性能测试

参考案例

参考案例

访问性能测试控制台

门户类网站性能测试分析及调优

1 背景

前段时间，性能测试团队经历了一个规模较大的门户网站的性能优化工作，该网站的开发和合作涉及多个组织和部门，而且网站的重要性不言而喻，同时上线时间非常紧迫，关注度也很高，所以对于整个团队的压力也非常大。
在此，把整个经历过程给大家分享一下，包括了主要包括了如何使用性能测试的压测工具，压测前的性能问题评估，以及压测执行后的性能问题分析、瓶颈定位。
该门户网站的服务器是放在华通和阿里云的平台上的，所以对华通和阿里共建的云平台安全及应急措施方面要求非常高，需要团队给予全力的保障和配合。
性能测试（Performance Testing）是集测试机管理、测试脚本管理、测试场景管理、测试任务管理、测试结果管理为一体的性能云测试平台，可以帮助您全方位的评估云上系统性能。

本次优化主要是使用了该测试平台服务对客户搭建在ECS上的服务器进行多种类型（性能测试、负载测试、压力测试、稳定性测试、混合场景测试、异常测试等）的性能压测、调试和分析，最终达到满足期望预估的性能目标值，且上线后在高峰期满足实际的性能和稳定要求。

2 术语定义

在介绍项目经历之前，再明确一下测试当中用到的专业指标术语定义，包括但不仅限于以下：

PV: 即PageView, 即页面浏览量或点击量，用户每次刷新即被计算一次。我们可以认为，用户的一次刷新，给服务器造成了一次请求。

UV: 即UniqueVisitor,访问您网站的一台电脑客户端为一个访客。00:00-24:00内相同的客户端只被计算一次。

TPS:TPS(Transaction Per Second)每秒钟系统能够处理的交易或事务的数量，它是衡量系统处理能力的重要指标。

响应时间: 响应时间是指从客户端发一个请求开始计时，到客户端接收到从服务器端返回的响应结果结束所经历的时间，响应时间由请求发送时间、网络传输时间和服务器处理时间三部分组成。

VU: Virtual user，模拟真实业务逻辑步骤的虚拟用户，虚拟用户模拟的操作步骤都被记录在虚拟用户脚本里。一般性能测试过程中，通俗称之为并发用户数。

TPS波动: 系统性能依赖于特定的硬件、软件代码、应用服务、网络资源等，所以在性能场景执行期间，TPS可能会表现为稳定，或者波动，抑或遵循一定的上升或下降趋势。我们用TPS波动系数来记录这个指标值。

CPU: CPU资源是指性能测试场景运行的这个时间段内，应用服务系统的CPU资源占用率。CPU资源是判断系统处理能力以及应用运行是否稳定的重要参数。

Load: 系统正在干活的多少的度量，队列长度。系统平均负载，被定义为在特定时间间隔
（1m，5m，15m）内运行队列中的平均进程数
。

I/O: I/O可分为磁盘IO和网卡IO。

JVM: 即java虚拟机，它拥有自己的处理器、堆栈、寄存器等，还有自己相应的指令系统。Java应用运行在JVM上面。

GC: GC是一种自动内存管理程序，它主要的职责是分配内存、保证被引用的对象始终在内存中、把不被应用的对象从内存中释放。FGC会引起JVM挂起。

网速: 网络中的数据传输速率，一般以Byte/s为单位。通过ping延时来反映网速。

流量: 性能测试中，一般指单位时间内流经网卡的总流量。分为inbound和outbound，一般以KB为单位。

3 评估

本次性能测试过程的参与人包括了阿里云应急保障小组等多部门人员，网站为外部供应商开发，阿里云提供云主机和技术支持。
该网站之前前期也由其他部门做了验收工作，进行了完整的性能测试，报告显示，性能较差，第一次测试，网站并发数没有超过35个，第二次测试，网站上做了优化后，静态页面缩小后，并发用户数100内 5s ，200内90%响应在15s以上，随着并发用户数的增加，页面响应最高可到20多秒，而且访问明显感觉较慢，所以联系了阿里云的技术支持，希望能够帮助诊断性能问题，给出优化建议。

测试业务	并发用户数	平均响应时间(秒)	90%响应时间（秒）
首页浏览	100	12.082	15.289
	200	23.949	29.092
分页浏览	100	8.973	12.343
	200	18.846	24.106

