《多线程服务器的适用场合》例释与答疑

陈硕 (giantchen_AT_gmail)

Blog.csdn.net/Solstice

2010 March 3 - rev 01

《多线程服务器的适用场合》（以下简称《适用场合》）一文在博客登出之后，有热心读者提出质疑，我自己也觉得原文没有把道理说通说透，这篇文章试图用一些实例来解答读者的疑问。我本来打算修改原文，但是考虑到已经读过的读者不一定会注意到文章的变动，干脆另写一篇。为方便阅读，本文以问答体呈现。这篇文章可能会反复修改扩充，请注意上面的版本号。

本文所说的"多线程服务器"的定义与前文一样，同时参见《多线程服务器的常用编程模型》（以下简称《常用模型》）一文的详细界定，以下"连接、端口"均指 TCP 协议。

1. Linux 能同时启动多少个线程？

对于 32-bit Linux，一个进程的地址空间是 4G，其中用户态能访问 3G 左右，而一个线程的默认栈 (stack) 大小是 10M，心算可知，一个进程大约最多能同时启动 300 个线程。如果不改线程的调用栈大小的话，300 左右是上限，因为程序的其他部分（数据段、代码段、堆、动态库、等等）同样要占用内存（地址空间）。

对于 64-bit 系统，线程数目可大大增加，具体数字我没有测试，因为我实际用不到那么多线程。

以下的关于线程数目的讨论以 32-bit Linux 为例。

2. 多线程能提高并发度吗？

如果指的是"并发连接数"，不能。

由问题 1 可知，假如单纯采用 thread per connection 的模型，那么并发连接数最多 300，这远远低于基于事件的单线程程序所能轻松达到的并发连接数（几千上万，甚至几万）。所谓"基于事件"，指的是用 IO multiplexing event loop 的编程模型，又称 Reactor 模式，在《常用模型》一文中已有介绍。

那么采用《常用模型》一文中推荐的 event loop per thread 呢？至少不逊于单线程程序。

小结：thread per connection 不适合高并发场合，其 scalability 不佳。event loop per thread 的并发度不比单线程程序差。

3. 多线程能提高吞吐量吗？

对于计算密集型服务，不能。

假设有一个耗时的计算服务，用单线程算需要 0.8s。在一台 8 核的机器上，我们可以启动 8 个线程一起对外服务（如果内存够用，启动 8 个进程也一样）。这样完成单个计算仍然要 0.8s，但是由于这些进程的计算可以同时进行，理想情况下吞吐量可以从单线程的 1.25cps （calc per second） 上升到 10cps。（实际情况可能要打个八折——如果不是打对折的话。）

假如改用并行算法，用 8 个核一起算，理论上如果完全并行，加速比高达 8，那么计算时间是 0.1s，吞吐量还是 10cps，但是首次请求的响应时间却降低了很多。实际上根据 Amdahl's law，即便算法的并行度高达 95%，8 核的加速比也只有 6，计算时间为 0.133s，这样会造成吞吐量下降为 7.5cps。不过以此为代价，换得响应时间的提升，在有些应用场合也是值得的。

这也回答了问题 4。

如果用 thread per request 的模型，每个客户请求用一个线程去处理，那么当并发请求数大于某个临界值 T′ 时，吞吐量反而会下降，因为线程多了以后上下文切换的开销也随之增加（分析与数据请见《A Design Framework for Highly Concurrent Systems》 by Matt Welsh et al.）。thread per request 是最简单的使用线程的方式，编程最容易，简单地把多线程程序当成一堆串行程序，用同步的方式顺序编程，比如 Java Servlet 中，一次页面请求由一个函数 HttpServlet#service(HttpServletRequest req, HttpServletResponse resp) 同步地完成。

为了在并发请求数很高时也能保持稳定的吞吐量，我们可以用线程池，线程池的大小应该满足"阻抗匹配原则"，见问题 7。

线程池也不是万能的，如果响应一次请求需要做比较多的计算（比如计算的时间占整个 response time 的 1/5 强），那么用线程池是合理的，能简化编程。如果一次请求响应中，thread 主要是在等待 IO，那么为了进一步提高吞吐，往往要用其它编程模型，比如 Proactor，见问题 8。

4. 多线程能降低响应时间吗？

如果设计合理，充分利用多核资源的话，可以。在突发 (burst) 请求时效果尤为明显。

例1: 多线程处理输入。

以 memcached 服务端为例。memcached 一次请求响应大概可以分为 3 步：

1. 读取并解析客户端输入
2. 操作 hashtable
3. 返回客户端

在单线程模式下，这 3 步是串行执行的。在启用多线程模式时，它会启用多个输入线程（默认是 4 个），并在建立连接时

按 round-robin 法把新连接分派给其中一个输入线程，这正好是我说的 event loop per thread 模型。这样一来，第 1 步的操作就能多线程并行，在多核机器上提高多用户的响应速度。第 2 步用了全局锁，还是单线程的，这可算是一个值得继续改进的地方。

