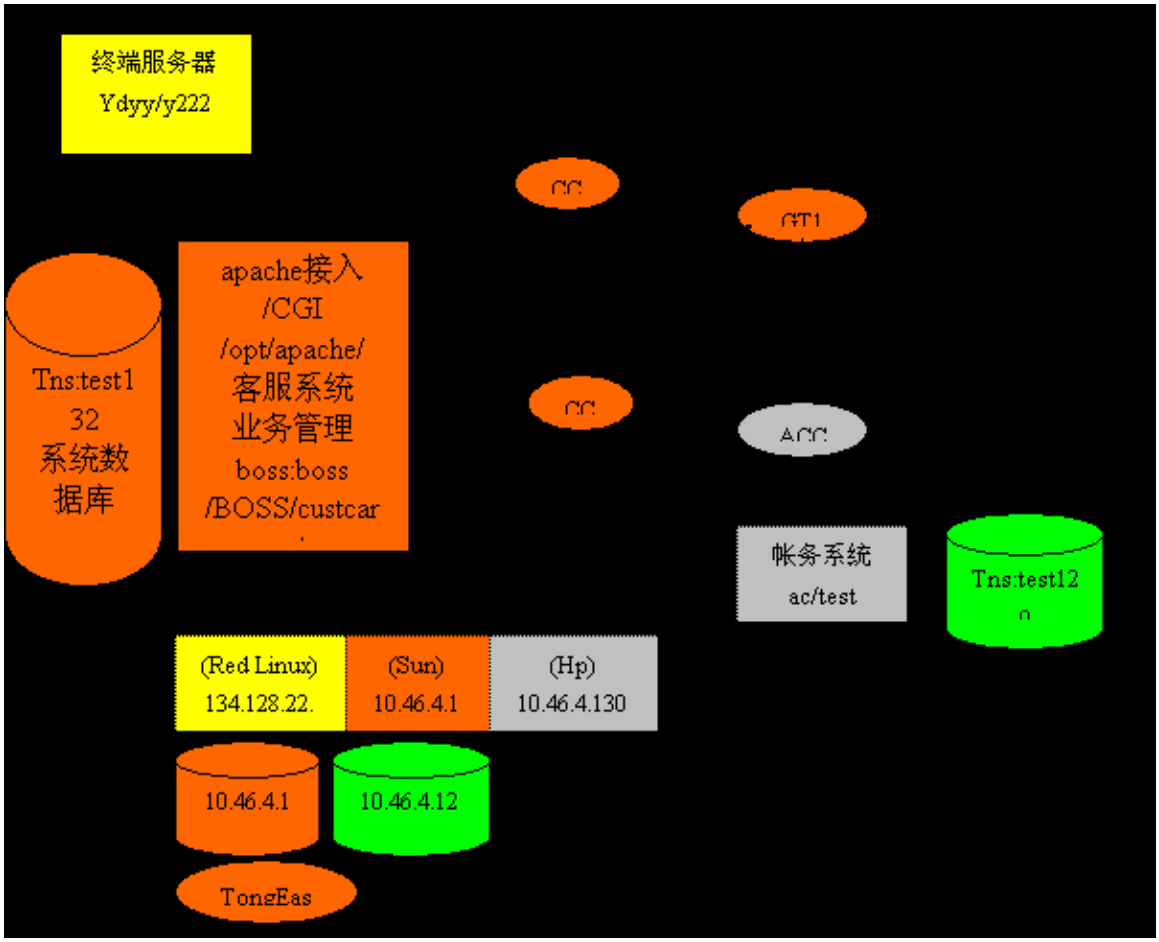
性能测试数据分析经验

作者：TongEASY 组　刘川

我以为福建移动 BOSS 系统做的一个小规模的性能测试为例，谈谈我在数据分析中的一些经验。测试用例，模式如下图。



一台 Linux 模拟 Browser（简称 browser）向主机 SUN 发 HTTP 请求，SUN 上启动 Apache Web Server 将请求交给 FCGI 程序。FCGI 程序作为 TE 节点 CC1（简称 fcgi）的客户进程发起 TE 事务，经 GT1 名字服务向 CC2 （简称 svr_cc）发送一个分支，CC2 上服务嵌套经 GT1 名字服务向 ACCTFZ（在 HP 上，简称 svr_ac）发送一个分支。

