器负荷外没有任何作用。长租约又相应的使得失效延迟和假共享的问题更加严重，而且服务器要记录大量客户端数据。既然有可变生存期，那自然的也有了可变租约（variable leases）的想法。1998年威斯康辛大学麦迪逊分校的Pei Cao和Chengjue Liu就提出了一种叫two-tier的方案，就是区分只是偶然读取数据的客户端和真正需要强一致性的客户端，只对后者才发放租约（由客户端显示的提出请求）。更进一步的，他们分析了如何根据不同情况调整租约属性，例如数据修改减少时增加租约期限，存储空间不够时则缩短租约期限甚至要求客户端放弃租约。

# 卷租约

在1999年的论文中，德克萨斯大学奥斯汀分校的Jian Yin等提出了互联网环境下保证强一致性的卷租约机制。

所谓卷租约（volume leases）， 与上面提到的针对“安装文件”的租约有点类似，即由一个租约会覆盖多个相关文件，其期限较短，一般是数十秒，一般要在过期前续约。卷租约因为期限短，续约操作就比较频繁，但用户往往会同时读取一个卷下的多个文件，因此这个开销分摊到了多个文件上，依然是可以接受的。如果光采用卷租约，会带来非常严重的假共享问题，因为拥有卷租约的客户端各自关注的数据可能很不相同。因此服务器还另外提供了对象租约（object leases），就是只覆盖一个文件的普通租约，期限一般较长，可以是数小时甚至数天，以取得较小的续约开销。客户端要保证缓存有效，就必须同时拥有对象租约和卷租约。修改操作需要征求同时拥有卷租约和对象租约的客户端的同意，如果出现客户端机器失效或者网络分割的情况，最多只需等待卷租约过期，就可以安全的修改数据。

一旦卷租约过期，服务器就认为相应的对象租约也都过期，客户端重新获取卷租约时，会检查相应的对象租约的有效性，如果数据没有修改就续约，否则清空缓存。事实上，卷租约和对象租约类似于心跳和回调，前者主要用来确定客户端的状态，后者用来定位对数据有兴趣的客户端。通过两种租约的结合，可以较好的平衡失效延迟和通信开销。实验数据表明，要取得同样的失效延迟时间，卷租约机制可以比普通租约机制减少三成到四成的消息数。粗一想或许还会奇怪，加了一个租约怎么能够减少通信呢？这是因为要取得同样的失效延迟，普通租约就必须把租约期限设定在延迟期限内（比如20秒），而如果使用卷租约，只需把卷租约的期限设定在延迟期限内，而对象租约的时间可以设的很长。因为卷租约的通信开销有好几个对象分摊，而对象租约的开销因为期限很长事实上非常小，所以总的开销会小于只使用普通租约的情况。

# 租约的其他应用

以上我们只讨论了如何用租约来维护缓存一致性，其实租约的应用范围非常广泛。

在 Frangipani分布式文件系统中实现了一个分布式的锁，客户端在获取锁之前，首先要获取一个租约，并且必须在租约过期前续约。这个客户端获得的锁都与这个租约相关联，如果租约过期了，锁服务器就会自动的释放这些锁。这里的租约就是对锁有效性的一个保证，通过维护客户端租约，避免了为每个锁维护期限的开销。这里的租约就相当于心跳。

在gfs中，每个文件块都有多个副本分布在多个chunkserver上，在 并行追加时必须有一个全局统一的追加顺序。当然这个顺序可以由master来确定，但是这样会大大增加master的负荷。另一种方法可以由多个 chunkserver通过一致性协议（比如Paxos）来达成一个一致，但这样开销太大。gfs使用了租约机制，就是对每个文件块，由master向一个chunkserver发放租约，在租约期限内就由它来负责并行追加操作的顺序。chunkserver正常运行时可以一直续约，如果出现了机器失效或者网络分割的情况，master就在租约过期以后把租约交给另外一个chunkserver。在某些情况下，master也会联系拥有租约的 chunkserver，请它们提前释放租约。

很多情况下，系统中已经有一个保证一致性的中心服务，如某个单一服务器或者是实现了Paxos协议的一组服务器，但如果所有的功能都需要通过这个中心服务，很容易造成性能瓶颈。为了提高效率和扩展性，可以通过租约把一致性扩展到更多服务上。事实上用租约来维护缓存一致性也是相当于把服务器上的数据一致性扩展到了客户端。

# 到底租约是什么？

在很多时候，租约的定义似乎很模糊，有的时候租约类似心跳，有的时候又类似于锁。到底租约的本质是什么呢？

回到租约最原始的定义：租约就是在一定期限内给予持有者特定权力的 协议。我觉得这里的期限就是租约的根本特性，正是这一特性使得租约可以容忍机器失效和网络分割。在期限之内，租约其实就是服务器和客户端之间的协议，而这 个协议的内容可以五花八门。如果协议内容是服务器确认客户端还存活，那么这个租约的功能就相当于心跳；如果协议内容是服务器保证内容不会被修改，那么这个 租约就相当于读锁；如果协议内容是服务器保证内容只能被这个客户端修改，那么这个租约就相当于写锁。租约这种灵活性和容错性，使其成为了维护分布式系统一致性的有效工具。