比如，有两个用户同时发出了请求，这两个用户的连接正好分配在两个 IO 线程上，那么两个请求的第 1 步操作可以在两个线程上并行执行，然后汇总到第 2 步串行执行，这样总的响应时间比完全串行执行要短一些（在"读取并解析"所占的比重较大的时候，效果更为明显）。请继续看下面这个例子。

例2: 多线程分担负载。

假设我们要做一个求解 Sudoku 的服务（见《谈谈数独》），这个服务程序在 9981 端口接受请求，输入为一行 81 个数字（待填数字用 0 表示），输出为填好之后的 81 个数字 (1 ~ 9)，如果无解，输出 "NO\r\n"。

由于输入格式很简单，用单个线程做 IO 就行了。先假设每次求解的计算用时 10ms，用前面的方法计算，单线程程序能达到的吞吐量上限为 100req/s，在 8 核机器上，如果用线程池来做计算，能达到的吞吐量上限为 800req/s。下面我们看看多线程如何降低响应时间。

假设 1 个用户在极短的时间内发出了 10 个请求，如果用单线程"来一个处理一个"的模型，这些 reqs 会排在队列里依次处理（这个队列是操作系统的 TCP 缓冲区，不是程序里自己的任务队列）。在不考虑网络延迟的情况下，第 1 个请求的响应时间是 10ms；第 2 个请求要等第 1 个算完了才能获得 CPU 资源，它等了 10ms，算了 10ms，响应时间是 20ms；依次类推，第 10 个请求的响应时间为 100ms；10个请求的平均响应时间为 55ms。

如果 Sudoku 服务在每个请求到达时开始计时，会发现每个请求都是 10ms 响应时间，而从用户的观点，10 个请求的平均响应时间为 55ms，请读者想想为什么会有这个差异。

下面改用多线程：1 个 IO 线程，8 个计算线程（线程池）。二者之间用 BlockingQueue 沟通。同样是 10 个并发请求，第 1 个请求被分配到计算线程1，第 2 个请求被分配到计算线程 2，以此类推，直到第 8 个请求被第 8 个计算线程承担。第 9 和第 10 号请求会等在 BlockingQueue 里，直到有计算线程回到空闲状态才能被处理。（请注意，这里的分配实际上是由操作系统来做，操作系统会从处于 Waiting 状态的线程里挑一个，不一定是 round-robin 的。）

这样一来，前 8 个请求的响应时间差不多都是 10ms，后 2 个请求属于第二批，其响应时间大约会是 20ms，总的平均响应时间是 12ms。可以看出比单线程快了不少。

由于每道 Sudoku 题目的难度不一，对于简单的题目，可能 1ms 就能算出来，复杂的题目最多用 10ms。那么线程池方案的优势就更明显，它能有效地降低"简单任务被复杂任务压住"的出现概率。

以上举的都是计算密集的例子，即线程在响应一次请求时不会等待 IO，下面谈谈更复杂的情况。

5. 多线程程序如何让 IO 和"计算"相互重叠，降低 latency?

基本思路是，把 IO 操作（通常是写操作）通过 BlockingQueue 交给别的线程去做，自己不必等待。

2010年5月 (1)
2010年4月 (1)
2010年3月 (2)
2010年2月 (4)

最新评论

1. Re:C++ 工程实践(8)：值语义
页面字体与背景很舒服
--设计与艺术
2. Re:从《C++ Primer 第四版》入手学习 C++
陈大师没有继续在博客园上更新博客了啊。
--IclodQ
3. Re:多线程服务器的常用编程模型
厉害了
--素笔描青眉
4. Re:谈一谈网络编程学习经验（06-08更新）
楼主已经达到独孤求败的境界了。
--孤火
5. Re:Muduo 网络编程示例之八：用 Timing wheel 踢掉空闲连接
有点意思，不错不错~
--oyld

阅读排行榜

1. 一种自动反射消息类型的 Google Protobuf 网络传输方案(35437)
2. 关于 TCP 并发连接的几个思考题与试验(26905)
3. C++ 工程实践(7)：iostream 的用途与局限(19489)
4. 多线程服务器的常用编程模型(18736)
5. 分布式系统的工程化开发方法(18675)