测试 3 种压力情况，即 10browser/10fcgi/5svr_cc 服务进程/2svr_ac 服务进程，20browser/20fcgi /10svr_cc 服务进程/2svr_ac 服务进程，30browser/20fcgi/10svr_cc 服务进程/2svr_ac 服务进程。

一．记录数据。

各个部分的应用程序在程序中关键地方记录时间，精确到微秒（毫秒也可以），并按一定格式写入日志文件。这样可以并计算相同应用相临时间点之间的平均时间差（编程序分析日志文件或用 excel 导入）。有了时间差，才能分析出整个系统性能的瓶颈（处理慢的环节）。下面给出一个 fcgi 进程（也是 TE 客户端进 程）的日志文件片段。

[19:21:32.628.512] begin_tpbegin

[19:21:32.628.833] end_tpbegin

[19:21:32.628.944] begin_tpsetbranch

[19:21:32.629.053] end_tpsetbranch

[19:21:32.629.487] begin_tpcall

[19:21:37.102.996] end_tpcall

[19:21:37.103.806] begin_tpcommit

[19:21:37.432.253] end_tpcommit

[19:21:40.405.345] begin_tpbegin

[19:21:40.405.532] end_tpbegin

[19:21:40.405.639] begin_tpsetbranch

[19:21:40.405.742] end_tpsetbranch

[19:21:40.406.175] begin_tpcall

[19:21:46.732.888] end_tpcall

[19:21:46.733.650] begin_tpcommit

[19:21:46.832.538] end_tpcommit

第一个字段是时间点时间，第二个字段是该时间点描述串，两个字段用空格间隔。从这个文件可以看出，fcgi 进程是循环处理业务的。 begin_tpbegin 处开始一笔业

务，begin_tpcall 和 end_tpcall 之间是 TE_tpcall()发起请求到收到应答的时 间，begin_tpbegin 和 end_tpbegin 之间是 TE_tpcommit()发起提交到收到提交应答的时间。而 end_tpcommit 到 begin_tpcall 是 fcgi 进程从一笔业务结束到开始下一笔业务的时间，在这里也就是 fcgi 进程从 Web Server 获取 HTTP 请求的时间。

