多线程服务器的适用场合

陈硕 (giantchen_AT_gmail)

www.cnblogs.com/Solstice

2010 Feb 28

这篇文章原本是前一篇博客《多线程服务器的常用编程模型》（以下简称《常用模型》）计划中的一节，今天终于写完了。

“服务器开发”包罗万象，本文所指的“服务器开发”的含义请见《常用模型》一文，一句话形容是：跑在多核机器上的 Linux 用户态的没有用户界面的长期运行的网络应用程序。“长期运行”的意思不是指程序 7x24 不重启，而是程序不会因为无事可做而退出，它会等着下一个请求的到来。例如 wget 不是长期运行的，httpd 是长期运行的。

正名

与前文相同，本文的“进程”指的是 fork() 系统调用的产物。“线程”指的是 pthread_create() 的产物，而且我指的 pthreads 是 NPTL 的，每个线程由 clone() 产生，对应一个内核的 task_struct。本文所用的开发语言是 C++，运行环境为 Linux。

首先，一个由多台机器组成的分布式系统必然是多进程的（字面意义上），因为进程不能跨 OS 边界。在这个前提下，我们把目光集中到一台机器，一台拥有至少 4 个核的普通服务器。如果要在一台多核机器上提供一种服务或执行一个任务，可用的模式有：

1. 运行一个单线程的进程
2. 运行一个多线程的进程
3. 运行多个单线程的进程
4. 运行多个多线程的进程

这些模式之间的比较已经是老生常谈，简单地总结：

- 模式 1 是不可伸缩的 (scalable)，不能发挥多核机器的计算能力；
- 模式 3 是目前公认的主流模式。它有两种子模式：
  - 3a 简单地把模式 1 中的进程运行多份，如果能用多个 tcp port 对外提供服务的话；
  - 3b 主进程+woker进程，如果必须绑定到一个 tcp port，比如 httpd+fastcgi。
- 模式 2 是很多人鄙视的，认为多线程程序难写，而且不比模式 3 有什么优势；

- 模式 4 更是千夫所指，它不但没有结合 2 和 3 的优点，反而汇聚了二者的缺点。

本文主要想讨论的是模式 2 和模式 3b 的优劣，即：什么时候一个服务器程序应该是多线程的。

从功能上讲，没有什么是多线程能做到而单线程做不到的，反之亦然，都是状态机嘛（我很高兴看到反例）。从性能上讲，无论是 **IO bound** 还是 **CPU bound** 的服务，多线程都没有什么优势。那么究竟为什么要用多线程？

在回答这个问题之前，我先谈谈必须用必须用单线程的场合。

必须用单线程的场合

据我所知，有两种场合必须使用单线程：

1. 程序可能会 fork()
2. 限制程序的 CPU 占用率

先说 fork()，我在《Linux 新增系统调用的启示》中提到：

> fork() 一般不能在多线程程序中调用，因为 Linux 的 fork() 只克隆当前线程的 thread of control，不克隆其他线程。也就是说不能一下子 fork() 出一个和父进程一样的多线程子进程，Linux 也没有 forkall() 这样的系统调用。forkall() 其实也是很难办的（从语意上），因为其他线程可能等在 condition variable 上，可能阻塞在系统调用上，可能等着 mutex 以跨入临界区，还可能在密集的计算中，这些都不好全盘搬到子进程里。

更为糟糕的是，如果在 fork() 的一瞬间某个别的线程 a 已经获取了 mutex，由于 fork() 出的新进程里没有这个“线程a”，那么这个 mutex 永远也不会释放，新的进程就不能再获取那个 mutex，否则会死锁。（这一点仅为推测，还没有做实验，不排除 fork() 会释放所有 mutex 的可能。）

综上，一个设计为可能调用 fork() 的程序必须是单线程的，比如我在《启示》一文中提到的“看门狗进程”。多线程程序不是不能调用 fork()，而是这么做会遇到很多麻烦，我想不出做的理由。

一个程序 fork() 之后一般有两种行为：

1. 立刻执行 exec()，变身为另一个程序。例如 shell 和 inetd；又比如 lighttpd fork() 出子进程，然后运行 fastcgi 程序。或者集群中运行在计算节点上的负责启动 job 的守护进程（即我所谓的“看门狗进程”）。
2. 不调用 exec()，继续运行当前程序。要么通过共享的文件描述符与父进程通信，协同完成任务；要么接过父进程传来的文件描述符，独立完成工作，例如 80 年代的 web 服务器 NCSA httpd。

这些行为中，我认为只有“看门狗进程”必须坚持单线程，其他的均可替换为多线程程序（从功能上讲）。

单线程程序能限制程序的 CPU 占用率。

这个很容易理解，比如在一个 8-core 的主机上，一个单线程程序即便发生 busy-wait（无论是因为 bug 还是因为

overload），其 CPU 使用率也只有 12.5%，即沾满 1 个 core。在这种最坏的情况下，系统还是有 87.5% 的计算资源可供其他服务进程使用。

因此对于一些辅助性的程序，如果它必须和主要功能进程运行在同一台机器的话（比如它要监控其他服务进程的状态），那么做成单线程的能避免过分抢夺系统的计算资源。

基于进程的分布式系统设计

《常用模型》一文提到，分布式系统的软件设计和功能划分一般应该以“进程”为单位。我提倡用多线程，并不是说把整个系统放到一个进程里实现，而是指功能划分之后，在实现每一类服务进程时，在必要时可以借助多线程来提高性能。对于整个分布式系统，要做到能 scale out，即享受增加机器带来的好处。