评论排行榜

1. 计算机图书赠送(40)
2. 从《C++ Primer 第四版》入手学习 C++(22)

例1: logging

在多线程服务器程序中，日志 (logging) 至关重要，本例仅考虑写 log file 的情况，不考虑 log server。

在一次请求响应中，可能要写多条日志消息，而如果用同步的方式写文件（fprintf 或 fwrite），多半会降低性能，因为：

- 文件操作一般比较慢，服务线程会等在 IO 上，让 CPU 闲置，增加响应时间。
- 就算有 buffer，还是不灵。多个线程一起写，为了不至于把 buffer 写错乱，往往要加锁。这会让服务线程互相等待，降低并发度。（同时用多个 log 文件不是办法，除非你有多个磁盘，且保证 log files 分散在不同的磁盘上，否则还是受到磁盘 IO 瓶颈制约。）

解决办法是单独用一个 logging 线程，负责写磁盘文件，通过一个或多个 BlockingQueue 对外提供接口。别的线程要写日志的时候，先把消息（字符串）准备好，然后往 queue 里一塞就行，基本不用等待。这样服务线程的计算就和 logging 线程的磁盘 IO 相互重叠，降低了服务线程的响应时间。

尽管 logging 很重要，但它不是程序的主要逻辑，因此对程序的结构影响越小越好，最好能简单到如同一条 printf 语句，且不用担心其他性能开销，而一个好的多线程异步 logging 库能帮我们做到这一点。（Apache 的 log4cxx 和 log4j 都支持 AsyncAppender 这种异步 logging 方式。）

例2: memcached 客户端

假设我们用 memcached 来保存用户最后发帖的时间，那么每次响应用户发帖的请求时，程序里要去设置一下 memcached 里的值。这一步如果用同步 IO，会增加延迟。

对于"设置一个值"这样的 write-only idempotent 操作，我们其实不用等 memcached 返回操作结果，这里也不用在乎 set 操作失败，那么可以借助多线程来降低响应延迟。比方说我们可以写一个多线程版的 memcached 的客户端，对于 set 操作，调用方只要把 key 和 value 准备好，调用一下 asyncSet() 函数，把数据往 BlockingQueue 上一放就能立即返回，延迟很小。剩下的时就留给 memcached 客户端的线程去操心，而服务线程不受阻碍。

其实所有的网络写操作都可以这么异步地做，不过这也有一个缺点，那就是每次 asyncWrite 都要在线程间传递数据，其实如果 TCP 缓冲区是空的，我们可以在本线程写完，不用劳烦专门的 IO 线程。Jboss 的 Netty 就使用了这个办法来进一步降低延迟。

以上都仅讨论了"打一枪就跑"的情况，如果是一问一答，比如从 memcached 取一个值，那么"重叠 IO"并不能降低响应时间，因为你无论如何要等 memcached 的回复。这时我们可以用别的方式来提高并发度，见问题8。（虽然不能降低响应时间，但也不要浪费线程在空等上，对吧）

另外以上的例子也说明，BlockingQueue 是构建多线程程序的利器。

6. 为什么第三方库往往要用自己的线程？

3. 学之者生，用之者死——ACE历史与简评(20)
4. 关于 TCP 并发连接的几个思考题与试验(20)
5. 发布一个基于 Reactor 模式的 C++ 网络库(17)

推荐排行榜

1. 谈一谈网络编程学习经验（06-08 更新）(30)
2. 关于 TCP 并发连接的几个思考题与试验(18)
3. 从《C++ Primer 第四版》入手学习 C++(16)
4. 分布式系统的工程化开发方法(10)
5. 计算机图书赠送(10)

往往因为 event loop 模型没有标准实现。如果自己写代码，尽可以按所用 Reactor 的推荐方式来编程，但是第三方库不一定能很好地适应并融入这个 event loop framework。有时需要用线程来做一些串并转换。

对于 Java，这个问题还好办一些，因为 thread pool 在 Java 里有标准实现，叫 ExecutorService。如果第三方库支持线程池，那么它可以和主程序共享一个 ExecutorService ，而不是自己创建一堆线程。（比如在初始化时传入主程序的 obj。）对于 C++，情况麻烦得多，Reactor 和 Thread pool 都没有标准库。

例1：libmemcached 只支持同步操作

libmemcached 支持所谓的"非阻塞操作"，但没有暴露一个能被 select/poll/epoll 的 file describer，它的 memcached_fetch 始终会阻塞。它号称 memcached_set 可以是非阻塞的，实际意思是不必等待结果返回，但实际上这个函数会同步地调用 write()，仍可能阻塞在网络 IO 上。

