« 随机地形生成 | 返回首页 | GDC China 2014 »

skynet 服务的过载保护

最近我们的新游戏《天天来战》上了腾讯平台，由于瞬间用户量过大，发现了几个 bug。

这几个 bug 都是在最后一周赶进度时编写业务的同学写的太仓促，在一些处理请求的流水线上使用了时间复杂度 O(n) 以上的算法导致的问题。这些时间开销大的操作，虽然并不常见，但操作误放在了和用户登录相关的服务中，导致一旦阻塞，使得用户登录受到影响。

具体 bug 没什么好谈的，把业务拆分开，以及用 O(Log N) 或 O(1) 的算法重新实现后就好了。但发生 bug 后，skynet 的整体表现值得一提。

按原有的设计，skynet 可以视为一个简单的操作系统。每个服务都是这个系统中的进程。不相关的业务应该互不干扰（使用多核的硬件，核心越多，就可以表现的越好）。在这次事件中，的确也做到了受影响的部分（登录）处理能力不足，用户无法正常操作时；另一些做无关操作（副本游戏）的用户没有收到影响。但在服务过载后的恢复环节却做的不够。

由于 bug 的影响，有类消息的处理能力只有 20 次/s 左右。当需要处理的消息频率超过 20 次后（在线玩家超过 8000 人以后出现），该服务过载，导致整个系统处于半瘫痪状态。新用户无法正常进入，直到在线人数下降到 4000 人都没有好转。但已在玩游戏的用户没有受到影响，所以没有做任何处理。大约在 2 个多小时后，系统自己维护正常。这比预期时间要长得多。

这是上周遇到的问题。昨天又在新一轮导量中出现了类似的问题。由于配置问题把大量玩家（十万数量级）引到同一组服务器，导致该服务器几乎无法创建新角色，同时老玩家登录也无法获取自己的角色（因为和创建角色在同一服务内）。如果玩家有足够耐心，等待 10 分钟，还是可以正常进入游戏。这个状态在分流新用户后，得到了缓解。但服务器依然用了小时级的时间才自我恢复。经事后排查，同样是一处 bug 导致的性能问题，但自我恢复时间过长也值得关注。

在 skynet 框架的基础上做设计，需要积累经验。应该尽量去掉单一的热点。如果很多业务流程都经过同一个服务，那么这个服务的处理能力很容易就约束整个系统。

如果这种单点服务无法拆分，那么除了尽可能的优化它（以提高整体处理能力）之外，对其做过载保护也是很有必要的。

在上面的实例中，过载造成的雪崩是这样发生的：

有一个 skynet 内部服务 D ，它负责处理大量的数据。数据源来至于玩家客户端代理 agent 。每个玩家都在 skynet 系统中有一个 agent 服务做为状态维持的代理，而 D 并不维持状态，它仅处理 agent 发给它的请求。

注：这是一个明显的热点，D 的处理能力很容易影响整体的处理能力。我们应该对 D 做拆分，比如按业务拆分开，或按玩家 ID 拆分。但这不是本篇谈论的重点。

D 的处理能力有限，当请求过多时，每个请求都需要较长时间才能得到回应。这就是所谓的过载。过载发生一旦被感知，维护人员可以做的最快的应对策略是把用户分流，但为什么用户减少后，系统没有立刻恢复过来呢？

这是因为，过载发生的那段时间内，每个用户都在等待，而大部分用户又没有足够的耐心，他们选择了断开连接，重新登录再试。而 D 无法感知外部连接情况，它平等对待没有请求。结果在同一个用户的不断尝试中，待处理队列中堆积了大量的无效业务。直到业务处理完成要把结果发回时才发现回应方已经不需要结果了。

这种情况一旦发生，有效处理能力进一步的减少，加剧了过载。即使屏蔽了新用户进入，依然在很长时间都得不到缓解。

我们可以用一个形象的比喻来描述这个问题。

有一家餐厅很大，可以坐很多桌人。这是当地唯一的一家餐厅，所有的饥肠辘辘的吃客都会拥进来坐满餐厅。

但餐厅的厨师有限，在客人很多的时候需要更多的时间做菜。每桌坐下来的客人都会把点单送入厨房，然后等待。尤其

是客人点一些非常耗时的菜时，所有客人都等待更长时间。

有些没有耐心的客人会忍不住离开，但是厨师并不知道，他无暇分清做的菜是哪位客人的，只是按订单一盘盘做。等菜做好后为时已晚，客人已经走了，只好倒掉。

而这些离开的食客往往又耐不住饥肠辘辘返来，可按照规则，他们必须坐到新桌上重新点餐。缺乏有效的沟通，即使之前的点单还没开始做也不能取消。只好眼睁睁看着之前点的菜被端上来又倒掉，还得继续空等他们后来下的单。

