加餐 | 单服务器高性能模式性能对比

2020-12-02 李运华 来自北京

《从0开始学架构》

你好，我是华仔。

我们架构课的第 18 讲和第 19 讲主题是单服务器高性能模式，我们讲了 PPC 与 TPC、Reactor 与 Proactor，从理论上跟你详细讲述了不同模式的实现方式和优缺点，但是并没有给出详细的测试数据对比，原因在于我自己没有整套的测试环境，也不能用公司的服务器做压力测试，因此留下了一个小小的遗憾。

幸运的是，最近我在学习的时候，无意中在网络上找到一份非常详尽的关于 Linux 服务器网络模型的详细系列文章。作者通过连载的方式，将 iterative、forking（对应专栏的 PPC 模式）、preforked（对应专栏的 prefork 模式）、threaded（对应专栏的 TPC 模式）、prethreaded（对应专栏的 prethread 模式）、poll、epoll（对应专栏的 Reactor 模式）共 7 种模式的实现原理、实现代码、性能对比都详尽地进行了阐述，完美地弥补了专栏内容没有实际数据对比的遗憾。

因此我把核心的测试数据对比摘录出来，然后基于数据来进一步阐释，也就有了这一讲的加餐。我想第一时间分享给你，相信今天的内容可以帮助我们加深对课程里讲过的理论的理解。

下面是作者对 7 种模式的性能测试对比结果表格，作者在文章中并没有详细地介绍测试环境，只是简单提到了测试服务器是租来的云服务器，**CPU 只有 1 核**（没有说明具体的 CPU 型号），对于内存、带宽、磁盘等信息并没有介绍，我们假设这些硬件相关性能都足够。从理论上来说，网络模型的核心性能部件就是 CPU，因此如下数据是具备参考意义的。

	requests/second						
concurrency	**iterative**	**forking**	**preforked**	**threaded**	**prethreaded**	**poll**	**epoll**
20	7	112	2,100	1,800	2,250	1,900	2,050
50	7	190	2,200	1,700	2,200	2,000	2,000
100	7	245	2,200	1,700	2,200	2,150	2,100
200	7	330	2,300	1,750	2,300	2,200	2,100
300	–	380	2,200	1,800	2,400	2,250	2,150
400	–	410	2,200	1,750	2,600	2,000	2,000
500	–	440	2,300	1,850	2,700	1,900	2,212
600	–	460	2,400	1,800	2,500	1,700	2,519
700	–	460	2,400	1,600	2,490	1,550	2,607
800	–	460	2,400	1,600	2,540	1,400	2,553
900	–	460	2,300	1,600	2,472	1,200	2,567
1,000	–	475	2,300	1,700	2,485	1,150	2,439
1,500	–	490	2,400	1,550	2,620	900	2,479
2,000	–	350	2,400	1,400	2,396	550	2,200
2,500	–	280	2,100	1,300	2,453	490	2,262
3,000	–	280	1,900	1,250	2,502	wide variations	2,138
5,000	–	wide variations	1,600	1,100	2,519	–	2,235
8,000	–	–	1,200	wide variations	2,451	–	2,100
10,000	–	–	wide variations	–	2,200	–	2,200
11,000	–	–	–	–	2,200	–	2,122
12,000	–	–	–	–	970	–	1,958
13,000	–	–	–	–	730	–	1,897
14,000	–	–	–	–	590	–	1,466
15,000	–	–	–	–	532	–	1,281

这张图的数据比较多，如何去看懂这样的性能测试数据表格呢？我来分享一个有用的技巧：**横向看对比，纵向看转折。**

横向看对比

比如，当并发连接数是 1000 的时候，可以看出 preforking、prethreaded、epoll 三种模式性能是相近的，也意味着 epoll 并不是在任何场景都具备相比其它模式的性能优势。

requests/second							
concurrency	iterative	forking	preforked	threaded	prethreaded	poll	epoll
900	–	460	2,300	1,600	2,472	1,200	2,567
1,000	–	475	2,300	1,700	2,485	1,150	2,439
1,500	–	490	2,400	1,550	2,620	900	2,479

