skynet 的新组播方案

最近在做 skynet 的 0.2 版。主要增加的新特性是重新设计的组播模块。

组播模块在 skynet 的开发过程中，以不同形式存在过。最终在 0.1 版发布前删除了。原因是我不希望把这个模块放在核心层中。

随着 skynet 的基础设施逐步完善，在上层提供一个组播方案变得容易的多。所以我计划在 0.2 版中重新提供这个模块。注：在 github 的仓库中，0.2 版的开发在 dev 分支中，只到 0.2 版发布才会合并到 master 分支。这部分开发中的特性的实现和 api 随时都可能改变。

目前，我打算提供 publish/subscribe 风格的 API 。组播消息通过 publish 接口发布出去，所有调用过 subscribe 接口的服务都可以收到消息。

在设计上，每个 skynet 节点都存在一个用于组播的专门服务。组播并非核心层模块，组播消息不是直接发布出去的，而需要通过组播服务进行。

在设计组播模块的结构时，我们需要分两种情况处理。对于在节点内部传播的消息，由于在同一进程内，所以会共享一块内存，用引用计数管理声明期。对于在网络中传播的消息，跨节点时需要复制一份消息内容，用于网络投递。

先来看较简单的内部消息的传播过程：

单个 skynet 节点中有一个唯一的 multicastd 的服务，在首次请求组播服务时会启动。

首先，必须有人请求创建一个 channel 。这个 channel 是一个 32bit 的数字，循环后可复用。低 8bit 必须是节点号，这样可以保证不同节点上创建出来的 channel 号是互不相同的。

multicastd 是用 lua 编写的，用了一张表记录 channel 号对应的订阅者。这里暂不考虑跨节点订阅的情况，这张表里全部记录的是本地服务地址。

multicast 模块的 api 全部在 lualib/multicast.lua 中以库形式提供。一共就是 new, delete, publish, subscribe, unsubscribe 几个。这些 api 最终都是把请求转给 multicastd 去处理。也就是说，发布组播消息，其实比普通的点对点消息传播路径要长。

所以值得注意的是：即使在同一服务中，先发送的组播消息未必比后发出的点对点消息要先抵达。但是，对 multicastd 发出 publish 请求，是一个有回应的 rpc call 。所以同一个 coroutine 中按次序做 publish 和 send 操作还是能保证时序的。这也是为什么 publish 方法要设计成会阻塞住当前 coroutine 运行的原因。

组播消息在提交到 multicastd 之前，在发起方就已经被打包成一个 C 结构指针。

```
struct mc_package {
    int reference;
    uint32_t size;
    void *data;
};
```

注意被打包的是指针而不是结构。这样才能做引用计数。消息内容也是用另一个指针间接引用的，这样方便消息打包。

由于在服务间传递的是一个指针，所以这条消息是禁止传播到进程之外的。这点由 multicast 库保证（用户不得直接向 multicastd 服务发送任何请求）。

multicastd 收到 publish 请求后，会统计本地订阅者的数量，给数据包加上准确的引用次数值，并将消息转发给所有订阅者。因为仅仅是转发一个指针，比转发消息体要廉价的多（这也是组播服务的存在意义）。

订阅和退订 channel 也是通过 multicastd 进行的。由于时序问题的存在，所以订阅和退订都被实现的有一定的容错行。重复订阅和重复退订（以及删除 channel 和退订的时序）都会被忽略。这里退订被实现为非阻塞的（不会打断发起退订方的 coroutine ，不必等待确认），是因为它需要在 gc 的流程中进行，而 gc 的执行上下文是不可控的。

如果组播只存在于进程内，那么以上都很容易实现。不可忽略的复杂性在于跨进程的多节点组播。

为了可以让多个节点间的组播可行。我为 skynet 增加了一个叫做 datacenter 的基础组件（当然这个组件在以后别的设施中也将用到）。datacenter 在 master 节点上启动了一个 lua 实现的树结构内存数据库。它对整个 skynet 网络都是可见的。它就像一个全局注册表一样，任何接入 skynet 网络的节点都可以读写它。

每个节点的 multicastd 启动后，都会把自己的地址注册到 datacenter 中，这样别的节点的 multicastd 都可以查询到兄弟的地址。

如果 multicastd 收到订阅请求后，它会先检查 channel 是不是在本地创建的。如果不是，除了要维护本地订阅这个 channel 的订阅者名单外，在本地第一次订阅这个 channel 的同时，要通知 channel 的所有者本节点要订阅这个 channel 。每个 channel 的管理者地址都可以在 datacenter 内查到。

收到远程订阅请求后，本地管理器仅记录这个 channel 被哪些节点订阅而不记录记录在每个远程节点上有具体哪些订阅者。当消息被组播时，对于有远程订阅者的 channel ，需要把 `struct mc_package` 的数据内容提取出来打包传输。这里的一个实现上的优化是，直接把消息走终端客户的组播通道。因为 multicastd 本身不会按常规用户那样订阅消息，所以数据格式可以不同（常规客户订阅的消息收到的是带引用计数的结构指针，而 multicastd 收到的就是消息本身）。

对于发布一个远程组播包（发布者和 channel 的创建位置不在同一个节点内），直接把包投递到 channel 所有地，看成是从那里发起的（但发送源地址不变）。

对于订阅者，它收到的组播消息是从专门的协议通道（PTYPE 为 2 ）获取的。为了使用灵活，并没有规定协议的具体编码形式。需要订阅者自己注册 pack 和 unpack 以及 dispatch 函数。默认使用 lua 编码协议，但可以改写。

因为 channel 是一个 32bit 整数，而组播消息是不需要应答的，所以可以复用消息的 session 字段，这也算是一个小优化。

---

multicast 目前仅提供 lua 层面的 API 。虽然理论上是可以通过 C 层直接收发包，但意义不大。API 以对象形式提供，每个 channel 都是一个 lua 对象。如果创建对象时没有填具体的 channel 编号，就会调用本地的 multicastd 创建出一个新的 channel 。对象在 gc 时不会销毁 channel （因为这个 channel 号有可能被传递到别的服务中继续使用），需要显式的调用 delete 方法销毁。但 channel gc 的时候，如果曾有订阅，会自动退订。

已知的设计缺陷：

由于 multicast 不在核心层实现，所以当一条组播消息被推送到目标消息队列中，在处理消息之前，服务退出。是没有任何渠道去减消息的引用。这在某些边界情况下会导致一定的内存泄露。

如果要解决这个泄露问题，必须在发送消息时记录下消息发给了谁（因为消息订阅者可能发生变化）。然后再想其它途径去释放它。做到这一点，除了结构上增加复杂度的成本外，运行成本的增加可能也会抵消掉组播的好处（减少数据复制带来的成本）。

所以，暂时不考虑完全解决这个问题。

云风 提交于 April 30, 2014 02:46 PM | 固定链接

COMMENTS

构建的思维很清晰，实际使用可能存在诸多优化的空间。
暂时处于膜拜的阶段，阅而习之。

但具体的适用处还是不甚明了。

相较于socket的机制，好像易用很多，重构出好多好方法。

如果作为demo测试的话，在什么平台验证比较好呢，sever的级别多大才适合，望指教一下

Posted by: Anonymous | (1) June 5, 2014 12:49 PM

POST A COMMENT

非这个主题相关的留言请到：留言本

名字：

Email 地址：

为了验证您是人类，请将六加一的结果（阿拉伯数字七）填写在下面：

URL:

记住我的信息？

留言：
（不欢迎在留言中粘贴程序代码）

提交