这样，餐厅的接待能力进一步下降，使得平均等待时间更长，超过了更多的客人的耐心。最后，即使所有的客人最终都离开不再回来，厨房里依然需要忙碌很久，去做那些没有人需要的订单。新来的客人也必须等待很久才能进餐。

这种管理方案显得如此愚蠢，没有餐厅会这么干。但在我们的系统中，的确很容易犯这个错误。skynet 的服务只有消息接收队列，没有消息发送队列。无论对方接收队列多长，请求总是无条件的投递过去。即使发送者退出，也无法撤回未完成的请求。处理消息的一方，在处理消息时，做到感知请求方是否存在再决定是否真正处理消息会增加很大的设计复杂度，同时也可能影响性能。

我建议用一个简单的模式来解决这种过载之后的性能进一步恶化的问题：

对于这种热点服务，对于所有请求（ 也只可以针对不能在 O(1) 时间处理的请求 ）都在回应协议中增加一个 boolean ，表示服务繁忙。请求方必须处理这个繁忙标记。

服务提供方在收到每个请求时，先不急于处理，而是通过 skynet.mqlen 检查当前的消息队列长度，来判定是否繁忙。（一个小优化是，可以不用每个消息都取 mqlen 判定繁忙，而让繁忙标记在设置后维持连续几个消息。）如果繁忙，则直接扔掉请求，直接返回忙，直到消息队列被处理的差不多再做进一步处理。

而发送请求方则把 skynet.call 的调用改成一个循环。检查到忙，再接着重试，直到正确处理为止。做一个简单的封装后，可以看成和之前是一样的。

用前面那个餐厅的例子，现在变成了这样：

当空桌过多时，直接赶走新来的客人。一旦客人变少，就继续按原有的方案把客人的点餐单送入厨房。一部分饥肠辘辘的客人被赶走后会回来继续排队（这相当于那个循环尝试，重新发送请求需要一点时间）。另一部分没有耐心的客人选择暂时离开，等他饿坏了再来（相当于断开重连），也重新排到队尾。但和之前的方案不同的是，重新重新排队的人之前没有向厨房提交过订单。

当然，这个方案仅仅减少了浪费，让情况不至于恶化。多雇几个厨师，以及改进菜品的制作时间必须双管齐下。

对于没有 skynet 的开发经验的同学，第一次犯错简直是必然的。所以我最近的工作都是在加强 skynet 对于过载的预警。在目前的 dev 分支上，任何一个服务的消息接收队列过长（超过 1024）都会记录一条 log ；但并非所有过载都是体现在内部消息队列上的。比如一个对外部数据库对接的服务，过载会体现在外部数据库连接的发送队列过长 ，同时也反映在服务未完成的请求增加。所以，当一个服务的 coroutine 池的大小持续增加（每增加 1024 个）时，也会自动记录 log 。

可以利用 skynet 内部的 profile 统计单条请求的处理时间也非常重要。找到热点服务中的耗时请求，重点优化。

云风 提交于 October 9, 2014 03:28 PM | 固定链接

COMMENTS

天天来战用u3d做的，云大大感觉性能怎么样，什么样的配置机器能流畅运行？

Posted by: huige | (21) November 13, 2014 06:35 PM

我想这只是这种基于消息的机制表现的问题之一。这种设计应该是为了以通用，不过我的体会是通用往往意味着复杂化，反而增加开发维护难度。我觉得服务器框架可以很简单，而且应该很简单，我现在用的整个框架150k,因为简单，所以才能在2个半月用这个框架从头完成一个可以签约的游戏服务器。

Posted by: yanwu | (20) October 29, 2014 07:08 PM

过来看看！

Posted by: LED广告车 | (19) October 23, 2014 02:33 PM

编译SkyNET通过了，但测试lua client.lua 有错误：
lua: error loading module 'clientsocket' from file 'luaclib/clientsocket.so':
luaclib/clientsocket.so: undefined symbol: luaL_setfuncs
stack traceback:
[C]: ?
[C]: in function 'require'
examples/client.lua:4: in main chunk
[C]: ?