纵向看转折

比如，prethreaded 模式（作者源码中设置了 100 个线程）在 11000 并发的时候性能有 2200，但 12000 并发连接的时候，性能急剧下降到只有 970，这是什么原因呢？我推测是 12000 并发的时候触发了 C10K 问题，线程上下文切换的性能消耗超越了 IO 处理，成为了系统的处理瓶颈。

requests/second							
concurrency	iterative	forking	preforked	threaded	prethreaded	poll	epoll
5,000	–	wide variations	1,600	1,100	2,519	–	2,235
8,000	–	–	1,200	wide variations	2,451	–	2,100
10,000	–	–	wide variations	–	2,200	–	2,200
11,000	–	–	–	–	2,200	–	2,122
12,000	–	–	–	–	970	–	1,958
13,000	–	–	–	–	730	–	1,897
14,000	–	–	–	–	590	–	1,466
15,000	–	–	–	–	532	–	1,281

按照上述“横向看对比，纵向看转折”的方式，我给你分享一下我的一些解读和发现。

1. 创建进程的消耗是创建线程的消耗的 4 倍左右。

requests/second							
concurrency	iterative	forking	preforked	threaded	prethreaded	poll	epoll
400	–	410	2,200	1,750	2,600	2,000	2,000
500	–	440	2,300	1,850	2,700	1,900	2,212
600	–	460	2,400	1,800	2,500	1,700	2,519
700	–	460	2,400	1,600	2,490	1,550	2,607
800	–	460	2,400	1,600	2,540	1,400	2,553
900	–	460	2,300	1,600	2,472	1,200	2,567
1,000	–	475	2,300	1,700	2,485	1,150	2,439
1,500	–	490	2,400	1,550	2,620	900	2,479
2,000	–	350	2,400	1,400	2,396	550	2,200
2,500	–	280	2,100	1,300	2,453	490	2,262
3,000	–	280	1,900	1,250	2,502	wide variations	2,138

2. 并发 2000 以内时，preforked、prethreaded、epoll 的性能相差无几，甚至 preforked 和 prethreaded 的性能有时候还稍微高一些。

requests/second							
concurrency	iterative	forking	preforked	threaded	prethreaded	poll	epoll
20	7	112	2,100	1,800	2,250	1,900	2,050
50	7	190	2,200	1,700	2,200	2,000	2,000
100	7	245	2,200	1,700	2,200	2,150	2,100
200	7	330	2,300	1,750	2,300	2,200	2,100
300	–	380	2,200	1,800	2,400	2,250	2,150
400	–	410	2,200	1,750	2,600	2,000	2,000
500	–	440	2,300	1,850	2,700	1,900	2,212
600	–	460	2,400	1,800	2,500	1,700	2,519
700	–	460	2,400	1,600	2,490	1,550	2,607
800	–	460	2,400	1,600	2,540	1,400	2,553
900	–	460	2,300	1,600	2,472	1,200	2,567
1,000	–	475	2,300	1,700	2,485	1,150	2,439
1,500	–	490	2,400	1,550	2,620	900	2,479
2,000	–	350	2,400	1,400	2,396	550	2,200

这也是内部系统、中间件等并发数并不高的系统并不一定需要 epoll 的原因，用 preforked 和 prethreaded 模式能够达到相同的性能，并且实现要简单。

3. 当并发数达到 8000 以上，只有 pthreaded 和 epoll 模式能够继续运行，但性能也有下降，epoll 的下降更加平稳一些。

requests/second							
concurrency	iterative	forking	preforked	threaded	prethreaded	poll	epoll
3,000	–	280	1,900	1,250	2,502	wide variations	2,138
5,000	–	wide variations	1,600	1,100	2,519	–	2,235
8,000	–	–	1,200	wide variations	2,451	–	2,100
10,000	–	–	wide variations	–	2,200	–	2,200
11,000	–	–	–	–	2,200	–	2,122
12,000	–	–	–	–	970	–	1,958
13,000	–	–	–	–	730	–	1,897
14,000	–	–	–	–	590	–	1,466
15,000	–	–	–	–	532	–	1,281

