Skynet 的一次大更新

Skynet (关于 skynet 的更多 blog 见右侧导航条上的 skynet tag) 的设计受我在 2006 年做过的一个卡牌游戏服务器影响很重，后来又受到 2008~2011 年期间 Erlang 的影响。多年的经验也让我背上许多思想包袱，以前觉得理所当然的东西，后来没来得及细想就加入了 skynet 里面。

最近项目稳定下来，并且开始了第一个手游项目，虽然带点试验性质，毕竟也是第 2 个我们自己正式使用 skynet 的项目了。做第 2 个服务器项目的晓靖同学对 skynet 提出了不少想法和疑问，让我重新考虑了以前的设计。最近花了一个月时间重写了大量的代码，为下一步重构底层设计做好准备。

在一个稍有历史的活着的项目上做改造是不容易的，时刻需要考虑向前兼容的问题，又不想因为历史包袱而放弃改良的机会，我只能尽力而为了。

最初我认为 skynet 是为分布式运算设计的，没有用 erlang 的主要原因是因为我们有大量的业务逻辑需要（习惯）用命令式语言编写。所以最早的 skynet 版本是基于 erlang 的，并把 lua 嵌入了 erlang 。或许只是我们的实现不太好，总之结果性能表现很糟糕。放弃直接用 erlang 编写业务逻辑，而是透过 C driver 到 lua 中去解析消息请求回应加大了中间层的负担。慢慢的我发现，erlang 带给我们的好处远不如坏处多。

除了性能，我们的代码充满了不同语言不同风格：有用 erlang 实现的，有用 lua 的，还有用 C 的。为了跨语言，又不得不定义了统一的消息格式(使用 google protobuffer)。这些加起来变得厚重很难维护。

最终我决定用 C 重写底层的代码，并重新考虑 skynet 底层模块的设计定位。

其实，对于我们的网游服务器来说，分布式运算，以及分布式运算的稳定性并不是第一位的。我们更需要的是充分挖掘多核处理器的单机性能以达到高实时的相应速度。分布式运算和并行运算显然不是同一类问题。skynet 只要专注于并行处理就可以了，顺带支持一下分布式可以说是为以前的设计做的兼容。

事实上，即使在过去基于 erlang 的版本，我们依然编写了两个独立的系统间通讯的模块，而不全部依赖于 erlang 自己的分布式设计。这个模块一直沿袭到现在依然存在于我们的项目中，让 skynet 自己的分布式节点管理显得有点多余。

skynet 专注点应该是一个单进程内的任务调度器，以及消息分发器。虽然理论上 skynet 可以嵌入各种不同的虚拟机，比如更多人爱用的 python 。但限于它目前的用户量太小，主要开发工作也是我一个人在做，我只能按我自己的喜好单维护 Lua 的版本。经过一年多的开发，我发现即使为了性能考虑，我也很少再自己直接用 C 去开发 skynet 的服务了。我可以写一个 Lua 的 C 模块挂在一个 Lua 服务上嵌入 skynet 中使用。Lua 就成了 skynet 事实上的标配。

Lua VM 所占用的额外空间小，以及启动新的 Lua VM 速度快就成了 Lua 最大的优势。虽然它在共享只读数据方面还比不上 Erlang ，但命令式语言这点很受我们的开发人员欢迎。Lua 的 coroutine 有很小的内存开销，对于 Lua 5.2 来说，不到 300 字节，这比 Golang 的 goroutine 还要小的多，而功能上甚至更超一筹。

很多同学问我，为什么 skynet 默认的 RPC 机制没有超时处理。我的回答是，如果你非要做超时，可以用现有机制模拟出来。现在的 skynet lua api 可以 sleep ，可以 wakeup 一个 sleep 的 coroutine ，可以 fork 一个 coroutine ，这些加起来足够实现一个带超时机制的 RPC 调用来。但我选择不直接提供，因为，如果每次 RPC 调用都考虑超时失败的情况的话，其带来的复杂度远远超过了 RPC 带来的便捷。