对于上层的应用而言，每个进程的代码量控制在 10 万行 C++ 以下，这不包括现成的 library 的代码量。这样每个进程都能被一个脑子完全理解，不会出现混乱。（其实我更想说 5 万行。）

这里推荐一篇 Google 的好文《Introduction to Distributed System Design》。其中点睛之笔是：分布式系统设计，是 design for failure。

本文继续讨论一个服务进程什么时候应该用多线程，先说说单线程的优势。

单线程程序的优势

从编程的角度，单线程程序的优势无需赘言：简单。程序的结构一般如《常用模型》所言，是一个基于 IO multiplexing 的 event loop。或者如云风所言，直接用阻塞 IO。

event loop 的典型代码框架是：

```
while (!done) {
  int retval = ::poll(fds, nfds, timeout_ms);
  if (retval < 0) {
    处理错误
  } else {
    处理到期的 timers
    if (retval > 0) {
      处理 IO 事件
    }
  }
}
```

event loop 有一个明显的缺点，它是非抢占的(non-preemptive)。假设事件 a 的优先级高于事件 b，处理事件 a 需要

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(18735)
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)

1ms，处理事件 b 需要 10ms。如果事件 b 稍早于 a 发生，那么当事件 a 到来时，程序已经离开了 poll() 调用开始处理事件 b。事件 a 要等上 10ms 才有机会被处理，总的响应时间为 11ms。这等于发生了优先级反转。

这可缺点可以用多线程来克服，这也是多线程的主要优势。

多线程程序有性能优势吗？

前面我说，无论是 IO bound 还是 CPU bound 的服务，多线程都没有什么绝对意义上的性能优势。这里详细阐述一下这句话的意思。

这句话是说，如果用很少的 CPU 负载就能让的 IO 跑满，或者用很少的 IO 流量就能让 CPU 跑满，那么多线程没啥用处。举例来说：

1. 对于静态 web 服务器，或者 ftp 服务器，CPU 的负载较轻，主要瓶颈在磁盘 IO 和网络 IO。这时候往往一个单线程的程序（模式 1）就能撑满 IO。用多线程并不能提高吞吐量，因为 IO 硬件容量已经饱和了。同理，这时增加 CPU 数目也不能提高吞吐量。
2. CPU 跑满的情况比较少见，这里我只好虚构一个例子。假设有一个服务，它的输入是 n 个整数，问能否从中选出 m 个整数，使其和为 0 （这里 n < 100, m > 0）。这是著名的 subset sum 问题，是 NP-Complete 的。对于这样一个“服务”，哪怕很小的 n 值也会让 CPU 算死，比如 n = 30，一次的输入不过 120 字节（32-bit 整数），CPU 的运算时间可能长达几分钟。对于这种应用，模式 3a 是最适合的，能发挥多核的优势，程序也简单。

也就是说，无论任何一方早早地先到达瓶颈，多线程程序都没啥优势。

说到这里，可能已经有读者不耐烦了：你讲了这么多，都在说单线程的好处，那么多线程究竟有什么用？

适用多线程程序的场景

我认为多线程的适用场景是：提高响应速度，让 **IO** 和“计算”相互重叠，降低 **latency**。

虽然多线程不能提高绝对性能，但能提高平均响应性能。

一个程序要做成多线程的，大致要满足：

- 有多个 CPU 可用。单核机器上多线程的优势不明显。
- 线程间有共享数据。如果没有共享数据，用模型 3b 就行。虽然我们应该把线程间的共享数据降到最低，但不代表没有；
- 共享的数据是可以修改的，而不是静态的常量表。如果数据不能修改，那么可以在进程间用 shared memory，模式 3 就能胜任；
- 提供非均质的服务。即，事件的响应有优先级差异，我们可以用专门的线程来处理优先级高的事件。防止优先级反转；
- latency 和 throughput 同样重要，不是逻辑简单的 IO bound 或 CPU bound 程序；

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)

- 利用异步操作。比如 logging。无论往磁盘写 log file，还是往 log server 发送消息都不应该阻塞 critical path；
- 能 scale up。一个好的多线程程序应该能享受增加 CPU 数目带来的好处，目前主流是 8 核，很快就会用到 16 核的机器了。
- 具有可预测的性能。随着负载增加，性能缓慢下降，超过某个临界点之后急速下降。线程数目一般不随负载变化。
- 多线程能有效地划分责任与功能，让每个线程的逻辑比较简单，任务单一，便于编码。而不是把所有逻辑都塞到一个 event loop 里，就像 Win32 SDK 程序那样。