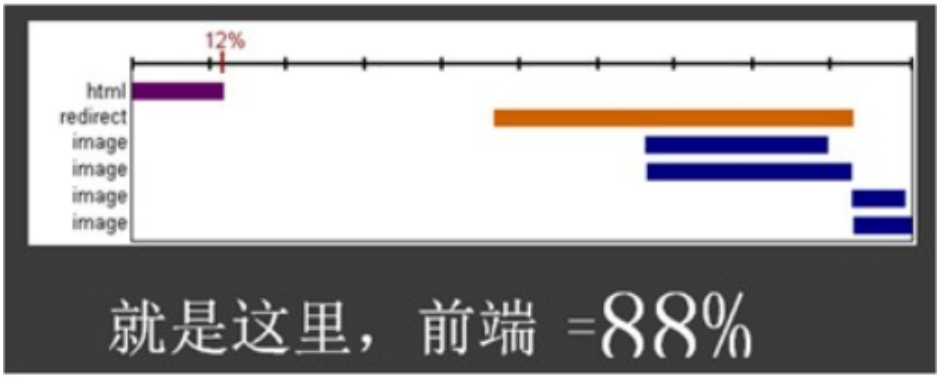
经过会议讨论后，评估出最终的测试目标：带页面的所有静态资源一起，响应时间必须小于5秒，同时并发访问用户数最低500，TPS根据实际的结果来得出。

4 性能测试目标

- 并发用户数：>=500
- 业务响应时间：<5秒

5 分析

通过性能测试前端分析工具(未开放)分析，页面的响应时间88%左右都是消耗在前端资源加载上，服务器端消耗只占到了页面响应的12%左右;



一个网站的响应一般由四部分时间组成，前端、网络、服务器和数据库，前端主要是减小页面大小，减小页面请求数，优化页面js等，网络主要是使用CDN，优化连接数等，服务器主要是优化Apache，优化Tomcat，优化java代码等，数据库是优化sql语句，优化索引，优化数据存储等。<rb>

6 测试和优化

6.1 页面前端分析及优化

我们对页面的优化仍然从前端开始，首先通过性能测试的前端测试工具（未开放）进行扫描，我们发现以下问题并优化：

- Js较大，无压缩，同时存在重复请求，最多一个js加载4次，已做压缩和减少。
- Js位置不合理，阻碍页面加载。
- 外部css 考虑本地实现，减少调用
- Banner 背景图片较多，无压缩，建议合并
- 页面1的后台.do有4个，减少为3个

- 页面2的后台do有 2个，减少为1个
- 存在加载失败链接，404失败，同时次数非常多，更换为cnzz
- 页面加载外部资源失败 (qq等)，且不稳定
- 分享功能比较慢
- 外部资源建议异步实现，目前全部是jquery渲染，iframe嵌套，时间资源限制，后期优化
- 尽量减少或者不使用iframe
- 页面请求数太多，主要是js和css重复加载问题和图片较小导致的。 经过以上修改及配置服务器静态资源缓存后，性能提高25%，首页响应从1.5秒提高到1.1秒，并且前端优化持续进行。

6.2 服务端优化

一般核心页面都要求在300毫秒以下，非核心页面要求在500毫秒以下，同时重点关注并发时的负载和稳定，服务器端代码和响应的快速稳定是整个页面性能的重点。

6.2.1 脚本编写及场景构造

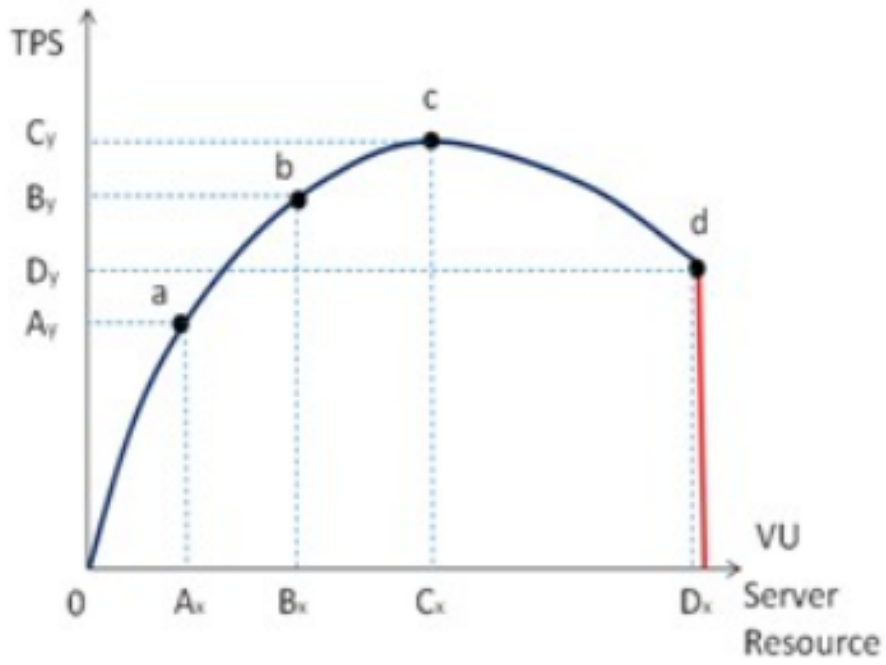
根据前期需求评估的内容，客户是一个门户网站，主要由不同功能页面组成的，各个功能页面当中又包含了静态内容和异步动态请求，所以，性能测试的脚本的编写主要涵盖了各页面的请求和相关静态资源的请求，这里存在一个串行和并行的概念：