4. prethreaded 模式在 12000 并发连接的时候，性能急剧下降。

requests/second							
concurrency	iterative	forking	preforked	threaded	prethreaded	poll	epoll
5,000	–	wide variations	1,600	1,100	2,519	–	2,235
8,000	–	–	1,200	wide variations	2,451	–	2,100
10,000	–	–	wide variations	–	2,200	–	2,200
11,000	–	–	–	–	2,200	–	2,122
12,000	–	–	–	–	970	–	1,958
13,000	–	–	–	–	730	–	1,897
14,000	–	–	–	–	590	–	1,466
15,000	–	–	–	–	532	–	1,281

推测是触发了 C10K 问题，线程上下文切换的性能消耗超越了 IO 处理的性能消耗。

5. poll 模式随着并发数增多稳定下降，因为需要遍历的描述符越多，其性能越低。

requests/second							
concurrency	iterative	forking	preforked	threaded	prethreaded	poll	epoll
20	7	112	2,100	1,800	2,250	1,900	2,050
50	7	190	2,200	1,700	2,200	2,000	2,000
100	7	245	2,200	1,700	2,200	2,150	2,100
200	7	330	2,300	1,750	2,300	2,200	2,100
300	–	380	2,200	1,800	2,400	2,250	2,150
400	–	410	2,200	1,750	2,600	2,000	2,000
500	–	440	2,300	1,850	2,700	1,900	2,212
600	–	460	2,400	1,800	2,500	1,700	2,519
700	–	460	2,400	1,600	2,490	1,550	2,607
800	–	460	2,400	1,600	2,540	1,400	2,553
900	–	460	2,300	1,600	2,472	1,200	2,567
1,000	–	475	2,300	1,700	2,485	1,150	2,439
1,500	–	490	2,400	1,550	2,620	900	2,479
2,000	–	350	2,400	1,400	2,396	550	2,200
2,500	–	280	2,100	1,300	2,453	490	2,262
3,000	–	280	1,900	1,250	2,502	wide variations	2,138

类似的还有 select 模式，作者没有单独写 select，因为两者原理基本类似，区别是 select 的最大支持连接数受限于 FD_SETSIZE 这个参数。

6. epoll 在并发数超过 10000 的时候性能开始下降，但下降比较平稳。

requests/second							
concurrency	iterative	forking	preforked	threaded	prethreaded	poll	epoll
20	7	112	2,100	1,800	2,250	1,900	2,050
50	7	190	2,200	1,700	2,200	2,000	2,000
100	7	245	2,200	1,700	2,200	2,150	2,100
200	7	330	2,300	1,750	2,300	2,200	2,100
300	–	380	2,200	1,800	2,400	2,250	2,150
400	–	410	2,200	1,750	2,600	2,000	2,000
500	–	440	2,300	1,850	2,700	1,900	2,212
600	–	460	2,400	1,800	2,500	1,700	2,519
700	–	460	2,400	1,600	2,490	1,550	2,607
800	–	460	2,400	1,600	2,540	1,400	2,553
900	–	460	2,300	1,600	2,472	1,200	2,567
1,000	–	475	2,300	1,700	2,485	1,150	2,439
1,500	–	490	2,400	1,550	2,620	900	2,479
2,000	–	350	2,400	1,400	2,396	550	2,200
2,500	–	280	2,100	1,300	2,453	490	2,262
3,000	–	280	1,900	1,250	2,502	wide variations	2,138
5,000	–	wide variations	1,600	1,100	2,519	–	2,235
8,000	–	–	1,200	wide variations	2,451	–	2,100
10,000	–	–	wide variations	–	2,200	–	2,200
11,000	–	–	–	–	2,200	–	2,122
12,000	–	–	–	–	970	–	1,958
13,000	–	–	–	–	730	–	1,897
14,000	–	–	–	–	590	–	1,466
15,000	–	–	–	–	532	–	1,281

