I'm OWenT
Challenge Everything

关于BUS通信系统的一些思考（一）

Table Of Content

目录

概述

如何保证一个进程或线程能安全稳定地把一段消息发送到另一个进程和线程，甚至是另一台机器的进程或线程，再或是要通过代理转发到另一个进程或线程，一直是一个比较麻烦的问题。

最近看了一些和BUS系统有关的东西。对于游戏服务器集群所使用的BUS通信系统有一些想法和思路，但是由于我对其他类型的业务和框架不是很熟悉，有些想法可能仅是站在游戏服务端的立场上，所以可能有些地方还有一些局限性。

BUS系统在我最理想的状态是，只需要两方的ID，发送屏蔽消息流转细节（废话，现在是个这种系统就有这功能），连接关系尽量简单，按需可用，尽量保证可靠性，更要的是逻辑简单。但是一直没找到一个特别完美的解决方案。

一些概念性的东西直接维基百科即可，比如消息队列,我这里就不再另外贴概念了。

静态共享内存通道BUS

首先是我们公司这里所采用的BUS公共组件是基于共享内存的。 整体的设计结构大致分为两种，**一种是同一物理机下的进程间通信**，**另一种是不同物理机进程间的通信**。

单机节点间通信

每个节点都有一个32位的ID，然后每两个节点之间都会建立两条通道。

节点A->节点B: 通道一：消息从A流向B
节点B->节点A: 通道二：消息从B流向A

如上图所展示，对节点A来说，通道一是发送节点，通道二是接收节点；对节点B而言，通道一是接收节点，通道二是发送节点。

同时，这里的**通道由共享内存**构建，这么做的好处是即便程序崩溃了，消息也不会丢失。另外节点对消息的处理使用*无锁队列*实现。

现在无锁队列已经有很多种实现了，云峰曾经写过一些分享,他的skynet[1]里也有一个GCC下的进程间通信的无锁队列实现，这种实现很像zeromq[2]的inproc协议。

上面这些无锁队列都是运用于进程内存的，还有可以运用于共享内存的例子，具体可以参照gaccob的blog《游戏服务器系列（1）——无锁的共享内存通信》。或是boost[3]库里的进程间通信部分和无锁容器部分。

但是由于这里的BUS都是单读单写，所以实现起来也比较简单暴力。并且通信和socket一样是**面向连接**的。

跨机进程间通信

上面已经解决了单机进程间通信，然后要解决跨机器通信的时候不得不借助网络，为了减少网络节点铺成网状，设计了一个代理节点。

代理节点会监听端口，进行消息转发，这样就把内部的多个节点对外都收敛到了代理结点上。同时对每个对外的节点设计了一个缓存通道，并且对消息编号，发出的消息如果没有回执尝试重发，收到重复序号的消息则忽略。 这样保证了只要节点发送接口返回成功，消息就

一定不会丢，一定存在再某个信道里，并且消息由序号保证不重复。

节点A->物理机一代理节点: 共享内存通道
物理机一代理节点->物理机二代理节点: 网络通道
物理机二代理节点->节点B: 共享内存通道
节点B->物理机二代理节点: 共享内存通道
物理机二代理节点->物理机一代理节点: 网络通道
物理机一代理节点->节点A: 共享内存通道

总结

这种设计方式非常简单高效，而且最大的优点是保证里消息的可靠性。但是有两个问题，第一个问题是节点间两两互相建立信道，意味着信道很多，而且是网状的。虽然在物理机之间通信的时候做过一次收敛，但是内部还是网状的连接，同时为了维持这些信道，内存开销不小。 另一个问题就是标题里提到的静态，至于为什么叫**静态**的呢？因为节点连接的每个通道都要事先建立。所以这也就引出了接下来我想谈到的动态共享内存BUS。

动态共享内存通道BUS

为了解决**静态**的问题，我们这的一位专家设计了动态共享内存BUS系统。其实原理很简单，就是再静态共享内存通道的基础上，给代理节点增加通道管理功能。

首先所有共享内存通道都由代理节点分配和管理，管理过程大致分几步：

1. 代理节点通过网络监听管理端口
2. 当子节点上线时连接到代理节点管理端口，发送注册消息，代理节点分配消息通道
3. 当子节点之间通信时首先检查本地有没有直连通道，有的话通过直连通道发送消息，否则发给代理节点，由代理节点转发
4. 代理节点收到消息转发请求以后先检测两边通信的节点是否都是下属的子节点，如果是就新建两个通道，并通知子节点下次通信用这两个通道作直连通道，然后转发消息。否则就把消息转发到远程机器的代理节点

这么做简单地说就是让代理节点来管理共享内存通道，但是带来地另一个问题就是程序恢复时怎么恢复通道。一种方式时通道信息也记录到共享内存里去，但是这边地实现比较暴力一点，会根据通信双方节点ID和代理节点配置算出来一个唯一共享内存ID。只要配置不变，ID不变，共享内存Key是不变的。