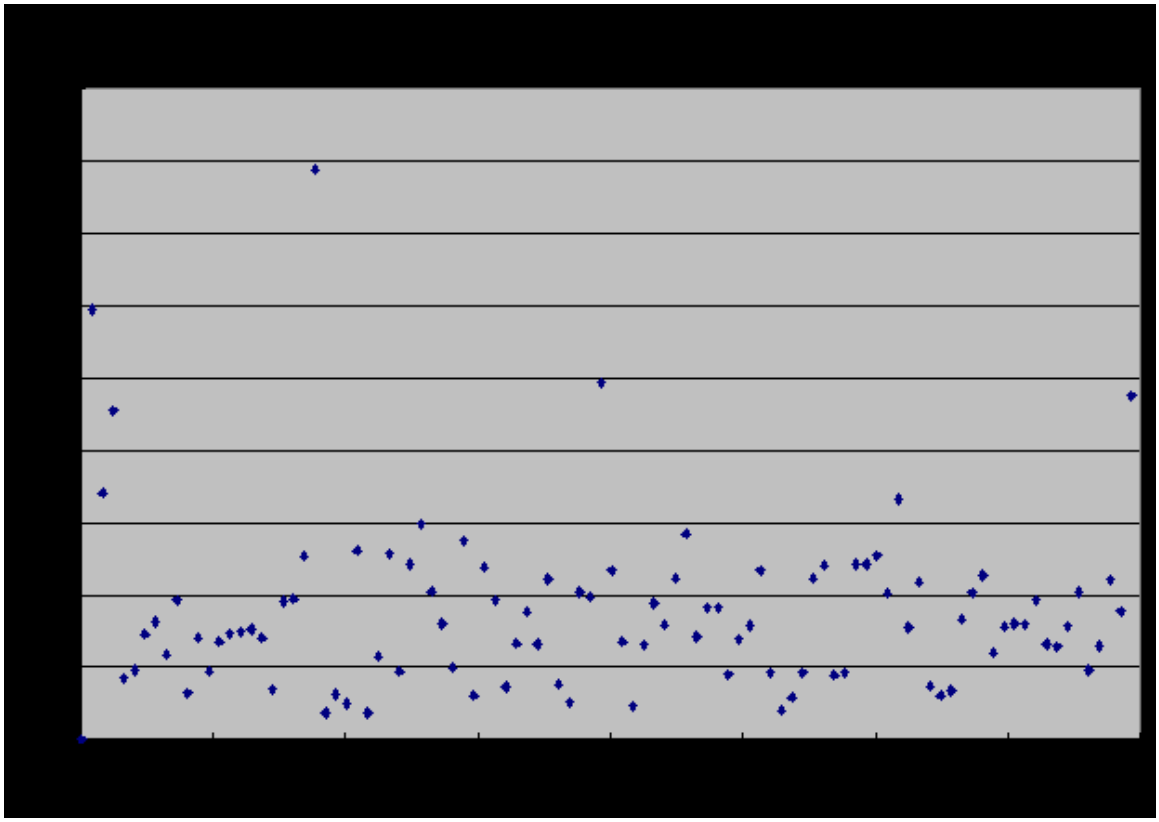
从这种格式的原始数据文件可以编程序计算出相临时间点之间的时间差，并可以计算出所有交易的平均时间差。也可以用 excel 打开这种格式的原始数据文件， 按空格分隔各列，读入 excel 后就可以使用 excel 提供的函数（入 SEC()函数从 hh:mm:ss 时间格式换算成秒）和公式计算时间差和平均值，以 及产生图表等等。事实证明 excel 的功能是十分强大和方便的。

