二级共享内存集群实现

作者: freeeyes，changsong

1. 设计目标

随着应用的越来越复杂，对数据同步和存储的要求越来越多，有时候我们面对大量复杂的数据加载和数据同步代价，尤其是如果单服务器无法承载大量应用数据的时候，我们如何可以实现动态扩展服务器已达到负载均衡的目标。

此设计只针对平均分布应用数据而建立，上不考虑特例服务器集群结构。

考虑到目前的需求，参考了目前流行的几种中间件模式，比如ICE, TAO和Tuxedo，由于本人有一些ICE和TAO的实际商业化经验，所以觉得纯用中间件无法解决多级数据缓冲的问题，而且中间件部署复杂，应用中的最大管理问题是学习成本较为高昂，而我们的实际应用并不用到太多中间件的高级功能。所以暂不考虑中间件。用最简单的实现去解决需要处理的需求。

此设计主要解决以下几个问题。

1. 让应用剥离对IO数据更新的管理，应用只专注于与其自己用的共享内存的管理，不在负责共享内存新数据的加载和存储。
2. 提供专门的共享内存维护进程（以下简称此进程为watch），负责用于同步上级数据源，
3. 提供可以动态伸缩的多服务器架构。
4. 提供可管理动态伸缩的服务器监控者（以下简称Grid Proxy）,负责可配置动态的服务器增减。
5. 设计实现原理

数据源

可以是

Memcached

Oracle

Mysql

File

服务器

共享内存A

应用程序A

Watch

Grid Proxy

用于管理所有注册的watch

这是一个具体实现的最小单元

此单元可以平行扩展

服务器可以是多台平行扩展，这里应用程序A对数据的依赖全部从共享内存A获得，这里的应用共享内存可以是多个。可以是一个M对N的关系。目前的结构只针对读多写少的情况，写多读少的情况，需要对Wacth有一些特殊的设计，这里暂不论述。

对于此架构，应用程序并不是它管理的内同，应用逻辑和此架构是相对独立的。此架构对于共享内存，只做同步管理。同步规则是定时memcopy同步远程的数据源中的对象。达到同步数据的目的。

同步规则，在Wacth里面实现。

Watch根据同步配置文件，从远端数据源中获得相应的自己感兴趣的数据，在获得数据源数据的时候，只存在本进程的内存空间，当同步完成，再统一刷到共享内存A中完成数据同步。

Grid Proxy是负责监控每个Watch的监控状态。当新的Watch启动的时候，会自动连接到Grid Proxy，Grid Proxy收到新Watch注册信息，会通知所有已注册的Watch，这些Watch会根据逻辑重新加载自己感兴趣的数据，达到自动数据平衡的能力。如果一个Watch因为某些原因离线或者挂掉，Grid Proxy也会通知所有的Watch知道，从而达到数据均衡分布。

这样做，当数据源宕机，或者本地宕机，都可以迅速恢复，保证数据最大可用性。二级缓冲的最大作用是减少IO通讯量。

1. 设计需要的进程以及进程内部结构

共享内存，对于次架构，共享内存只是一个二进制块，对于C++和C而言，我只要知道共享内存单元的大小和count即可，那么Sizeof(T)\*Count 便可获得当前共享内存的大小，这些完全可以作为Watch的配置文件。这样可以管理若干个共享内存的同步操作。( linux下考虑每个操作子进程操作)。

实际每个子进程，对应一个同步规则。

可以从远端数据源中获得当前共享内存需要的数据，这部分数据被当前子进程new出来，在本地内存中管理，当所有感兴趣的内存同步完成后，再一次性memcopy到共享内存中去，这次拷贝需要进行文件锁。拷贝完成后，逻辑进程再进行访问，因为memcopy远比获取同步修改要快的多，所以同步时间比较短。具体等待实测数据附证。

Grid Proxy是一个完整的监控Watch的工具，它的主要工作是监控是否所有的Watch都在健康工作，同时如果有新的Watch添加进集群，那么会通知其他所有的Watch，稀释或者浓缩共享数据。