如果我们把 skynet 的一个进程看成一体的，那么它和以前传统的单进程服务器没有两样。进程内部的 RPC 调用其实不是 RPC 调用，并没有出进程嘛。使用 Lua 已经可以利用 Lua VM 这个沙盒防御大部分逻辑错误，让服务间的调用产生异常时请求者可以知晓。如果一端不返回，那么一定是代码写错了。这跟调用一个函数死循环，或是多线程程序死锁并没有区别，我们需要的是 debug 而不是用超时来防御。为了方便 debug ， skynet 已经提供了许多性能剖析的模块，有点已经放在开源版本里，有的还不太成熟，只是自己项目在用，等有机会整理后也会开源。

另一个常见的问题是，为什么 skynet 的调度模块使用的是一个简单粗暴的处理方法。就是根据配置在一开始启动了固定数量的系统线程，组成一个工作线程池。它们一旦开工就老死不相往来。工作线程之间是没有任何消息通讯和状态同步的。它们唯一做的事情就是不停的从活跃的服务集中取一个出来，读取属于这个服务的消息队列中的第一个消息，处理它，然后（若消息队列不为空）把这个服务放回服务集中去。如果系统中没有需要处理的服务，它就简单的休眠 0.1

秒。

最后这个若无可处理之服务，就休眠 0.1 秒，被很多同学诟病。但是，除了保持简单这个理由外，它跟网游服务器的特点强相关。MMORPG 通常是保持数千个长连接，为几千个用户持续服务几个甚至几十个小时。这些连接上的数据频率并不高，一般一秒就 2K~20K 的数据。但 MMORPG 的内部逻辑非常复杂，对 CPU 要求很高。每个连接上推送来一个数据包，往往需要变成几十个内部请求在服务器内部处理。处理流水线也经常会超过 5,6 个环节。

如果服务器处理不过来，就会反过来限制外部连接上的数据。试想，如果玩家都看不见周围的人，他如何发大招去攻击他们呢？

所以，对于一个 MMORPG 服务器，大部分工作线程都处于热状态是常态。一旦负荷降下来，及时对单个玩家偶然发生 100ms 的延迟，也绝对比高负载下他的延迟要低，所以对用户体验来说，这是没有问题的。skynet 要做的是，不要让 cpu 空转浪费掉处理能力就行了。

不过，最近我还是对上面的设计做了一点修改，这来源于 skynet 的设计的一些变更，具体下面会展开。只是修改结果并不算特别理想，虽然可以提高在低负载下的响应速度，但高负载下的承载能力反而下降了一点。

最开始，我只想让 skynet 做好消息分发和任务调度的事情就够了。消息只是用来沟通任务进程的手段。一切都和系统关系不大。专注于单进程内的任务协作，我们可以做的很高效。一个服务把请求数据组织好，直接就可以把数据指针传递给流水线的下一个环节。中间可以省略掉数据拷贝的开销，流水线上的每个环节都可以直接对数据流做操作，只要能保证同时只有一个服务在处理就够了。这是比多进程模型而言最大的优势。

后来我发现，如果让 skynet 不仅仅做好 MMORPG 服务器做一类工作，比如扩展到 web 开发领域的话，外源消息，也就是从系统 socket 来过来的消息的比重会增加。也就是说，如果我不在底层把外源消息和内部消息做区分的话，我很难让外源消息也有同样的反应速度。