二．误差的判断和排除。

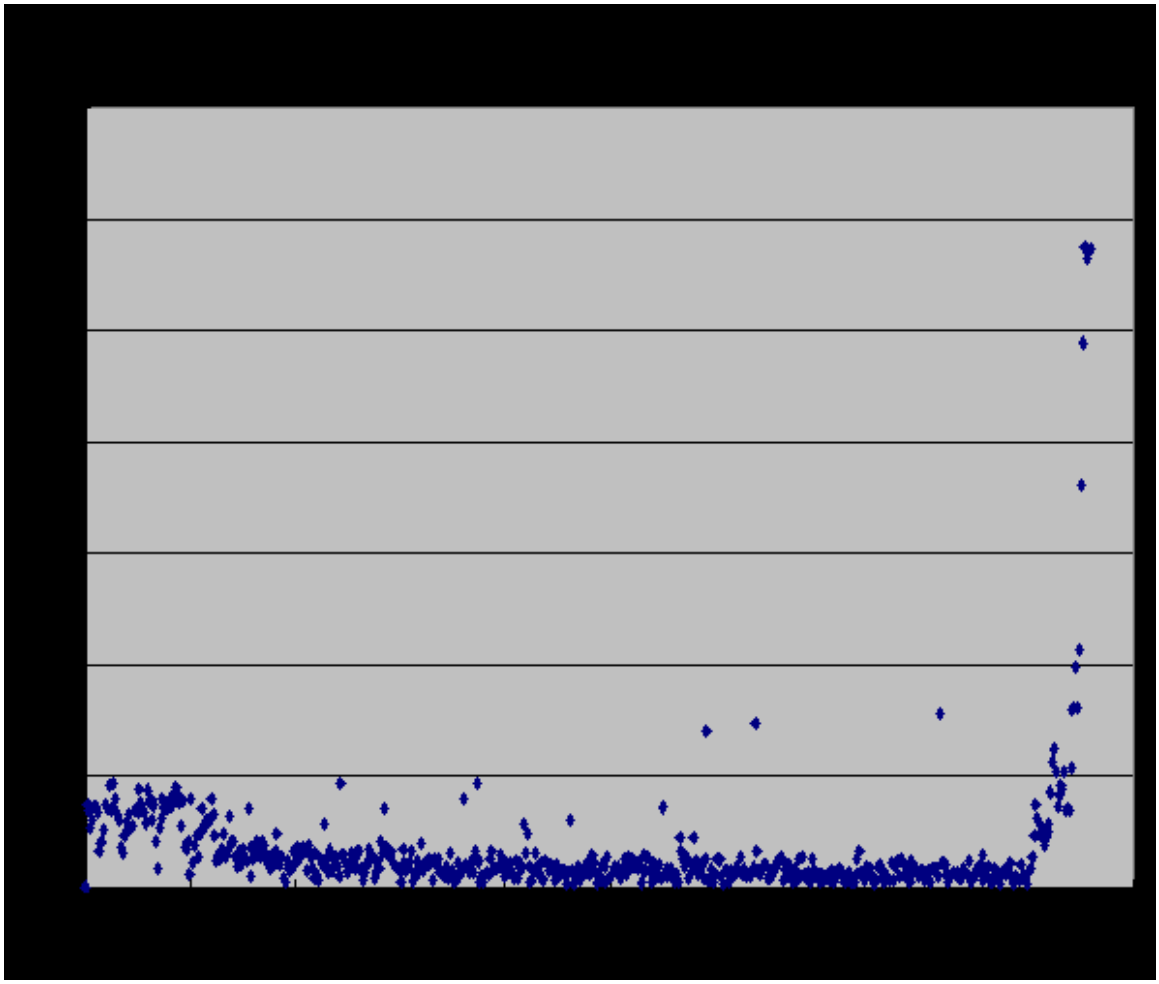
根据原始数据统计出的 3 种压力情况下 fcgi 各点时间如下。

		10_10_5_2(ms)	20_20_10_2(ms)	30_20_10_2(ms)
TPC（笔/秒）		2.16967	2.28571	2.21911
fcgi	receive from fcgi	343	931	1676
	tpcall	4096	7614	8067
	tpcommit	176	204	205
	total_fcgi	4615	8749	9731

比较 10_10_5_2 和 20_20_10_2，由于 20_20_10_2 在 SUN 主机上启动 fcgi 进程和 svr_cc 服务进程数都比 10_10_5_2 多，SUN 上的压力也较大，因此 receive from fcgi 的时间也较大是合理的。比较 20_20_10_2 和 30_20_10_2，2 在 SUN 主机上，压力没有变化，仅是有 HTTP 请求的排队（因为 browser 进程比 fcgi 进程多），SUN 上的压力应基本一样，而 receive from fcgi 的时间有较大的差别是不合理的。考虑到在测 30_20_10_2 并发时有失败业务，怀疑可能由于 browser 上加给 web Server 过大造成失败。这次性能测试主要测试系统正常（业务正常，压力情况是预计压力情况）时的系统情况，而在有业务失败时往往有些前端进程工作不正 常，不能对后台造成预期的压力，这时取平均值可能会造成较大误差。
拿出其中任一个 FCGI 进程的原始数据，比较其在 20_20_10_2 和 30_20_10_2 两种压力下的 receive from fcgi 的时间数据，并用 excel 产生图表，



入下图。



比较上面两个图表可以发现，20_20_10_2 时各时间点基本都平均分布在 1.5 秒之内，仅有极少数几个点在 1.5 秒之外，且最大不超过 4 秒，由此可以 认为对这些值取平均值的误差是可以接受的。而 30_20_10_2 时在测试开始阶段（800 笔之前）和结束阶段（4500 笔之后）的时间点明显高于中间阶 段的时间点，这应该是由于压力大时在测试开始阶段 30 个 browser 进程没有很快把压力压向 fcgi（压力小时也有这种情况，但时间会小的多），这样造 成 30 个 browser 进程也不是在相近时间内结束，在结束阶段只有少数 browser 进程仍没有完成，这时的系统压力变小，fcgi 进程等待 HTTP 请 求时间也变长。在 30_20_10_2 时这种非正常压力时间段很长并且数据差距很大，这时取全部时间段内的数值的平均值必然带来误差。从上图可以看到，应 该取 800 笔到 4500 之间系统稳定时的数据作为有效数据。注意其他环节的进程的时间统计也需要按这一笔数范围作为有效数据。经过修正后的全部数据见下 表。数据基本正常。