这种模式gaccob也提供了一个简单的例子。《游戏服务器系列（6）——共享内存通信之二》

另一个问题就是为了性能这里的实现里网络通信没有把转发消息缓存起来，所以跨机器通信的消息由丢失的风险。

另一个问题就是还是没有解决单机内**网状的共享内存通道连接**关系。

高性能开源消息队列组件- ZeroMQ

对于BUS系统和消息队列，也有一些很有名气的开源组件。比如这个，ZeroMQ。

ZeroMQ最大的特点就是是面向消息的，和前面提到的两种还有socket的通信方式完全不一样。

不过不得不说，ZeroMQ确实把通信模式总结得非常好，支持请求-回应模式、发布-订阅模式、路由消息等。而且它的上层API完成了一个非常重要的功能，就是使用zmq的ROUTER sock可以把接收方路由节点可以收敛到一个端点上。

但是它的面向消息的设计带来一个问题就是，逻辑过于重了。特别是它的很多模式都是基于*同步操作*的，而且**弱化了连接**的概念。在要构建服务器的需要的异步操作里不得不用一些底层的操作，并且它的通信模式对消息内容还有些**潜规则**（比如REQ消息会有一个节点名称包头和一个空包头、路由消息会有一个节点名称包头等）

另一个问题就是跨进程通信只支持网络socket和Unix socket。虽然ZeroMQ内部**提供了命名节点的失败重发**机制，但是仍然避免不了**进程崩溃会导致包丢失**的问题。

再一个问题就是即便使用Unix socket，性能还是比共享内存差不少。在写这篇博文前我按照前面第二种通信模式写了一个对**zeromq的压力测试**，具体代码可见这里https://gist.github.com/owent/72c3fd5f4bb63a863641。压测结果在这里
http://api.owent.net/resource/doc/link/zerqmq%20%E6%80%A7%E8%83%BD%E6%B5%8B%E8%AF%95%E6%8A%A5%E5%91%8A.xls

相对与前面共享内存通信而言，***大消息包时性能和前面的接近，小消息包时大约是前面共享内存性能的二分之一到三分之一***。而游戏进程间通信的消息体大多数情况下不大。

不过我个人觉得，最重要的问题还是消息的可靠性问题。

开源BUS组件- D—Bus[4]

D-Bus已经用于Gnome、Qt等一些知名的开源项目。这个组件我没有太深入的研究，一方面是由于其过于复杂了，另一方面虽然他是面向连接的，但是貌似依然**不支持共享内存**，所以估计性能上不会比ZeroMQ好。另外就是它早期被用作本机进程间通信的，跨机情况下不知道稳定性怎么样。

另外D-Bus通过发送不成功时dump消息到文件以下次发送来保证消息可靠性。这种做法可以很容易想到在连接闪断的时候CPU会飙高。

不过话说回来，D-Bus确实时一个完整的Bus通信系统，有完善的监控机制、完整的消息服务、完善的功能（虽然支持的模式不像ZeroMQ一样那么多）。

但是也是因为它太完善了，所以也就太庞大了。导致我不太喜欢。 另外这里有关于D-Bus的性能测试报告，看起来性能不怎么样啊。
http://pvanhoof.be/blog/index.php/2010/05/13/ipc-performance-the-report

其他BUS系统

Redhat搞了个dbus的分支**kdbus**[5]，貌似把这玩意整进linux内核了，据说性能会进一步提升。但是总感觉还要过内核，再加上上面的D-Bus性能测试报告，略微担忧。

另外，还看到个Android上的**Binder**，说是消息也是放在共享内存中，消息传递时是通过切换指针来完成，甚至通道切换都不要进行拷贝。这确实是个比较有意思的做法，但是需要系统提供驱动设备文件支持,并且所有节点共享binder的数据区。这就要求必须由比较复杂的内存管理机制来管理数据节点。而且貌似Android里实现的binder管理的内存也不是很大。

> 接下文... Written with StackEdit.

1. skynet: 云峰设计的基于c和lua的开原游戏服务器框架，https://github.com/cloudwu/skynet ↩
2. zeromq: 一个高效面向消息的消息队列组件，http://www.zeromq.org ↩
3. boost: 一个按照stl规范编写的跨平台高性能C++库，也被称为准STL库，很多C++标准特性都是由这里面提炼而来，http://www.boost.org ↩
4. D-Bus: 一个用于Gnome、Qt的开源Bus系统解决方案，http://www.freedesktop.org/Software/dbus ↩
5. kdbus: 内核态D-Bus接口系统，https://github.com/gregkh/kdbus ↩

Article/ Blablabla *2014-08-01*

bus ipc 无锁队列 消息队列 进程间通信

上一篇**关于BUS通信系统的一些思考（二）** 下一篇**[libiniloader] Project**

知识共享许可协议 2022 OWenT 本站源码, 发布者 Hugo, 主题 distinctionpp