内部消息是从内部处理流程中发出的，所以一旦产生，系统内一定至少有一个工作线程是活跃的（不然这条消息是如何产生的？),所以这条消息一定会被即时处理。

而外部消息的存在是 skynet 底层所不知道的。过去我用了一个叫 gate 的服务来处理外部连入的连接。它用启动一个 epoll （在 mac/bsd 上使用 kqueue）循环检查外部 socket 上的数据，并分布给内部服务。无论怎样实现都逃不了一个问题，gate 本身还需要处理 skynet 内部的请求，所以它不可能用 epoll 不断的死等，有消息过来就立即分发。对于高数据，长连接的环境，这不是问题。epoll 上永远有新的数据过来，gate 也不会休息。但在低负载环境下，一旦外部连接引发的一系列处理流程做完后，新的外部请求还没有发起的话，整个 skynet 都会陷入最长 0.1s 的休眠状态。（由于多线程的存在，往往是交替 sleep 的，所以实际休眠时间比 0.1s 要短）

这看似不是特别严重的问题，影响的仅仅是某些特定低负载环境下的性能测试跑分而已。只是感觉上有点怪，负载高，反而系统响应速度变快了。光是这个问题，简单粗暴的解决方法是减少系统空闲时的休眠时间，比如从 0.1 秒调整到 0.01 秒甚至更短就好了。所以单就这个问题是不会让我下决定重写几千行代码的。

促使我做大改动的原因是，目前处理外部连接数据的代码已经散布在 skynet 各处，当初我认为这并不重要，每个服务自己写好就够了，skynet 只关注内部消息分发，这个命题到底有没有问题？我最近思考的结果是，只要是一个在持续工作的系统，就一定持续的外部输入。skynet 要解决的问题并不是一开始准备好所有的输入，运算出结果，输出退出。所以我不应该在最底层回避持续外部输入这个问题。

而且操作 socket 的代码和系统强相关，不适合每个服务单独编写。所以之前我已经把 socket 处理收敛到 gate 以及 socket 两个服务中去了，但运行时用户依然会开启多份 gate 实例，在系统中跑多个 epoll 循环，对总体性能是有影响的。面对 CPU 高负载的系统，我们应该尽量减少系统调用，把 CPU 时间的充分利用放在用户态完成。这促使我重新编写 skynet 的 socket 相关代码，把它们从中间层移到底层去。

这个 socket 处理模块可以塞死在一个 epoll 循环上，一有外部消息就构造一个数据结构，并把其指针直接放到 skynet 的内部消息队列中。作为一个底层模块，它可以实现的更高效。对 skynet 动大手术前，我先实现了一个独立的 socket-server 模块。然后整合入 skynet 。

最终的结果还不错，新的 socket 库可以自己 listen/accept 外部连接了，这使得原有的 gate 显得多余。因为 socket 库更加灵活，不用固定分包协议，也减少了一个中间环节而更高效。我用新的 socket 库按以前的协议重写了 gate 。这个版本的 gate 仅仅是做在数据流上分包的处理，不直接操作系统 socker。它仅用于向前兼容，之后不会再推荐用它了。

最后我做了一点简单的性能测试，用 lua 基于新的 lua 版 socket 库编写了一个符合 Redis 的 ping 协议的服务器。就是接收到一个以 PING\r\n 结束的数据包时，回应一条 +PONG\r\n 。

使用 redis-benchmark -t ping -n 100000 -c 10 做了个简单的测试。

可想而知，使用 C 语言编写的 Redis 服务器一定可以在这个测试中得到最高分，因为这个协议处理非常简单，完全是考虑 IO 处理的能力。在我一台旧机器上，redis 可以跑到 40k qps 。

skynet 的测试程序不算差，达到了 33k qps 的成绩。我认为和 redis 的差距在于，我写的 lua socket 库需要把数据压入 lua vm ，这需要多做一次内部复制。且 lua 语言本身也不如 C 语言高效。换成 luajit 后，skynet 的成绩也可以达到 40k qps 了。

让我有一点小郁闷的是，如果我简单去掉新写的代码中 socket-server 模块中对 skynet worker 线程唤醒的调用，那么性能可以直接从 33k 上升到 37k qps 。反复核查的结果是，大量的 pthread_cond_signal 调用消耗掉了许多 CPU 时间。即使我反复优化，减少不必要的 signal 操作也弥补不回这点性能损失。我想，这就是提升复杂度带来的性能支出吧 —— 原本 worker 线程是不需要和外部同步任何状态的，现在可以接收外部信号而增加了沟通成本。