如果在我们的 reactor event handler 里调用了 libmemcached 的函数，那么 latency 就堪忧了。如果想继续用 libmemcached，我们可以为它做一次线程封装，按问题 5 例 2 的办法，同额外的线程专门做 memcached 的 IO，而程序主体还是 reactor。甚至可以把 memcached "数据就绪"作为一个 event，注入到我们的 event loop 中，以进一步提高并发度。（例子留待问题 8 讲）

万幸的是，memcached 的协议非常简单，大不了可以自己写一个基于 reactor 的客户端，但是数据库客户端就没那么幸运了。

例2：MySQL 的官方 C API 不支持异步操作

MySQL 的客户端只支持同步操作，对于 UPDATE/INSERT/DELETE 之类只要行为不管结果的操作（如果代码需要得知其执行结果则另当别论），我们可以用一个单独的线程来做，以降低服务线程的延迟。可仿照前面 memcached_set 的例子，不再赘言。麻烦的是 SELECT，如果要把它也异步化，就得动用更复杂的模式了，见问题 8。

相比之下，PostgreSQL 的 C 客户端 libpq 的设计要好得多，我们可以用 PQsendQuery() 来发起一次查询，然后用标准的 select/poll/epoll 来等待 PQsocket，如果有数据可读，那么用 PQconsumeInput 处理之，并用 PQisBusy 判断查询结果是否已就绪，最后用 PQgetResult 来获取结果。借助这套异步 API，我们可以很容易地为 libpq 写一套 wrapper，使之融入到程序所用的 reactor 模型中。

7. 什么是线程池大小的阻抗匹配原则？

我在《常用模型》中提到"阻抗匹配原则"，这里大致讲一讲。

如果池中线程在执行任务时，密集计算所占的时间比重为 P （0 < P <= 1），而系统一共有 C 个 CPU，为了让这 C 个 CPU 跑满而又不过载，线程池大小的经验公式 T = C/P。（T 是个 hint，考虑到 P 值的估计不是很准确，T 的最佳值可以上下浮动 50%。）

以后我再讲这个经验公式是怎么来的，先验证边界条件的正确性。

假设 C = 8, P = 1.0，线程池的任务完全是密集计算，那么 T = 8。只要 8 个活动线程就能让 8 个 CPU 饱和，再多也没用，因为 CPU 资源已经耗光了。

假设 C = 8, P = 0.5，线程池的任务有一半是计算，有一半等在 IO 上，那么 T = 16。考虑操作系统能灵活合理地调度 sleeping/writing/running 线程，那么大概 16 个“50% 繁忙的线程”能让 8 个 CPU 忙个不停。启动更多的线程并不能提高吞吐量，反而因为增加上下文切换的开销而降低性能。

如果 P < 0.2，这个公式就不适用了，T 可以取一个固定值，比如 5*C。

另外，公式里的 C 不一定是 CPU 总数，可以是“分配给这项任务的 CPU 数目”，比如在 8 核机器上分出 4 个核来做一项任务，那么 C=4。

8. 除了你推荐的 **reactor + thread poll**，还有别的 **non-trivial** 多线程编程模型吗？

有，Proactor。

如果一次请求响应中要和别的进程打多次交道，那么 proactor 模型往往能做到更高的并发度。当然，代价是代码变得支离破碎，难以理解。

这里举 http proxy 为例，一次 http proxy 的请求如果没有命中本地 cache，那么它多半会：

1. 解析域名 （不要小看这一步，对于一个陌生的域名，解析可能要花半秒钟）
2. 建立连接
3. 发送 HTTP 请求
4. 等待对方回应
5. 把结果返回客户

这 5 步里边跟 2 个 server 发生了 3 次 round-trip：

1. 向 DNS 问域名，等待回复；
2. 向对方 http 服务器发起连接，等待 TCP 三路握手完成；
3. 向对方发送 http request，等待对方 response。

而实际上 http proxy 本身的运算量不大，如果用线程池，池中线程的数目会很庞大，不利于操作系统管理调度。

这时我们有两个解决思路：

1. 把“域名已解析”，“连接已建立”，“对方已完成响应”做成 event，继续按照 Reactor 的方式来编程。这样一来，每次客户请求就不能用一个函数从头到尾执行完成，而要分成多个阶段，并且要管理好请求的状态（“目前到了第几步？”）。