		10_10_5_2(ms)	20_20_10_2(ms)	30_20_10_2(ms)
TPC（笔/秒）		2.16967	2.28571	2.21911

browser		4609	8750	13519
fcgi	receive from fcgi	343	931	954
	tpcall	4096	7614	8575
	tpcommit	176	204	202
	total_fcgi	4615	8749	9731
svr_cc	receive from TE	4	16	18
	service_before_tpcall	1346	3401	4201
	tpcall	927	927	598
	service_after_tpcall	38	45	53
	total_svr_cc	2315	4389	4870
	waiting&receive from TE	289	267	555
	service	636	611	418

经验：有时取平均值会有较大的误差，尤其是测试不完全正常的情况。这时需要仔细分析原始数据，排除造成误差的数据，以系统稳定（正常）时的数据作为有效数据。这里 excel 图表功能是一个非常好用的工具。

三．数据分析。

从业务流程中可以想象，fcgi 端的 TE_tpcall()时间似乎应该大致等于或略大于（网络传输时间等等）svr_cc 上服务总的时间加上排队时间， 而排队时间应该这样线性计算：前端并发数/服务端并发数*服务端服务单笔总的平均时间。但上表的实际数据是 TE_tpcall()时间小于 svr_cc 上 服务总的时间！类似的，browser 端的时间和 fcgi 上 fcgi 进程处理时间也有这种现象，只是相差很小。

这是因为实际情况不是我们想象的那样！SUN 主机上即有客户端 cc1（包括 FCGI）节点也有服务端 cc2 节点的进程和 TE 核心运行。如果 cc1 上只有一 个 fcgi 进程（TE 客户端进程）串行发起业务，我们可以认为 fcgi 端的 TE_tpcall()时间似乎应该大致等于或略大于（网络传输时间等 等）svr_cc 上服务总的时间。而当 cc1 上有多个 fcgi 进程（TE 客户端进程）并发发起业务时，在处理一笔服务业务逻辑的时间段内，SUN 上的 CPU 时间还会分给其他客户进程以及 TE 核心（处理其他事务），这样在客户端并发情况下，每笔业务的服务端处理时间实际上包含一部分其他笔业务（客户进程 和 TE 核心）的处理时间里。而所有的业务都是如此，最终 TE_tpcall()平均时间小于 svr_cc 上服务总的平均时间。

在 cc1 进程和 cc2 进程的关系中，由于他们在一台主机上，并且 cc1 进程不在少数，因此上述两个时间差别也较大。而在 browser 和 cc1 的关系中， 可以排除客户

端进程的处理（因为在另外一台机器上）对服务端的影响，而只有 TE 核心并行处理多个交易对单笔服务处理时间的影响。在测试环境下，服务业务处 理时间是主要时间，TE 处理时间相对很小，因此上述两个时间差别也非常小。