公司的几个同学纷纷做了别的框架下的性能测试。他们用的测试机器性能要好一些。

在另一台较好的机器上，redis 跑出了 180k qps 的高分。skynet 则可以达到 120k qps ， jit 版提升不太明显，大约可以增加 10k qps 。erlang 的版本也可以跑到 180k qps 左右，不过 erlang 版写的比较简陋，协议实现的不完全对，对 ping 协议的分包做的不完整，估计做对了以后会略微有一点损失。

基于 python 的 gevent 这方面性能表现比较差，只有 30kqps 左右。可能跟 python 性能比较低有关。如果换成 pypy 的话，可以提升到 60kqps ， 这远比 luajit 对 lua 的提升幅度高的多。

后面我的工作计划：

一 是把 skynet 的多机支持从底层拿掉，改到中间层去支持。这样可以简化掉底层代码。实际上我们自己的 MMORPG 项目在开发中，发现 skynet 底层的分布式支持做的远远不够，本身就需要在上面堆砌更多的代码才能做好的。比如一个玩家的 agent 服务，必须和他所属的 map 服务在同一个进程中才可以获得最快的反应速度。这让我必须支持 agent 从一个进程迁移到新的进程，而不可能维持在 skynet 中的唯一地址。

我们在开发中总是要计较一个关联服务是在同一个进程内，还是在别处。企图透明化服务的物理位置是不现实的。所以还不如就放在上层去支持。

二 是系统从底层中去掉为服务命名的支持。因为查询名字有一定的开销，大部分人在用的时候都是先把名字查好，以后在直接用数字地址发送消息。底层做的为名字服务缓存消息的功能不算特别完备，所以也没有人去用。同步全局名字这件事情放在底层做也是吃力不讨好的。

三 是加强服务的生命期管理。目前只对服务消失后做了简单的消息通知，还没有把这个特性利用起来。

四 是继续补完 socket 库的计划中功能。把 socket 的生命期和具体服务绑定在一起。可以在服务消失后自动关闭他所有的连接。这个设计已经做了，但是还没有实现。另外转移 socket 的所有权也是需要做的事情。之前 gate 的 forward 功能太粗糙，很难在发出转移指令后，但转移成功之前这段时间把事情做干净。

以上是一点流水账式的记录。鉴于这次改动的几千行代码，就不马上把改动push 到 github 上了。等我在公司内部项目中跑几天，观察一下是否有明显的 bug 再放出。正在使用 skynet 的同学们也请留意这次更新，请善待 bug 和它们的小伙伴们 :)

这个月底，我应邀参加在北京举行的软件开发大会。会和大家分享一下 skynet 的设计。好借这个机会推广我们这个开源项目。只有更多人用它，才会发展的更好。ppt 我已经写好了，放在这里供下载。

注：新写的 gate 转发 client 的发送包协议有所改动，把要转发的包的目的 id 放在了包尾（以前是包头），需要修改对应的 client 模块。不过新的设计可以直接利用 skynet 的 socket 库发送数据，所以 gate 的转发功能仅仅是兼容而已。gate 的其它协议没有变化。

云风 提交于 August 24, 2013 11:24 AM | 固定链接

COMMENTS

@agentzh 也在关注游戏领域啊，哈哈

Posted by: lee | (14) December 31, 2013 03:04 PM

mmorpg逻辑服务器真不必过多考虑分布式，主要着重并行，充分利用多核提高单机性能就好了，分布式功能由目录服务器完成。

Posted by: chenbk | (13) November 16, 2013 12:48 AM

后期是否考虑会有其他元素加入呢

Posted by: 如何管理好团队？ | (12) October 24, 2013 10:42 AM

学习到了，大更新！后期是否会考虑别的呢？

Posted by: 如何管理好团队？ | (11) October 24, 2013 10:41 AM

后期是否考虑支持服务优先级呢?