2. 用回调函数，让系统来把任务串起来。比如收到用户请求，如果没有命中本地 cache，立刻发起异步的 DNS 解析 startDNSResolve()，告诉系统在解析完之后调用 DNSResolved() 函数；在 DNSResolved() 中，发起连接，告诉系统在连接建立之后调用 connectionEstablished()；在 connectionEstablished() 中发送 http request，告诉系统在收到响应之后调用 httpResponsed()；最后，在 httpResponsed() 里把结果返回给客户。.NET 大量采用的 Begin/End 操作也是这个编程模式。当然，对于不熟悉这种编程方式的人，代码会显得很难看。Proactor 模式的例子可看 boost::asio 的文档，这里不再多说。

Proactor 模式依赖操作系统或库来高效地调度这些子任务，每个子任务都不会阻塞，因此能用比较少的线程达到很高的 IO 并发度。

Proactor 能提高吞吐，但不能降低延迟，所以我没有深入研究。

9. 模式 2 和模式 3a 该如何取舍？

这里的"模式"不是 pattern，而是 model，不巧它们的中译是一样的。《适用场合》中提到，模式 2 是一个多线程的进程，模式 3a 是多个相同的单线程进程。

我认为，在其他条件相同的情况下，可以根据工作集 (work set) 的大小来取舍。工作集是指服务程序响应一次请求所访问的内存大小。

如果工作集较大，那么就用多线程，避免 CPU cache 换入换出，影响性能；否则，就用单线程多进程，享受单线程编程的便利。

例如，memcached 这个内存消耗大户用多线程服务端就比在同一台机器上运行多个 memcached instance 要好。（除非你在 16G 内存的机器上运行 32-bit memcached，那么多 instance 是必须的。）

又例如，求解 Sudoku 用不了多大内存，如果单线程编程更方便的话，可以用单线程多进程来做。再在前面加一个单线程的 load balancer，仿 lighttpd + fastcgi 的成例。

线程不能减少工作量，即不能减少 CPU 时间。如果解决一个问题需要执行一亿条指令（这个数字不大，不要被吓到），那么用多线程只会让这个数字增加。但是通过合理调配这一亿条指令在多个核上的执行情况，我们能让工期提早结束。这听上去像统筹方法，确实也正是统筹方法。

请注意，我一般不在博客的评论跟帖中回复匿名用户的提问，如果希望我解答疑惑，请：1. 给我写信，2. 在 twitter 上 follow @bnu_chenshuo，3. 登陆以后评论。 如有不便，还请见谅。

分类: 多线程

陈硕
关注 - 0
粉丝 - 1170
+加关注

5 0

« 上一篇：多线程服务器的适用场合
» 下一篇：学之者生，用之者死——ACE历史与简评

posted on 2010-03-03 19:58 陈硕 阅读(3128) 评论(4) 编辑 收藏

评论

#1楼 2010-03-03 20:25 life++

我把你的头像看到老赵了。
老赵最近少活动了。
回复完看正文@_@

支持(0) 反对(0)

#2楼 [楼主] 2010-03-03 20:26 陈硕

@ 南院那
老赵比我瘦。

支持(0) 反对(0)

#3楼 2010-03-12 23:52 蛙蛙王子

看完了，写的很好，.NET把我们都惯坏了，对线程的使用我们没这么讲究，全用自带的ThreadPool。
写日志我们也是先写到一个内存队列里，一个独立的非线程池线程扫描队列，写到磁盘上。
对HTTP PROXY这种的确实挺麻烦，HTTP是一问一答式的，你虽然可以用异步发送请求，但你得等上一个请求的应答回来后才能发送下一个请求（在启用keep alive的情况下），虽然HTTP支持pipeline，但你同时发N个请求，这N个请求在服务端必须是按顺序返回的，对HTTP服务器的编写有一定的复杂性，一次发送A,B,C三个请求，如果C执行最快，也得等A执行完了，才能按A,B,C的顺序把应答反回来。我理解的是这样的，不知道对不对。
还有你说的轻松到达成千上万个连接，甚至上万个连接，我觉得应该比较少吧，用.NET做的网络服务器都能达到十几w个长连接，服务端监听一个端口所能承载的连接数是和未分页的内存数有关系，64位的大内存机器肯定不止几万个连接。
建议有空讲讲缓冲区的使用的一些经验。

支持(0) 反对(0)

#4楼 2010-05-25 14:32 三枚紫橄榄

你好，你在第8点中说到，Proactor会导致代价是代码变得支离破碎，难以理解.理由是需要保存状态，而不能再一个函数执行完毕.

reactor + thread poll难道就不是吗? 如果把你提到的所有步骤放到一个函数中, 阻塞还是异步执行? 如果异步执行, 岂不是跟proactor一样?如果是阻塞执行, 那么proactor也可以启动多个线程来进行这些操作?

有空的话,能否解释一下, 多谢了

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。