这些条件比较抽象，这里举一个具体的（虽然是虚构的）例子。

假设要管理一个 Linux 服务器机群，这个机群里有 8 个计算节点，1 个控制节点。机器的配置都是一样的，双路四核 CPU，千兆网互联。现在需要编写一个简单的机群管理软件（参考 LLNL 的 SLURM），这个软件由三个程序组成：

- 运行在控制节点上的 master，这个程序监视并控制整个机群的状态。
- 运在每个计算节点上的 slave，负责启动和终止 job，并监控本机的资源。
- 给最终用户的 client 命令行工具，用于提交 job。

根据前面的分析，slave 是个“看门狗进程”，它会启动别的 job 进程，因此必须是个单线程程序。另外它不应该占用太多的 CPU 资源，这也适合单线程模型。

master 应该是个模式 2 的多线程程序：

- 它独占一台 8 核的机器，如果用模型 1，等于浪费了 87.5% 的 CPU 资源。
- 整个机群的状态应该能完全放在内存中，这些状态是共享且可变的。如果用模式 3，那么进程之间的状态同步会成大问题。而如果大量使用共享内存，等于是掩耳盗铃，披着多进程外衣的多线程程序。
- master 的主要性能指标不是 throughput，而是 latency，即尽快地响应各种事件。它几乎不会出现把 IO 或 CPU 跑满的情况。
- master 监控的事件有优先级区别，一个程序正常运行结束和异常崩溃的处理优先级不同，计算节点的磁盘满了和机箱温度过高这两种报警条件的优先级也不同。如果用单线程，可能会出现优先级反转。
- 假设 master 和每个 slave 之间用一个 TCP 连接，那么 master 采用 2 个或 4 个 IO 线程来处理 8 个 TCP connections 能有效地降低延迟。
- master 要异步的往本地硬盘写 log，这要求 logging library 有自己的 IO 线程。
- master 有可能要读写数据库，那么数据库连接这个第三方 library 可能有自己的线程，并回调 master 的代码。
- master 要服务于多个 clients，用多线程也能降低客户响应时间。也就是说它可以再用 2 个 IO 线程专门处理和 clients 的通信。
- master 还可以提供一个 monitor 接口，用来广播 (pushing) 机群的状态，这样用户不用主动轮询 (polling)。这个功能如果用单独的线程来做，会比较容易实现，不会搞乱其他主要功能。
- master 一共开了 10 个线程：

    - 4 个用于和 slaves 通信的 IO 线程
    - 1 个 logging 线程
    - 1 个数据库 IO 线程
    - 2 个和 clients 通信的 IO 线程
    - 1 个主线程，用于做些背景工作，比如 job 调度
    - 1 个 pushing 线程，用于主动广播机群的状态
- 虽然线程数目略多于 core 数目，但是这些线程很多时候都是空闲的，可以依赖 OS 的进程调度来保证可控的延迟。

综上所述，master 用多线程方式编写是自然且高效的。

线程的分类

据我的经验，一个多线程服务程序中的线程大致可分为 3 类：

1. IO 线程，这类线程的的主循环是 io multiplexing，等在 select/poll/epoll 系统调用上。这类线程也处理定时事件。当然它的功能不止 IO，有些计算也可以放入其中。
2. 计算线程，这类线程的主循环是 blocking queue，等在 condition variable 上。这类线程一般位于 thread pool 中。
3. 第三方库所用的线程，比如 logging，又比如 database connection。

服务器程序一般不会频繁地启动和终止线程。甚至，在我写过的程序里，create thread 只在程序启动的时候调用，在服务运行期间是不调用的。

在多核时代，多线程编程是不可避免的，“鸵鸟算法”不是办法。

分类: 多线程

陈硕
关注 - 0
粉丝 - 1170
+加关注

« 上一篇：Linux 新增系统调用的启示
» 下一篇：《多线程服务器的适用场合》例释与答疑

posted on 2010-02-28 21:32 陈硕 阅读(7225) 评论(5) 编辑 收藏

评论

#1楼 2010-02-28 23:33 个人知识管理

收集到知识库中啊研读。
多线程编程在于做服务器端开发是最大的难点
（建议中高级的软件工程师确保自己能读懂此文 :)）

支持(0) 反对(0)

#2楼 2010-03-01 00:09 2828.com

收藏了。。

#3楼 2010-03-01 09:09 木乃伊

大概浏览了下，先上班，晚上回家慢慢看。

支持(0) 反对(0)

#4楼 2010-03-01 09:28 Old

好文章。

支持(0) 反对(0)

#5楼 2010-03-04 08:59 汝熹

学习了

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

相关博文：
· volatile的适用场合
· WebService 适用场合
· volatile适用场合
· 脚本适用场合
· 《多线程服务器的适用场合》例释与答疑

最新新闻：
· 24小时争夺战：便利店夜未眠
· 百度寻找新增长点：百度云承担百亿营收目标 今年将扩招近两千人
· 如何用"广场舞"模型设计社区团购增长
· 京东新开两家云计算公司 刘强东任总经理
· 折叠屏手机根本没想卖给你
» 更多新闻...