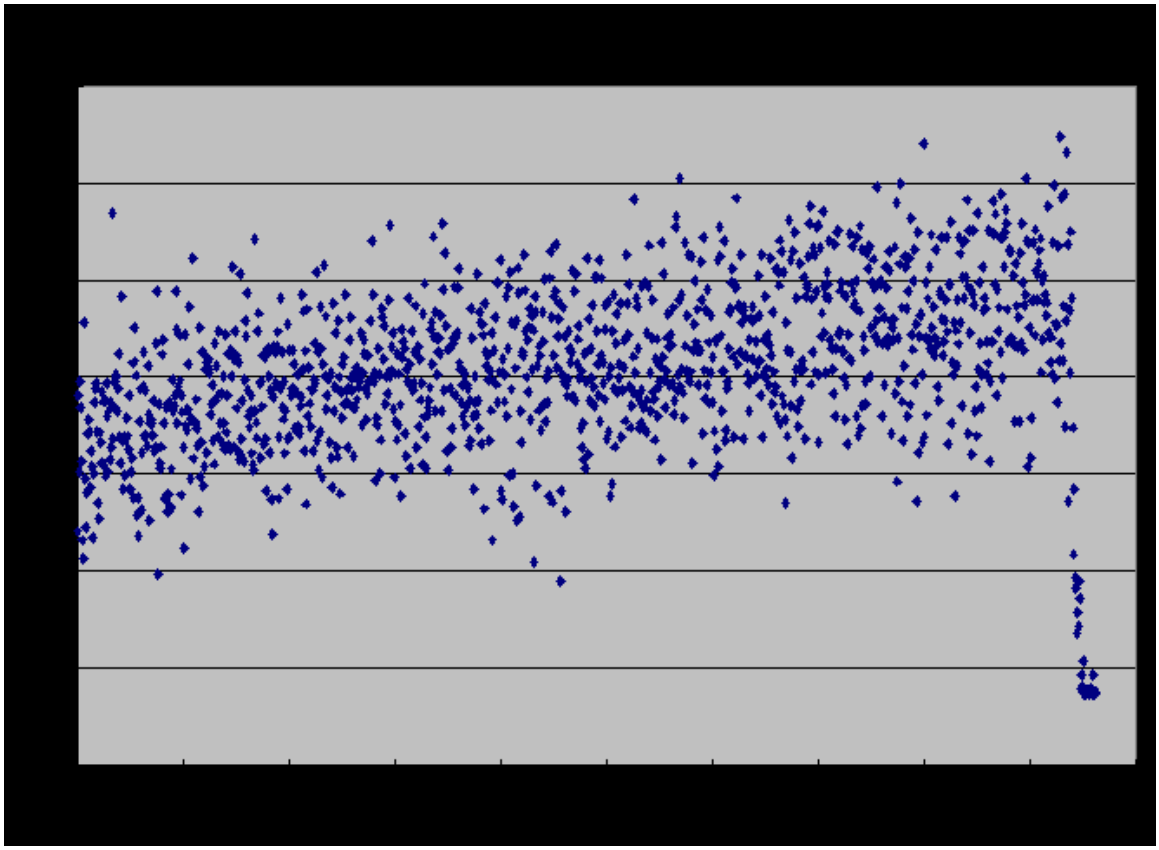
经验：在多前端并发情况下，前端等待服务端应答时间和服务端处理时间不是线性比例关系，会比预期线性比例关系算出的时间小。具体差别和测试环境配置有关系。客户端和服务端分开在不同的机器上会更接近线性比例关系。

四．平均值以外的东西。

对测试原始数据取平均值可以对整个系统的性能有一个了解，但有时只看平均值会遗漏一些同样有价值的东西。

取 30_20_10_2 测试中任一个 svr_cc 服务进程的原始数据，用 excel 读入，计算每笔 service_before_tpcall 时间（该服 务的主要业务逻辑时间，也是整个开户业务中最耗时间的环节），并制作 service_before_tpcall 时间随笔数变化的图表，我们可以看到服务 业务逻辑随数据库中记录数的变化规律，这显然是平均值无法体现的。

从图中可以看出，该业务逻辑时间的平均分布随笔数的增加而增加，也就是说，该业务逻辑时间和数据库中记录数有关（因为开户是插入操作，应该是该业务逻辑时间的平均分布随数据库记录数的增加而增加），在这种大型 OLTP 系统中，这种处理时间和数据库记录数有较大关系的现象是不好的。一般是因为数据库索引设置问题，需要调整数据库索引。



至于最后几十笔的时间急剧减小，是因为在最后几十笔时有些前端应用已经完成，对于后台的压力减小，服务处理时间自然减小，并且越到后来运行的前端应用越少，服务处理时间也越少。结合前面第二节的经验，在这个例子中，因为这些无效数据只占全部数据很少一部分，它们不会对平均值产生太大的误差。

经验：有时平均值数据不能反映系统全部特性。尤其对于系统关键环节，必要时还对原始数据做进一步统计分析。