Posted by: lpk | (10) September 18, 2013 09:51 AM

@agentzh

我用 jit.v.start() 看了一下, 当请求数量超 100 以后, TRACE 信息就出来了.

类似这样的:

[TRACE --- socket.lua:200 -- NYI: C function 0x412d0c98 at socket.lua:204]
[TRACE --- skynet.lua:317 -- NYI: FastFunc next at skynet.lua:318]
[TRACE --- skynet.lua:41 -- NYI: FastFunc coroutine.yield at skynet.lua:44]
[TRACE --- skynet.lua:57 -- NYI: FastFunc next at skynet.lua:58]
[TRACE --- skynet.lua:69 -- NYI: C function 0x410b7be8 at skynet.lua:70]
[TRACE 1 skynet.lua:257 return]
[TRACE --- redisping.lua:13 -- NYI: C function 0x412d0c98 at socket.lua:204]
[TRACE --- socket.lua:37 -- NYI: C function 0x410c6280 at socket.lua:45]
[TRACE --- socket.lua:200 -- NYI: C function 0x412d0c98 at socket.lua:204]
[TRACE --- skynet.lua:57 -- NYI: FastFunc next at skynet.lua:58]
[TRACE --- skynet.lua:317 -- NYI: FastFunc next at skynet.lua:318]
[TRACE --- skynet.lua:41 -- NYI: FastFunc coroutine.yield at skynet.lua:44]
[TRACE --- skynet.lua:69 -- NYI: C function 0x410b7be8 at skynet.lua:70]
[TRACE 2 socket.lua:21 return]
[TRACE --- redisping.lua:13 -- NYI: C function 0x412d0c98 at socket.lua:204]

我觉得 luajit 提升不多是因为 lua 在这个简单测试里做的工作并不算多. 比 redis 原生实现多出的拼接和复制字符串的部分, 又是在 C 库里实现的, 无法被优化.

Posted by: Cloud | (9) August 28, 2013 11:17 AM

另外，我觉得如果分别使用 on-CPU 和 off-CPU 火焰图工具对你这里参与性能测试的各种不同的服务进程逐一进行白盒的分析，并进行热代码路径的比较，会比简单的黑盒评测有趣和有意义的多 :)

见 https://github.com/agentzh/nginx-systemtap-toolkit#ngx-sample-bt 和 https://github.com/agentzh/nginx-systemtap-toolkit#ngx-sample-bt-off-cpu

Posted by: agentzh | (8) August 28, 2013 03:35 AM

你确认你的 Lua 代码运行在 LuaJIT 中时，其主要 Lua 代码路径确实被 JIT 编译了吗？（你可以很方便地通过 LuaJIT 自带的 jit.v 或者 jit.dump 模块对此进行验证。）

我担心的是，或许你测试的只是 LuaJIT 的解释器性能 :)

Posted by: agentzh | (7) August 28, 2013 03:31 AM

用tcmalloc编译重新测试下效率

Posted by: wstc | (6) August 26, 2013 02:43 PM

风哥，代码中多加点注释哦，让这个开源项目更加地壮大哈：）

Posted by: lovebird | (5) August 25, 2013 06:55 PM

谢谢skynet，谢谢云风，获益良多
sharedb的设计，很有意思，我觉得是基础设施里面很重要的一环，希望能更详细些，能开源一部分也好

Posted by: wesom | (4) August 24, 2013 02:24 PM

期待新的版本哦哦哦

Posted by: playboy | (3) August 24, 2013 01:28 PM

hello.你好，你正在做的工作和我的产品有很大的共同点。不过我们的产品基本已经解决你提到各种困扰。当然还有很大的空间可以提升我们的产品品质。有空可以私下交流。jimshao@163.com

Posted by: Anonymous | (2) August 24, 2013 12:47 PM

下載了PPT 感謝雲風兄的分享

Posted by: DaVinci | (1) August 24, 2013 12:20 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