Posted by: ldlan | (18) October 21, 2014 11:39 AM

编译SkyNET通过了，但测试lua client.lua 有错误：
lua: error loading module 'clientsocket' from file 'luaclib/clientsocket.so':
luaclib/clientsocket.so: undefined symbol: luaL_setfuncs
stack traceback:
[C]: ?
[C]: in function 'require'
examples/client.lua:4: in main chunk
[C]: ?

Posted by: ldlan | (17) October 21, 2014 11:38 AM

“有些没有耐心的客人会忍不住离开，但是厨师并不知道，他无暇分清做的菜是哪位客人的，只是按订单一盘盘做。等菜做好后为时已晚，客人已经走了，只好倒掉。”

这里一开始比喻就有些不明白，响应要求如果是实时的，应该有更多的厨子或者应该专门分配一部分厨子给这种实时性要求高的；而介于实时性高和低的 应该给消息处理队列标记，确认插队 或者直接抛弃处理。

Posted by: xiaoyulong | (16) October 21, 2014 10:58 AM

不错, 过载经常会出现在登录和初始化的时候.

Posted by: cat | (15) October 18, 2014 09:18 PM

登陆太频繁，需等待10分钟后再尝试。

Posted by: 登陆太频繁，需等待10分钟后再尝试。 | (14) October 17, 2014 03:15 PM

其实对于这种结果，就是其skenet的架构导致这个问题的产生，必然的！

Posted by: 成林 | (13) October 17, 2014 11:36 AM

云风大哥，我刚才下载看了下，在我的手机（分辨率800*480）上图像有一点显示不全。貌似只能用QQ号或者微信号登陆吗？好想玩玩看，但是我没有QQ和微信。。。

Posted by: Anonymous | (12) October 16, 2014 09:07 PM

您采用的处理中，遇到这种过载临界情况怎么办，情况描述如下：
消息量始终徘徊在过载保护阀值的限制附近，无法继续减少，因为Client还会定时发送请求（无论长链还是短链）

Posted by: WL | (11) October 15, 2014 10:14 AM

刚玩了一下，确实做的不错

Posted by: David Xu | (10) October 14, 2014 02:20 PM

惊呆了！天天来战原来是前辈做的啊！十分有意思和有新意的一个游戏，对风云前辈更加仰慕了。

Posted by: sam | (9) October 12, 2014 03:24 PM

比喻得很形象

Posted by: lichking | (8) October 11, 2014 02:37 PM

可以考虑一些类EMA之类的算法可以做一些快速拒绝 最主要的是系统本身要具备完整的数据和性能上报监控体系 恢复起来就很快

Posted by: Anonymous | (7) October 10, 2014 07:02 PM

云风你这个方案无非是把超载从服务提供方转移到了发送方而已,发送方在循环中不能处理任何事情.服务提供方建立双队列吧,其中一个队列是高优先级的,服务发送方在离开时,需要向服务提供方高优先级队列发送一个离开通知,服务提供方在检查到这个消息后,自动从另一个队列中剔除所有相关消息就行了.

Posted by: bull | (6) October 10, 2014 12:09 PM

请求方遇到服务方忙就在服务方挂一个事件，服务方在不忙时主动去通知请求方不忙了，然后请求方重新请求。

Posted by: fanfeilong | (5) October 9, 2014 11:35 PM

做一个引擎的复杂程度是非常高的。

Posted by: 唐吉 | (4) October 9, 2014 10:06 PM

Shameless plug:
WinCmder - Grouping Desktop Icons
http://www.wincmder.com

Posted by: wincmder | (3) October 9, 2014 05:19 PM

比喻很形象

Posted by: huanzh | (2) October 9, 2014 04:19 PM

“处理消息的一方，在处理消息时，做到感知请求方是否存在再决定是否真正处理消息会增加很大的设计复杂度，同时也可能影响性能。”
我们的解决办法是：客户端断开链接之后，服务器监测到对应的socket长连接断掉，就另起一个线程，去扫描待处理消息队列，把对应该socket长连接的消息(们)都修改有效标示为false（不删除） 这样当正常的处理消息线程处理它(们)的时候直接跳过

Posted by: wk3368 | (1) October 9, 2014 03:53 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