这个结论看起来比较简单，但是却隐含着一个关键的设计点：**epoll 不是万能的，连接数太多的时候单进程 epoll 也是不行的**。这也是为什么 Redis 可以用单进程 Reactor 模式，而 Nginx 必须用多进程 Reactor 模式，因为 Redis 的应用场景是内部访问，并发数一般不会超过 10000；而 Nginx 是互联网访问，并发数很容易超过 10000。

以上是我从性能对比数据中的一些发现，这些发现能够让我们更进一步理解专栏内容中讲到的理论知识和优缺点对比，这些数据也可以指导我们在实际的架构设计中根据应用场景来选择合适的模式。

最后，我也希望你能掌握“**横向看对比，纵向看转折**”这个分析技巧。这个技巧在查阅和审核性能测试数据以及各种对比数据的时候，能够帮助你发现很多数据背后隐含的观点和结论。

拓展阅读与学习指南：

1. 原作者的系列文章请参考：https://unixism.net/2019/04/linux-applications-performance-introduction/
2. 原作者的测试代码 GitHub 仓库地址：https://github.com/shuveb/zerohttpd

3. 原作者的代码实现了一个完整的基本功能的 HTTP 服务器，采用的是短链接的方式，还用到了 Redis 来保存内容，有的代码逻辑是比较复杂的，尤其是 epoll 的实现部分。如果你想自己简单的只是验证网络模型的性能，可以去掉其源码中 HTTP 的实现部分，只是简单地返回“hello world”这样的字符串即可。

精选留言 (7)

两只狮子
2021-02-20

华哥，请教一下concurrence列下面的数据是指的绝对并发数还是指的每秒的请求量呢?

作者回复: 并发连接数，每个连接压满，然后计算得到每秒请求数

8

YHK
2021-05-26

prethread 好强啊。比 poll和select都强，那epoll没出来之前，怎么还有人用多路复用

作者回复: prethread解决不了c10k问题，其实c10k问题出现前，多路复用用的不多

3

leesper
2021-01-13

我用Go写过一个网络库，得益于Go语言的强大，我对每个网络连接开了三个goroutine，一个负责读，一个负责写，一个负责配合工作者线程执行业务逻辑，但我一直想写一个基于epoll的网络库出来，https://github.com/leesper/tao

作者回复: 自己练手java可以直接参考Doug lee 的Java NIO编程的PPT，里面给个样例。

Go语言这样设计我感觉没有发挥Go的优势，三个goroutine之间的消息传递可能比较耗费性能，我觉得直接用一个goroutine处理一个连接比较好，意见供参考

FuriousEric

2022-08-02 来自广东

老师，请教下，表格中concurrency(并发连接数)这栏，作者是使用什么工具模拟这么多并发连接数的呢？也就是用啥工具测试的能产生这么多concurrency, 一般我做测试，用wrk, connections设置20个左右。但是图中concurrency高达1w+，是怎么模拟出来的?

作者回复: jmeter，ab，loadrunner这些测试工具都可以设定并发数（或者叫线程数），然后在多台机器上启动多个测试工具一起压测就可以达到很大的并发数了。

Join

2022-03-10

太棒了华仔, 之前看Linux高性能服务器开发中提到Reactor模式,可以没有测试数据支撑,这下可以弥补这个缺失了

作者回复: 找了好久才找到：）

Geek_c25f25

2021-12-01

从表格中发现另外一个数据，不管并发连接数是多少个，从epoll的数据看，都是2000+一点，也就是这个系统的QPS只能达到2000+。

作者回复: 对的，单核2000已经不错了：）

Geek_c25f25

2021-12-01

神文，都说reactor模式好，但是到底怎么个好还是要有测试数据支撑。老师的分析能力好强大啊，特别是redis,c10k这两个分析真的太精彩了，其他地方看不到这两个情况的分析。

作者回复: 如果有多核服务器测试，效果会更明显：）