串行：请求的页面和页面当中的静态资源、异步动态请求组成一个同步请求，每一个内容都作为一个事务（也可以共同组成一个事务，分开事务的好处是可以统计各部分的响应时间），这样压测任务执行时，线程就会根据事物的顺序分布调用执行，相当于一个页面的顺序加载，弊端是无法模拟实际IE的小范围并发，但这样测试的结果是最严格的。

并行：各个页面之前可以使用不同的任务，采用并行的混合场景执行，同时设置一定的比例（并发用户数），保证服务器承受的压力与实际用户访问相似。

场景并发用户数：经常会遇到“设置多大并发用户数合适？”的问题，因为在没有任何思考时间的时候，我们有一个简单的公式：
VU（并发压测用户数） = TPS（每秒执行事务数） × RT（响应时间）

所以，在寻找合适的并发用户数上，建议使用性能测试的“梯度模式”，逐渐增加并发用户数，这个时候压力也会越来越大，当TPS的增长率小于响应时间的增长率时，这就是性能的拐点，也就是最合理的并发用户数；当TPS不再增长或者下降时，这个时候的压力就是最大的压力，所使用的并发用户数就是最大的并发用户数。如果此时的TPS不满足你的要求，那么就需要寻找瓶颈来优化。如下图演示的一个性能曲线：



- a点：性能期望值
- b点：高于期望，系统安全
- c点：高于期望，拐点
- d点：超过负载，系统崩溃

注意：使用外网地址压测可能会造成瞬时流量较大，造成流量计费，从而产生费用，建议可以使用内网地址压测，避免损失。

6.2.2 第一阶段

在按照客户提供的4个URL请求构造场景压测以后，同时根据实际情况和客户要求，使用性能测试，构造相应的场景和脚本后，模拟用户实际访问情况，并且脚本中加上了img、js、css等各一个的最大静态资源：

- **条件：**5台ECS机器，200并发用户数，一个后台页面加3个静态页面（共150K）
- **结果：**静态4000TPS，动态1500TPS，服务器资源：CPU 98%

分析和优化：

性能指标：

场景名	并发用户数	事务名	性能指标		事务统计		
			TPS	RT(ms)	执行事务数	失败事务数	失败率
	200.0	hz	184.696	19.998	38786	0	0%
		css	1853.467	14.219	389228	0	0%
		img	1853.524	40.147	389240	0	0%
		jquery	1853.972	0.003	389334	0	0%
		col238	187.162	17.489	39304	0	0%
		col41792	184.696	17.155	38786	0	0%
		col41791	184.115	17.103	38664	0	0%
		index	369.596	24.354	77615	0	0%
		jiemu	371.677	105.105	78052	0	0%
		user	372.034	66.924	78127	0	0%

服务器资源消耗较高，超过75%，存在瓶颈，分析平台显示：

分析发现原来是apache到tomcat的连接等待导致，现象是100个并发压测，就有100个tomcat的java线程，而且全部是runnable的状态，轮询很耗时间。

同时发现用户使用的是http协议，非ajp协议，不过这个改动较大，需要使用mod_jk模块，时间原因，暂缓。

解决方法：修改了Apache和tomcat的连接协议 为nio协议，同时去除ssl 协议。Tomcat连接数据库池由30初始调整为300，减少开销

```
&lt;Connectorport="8080" protocol="org.apache.coyote.http11.Http11NioProtocol"
           connectionTimeout="20000"
           redirectPort="8443" /&gt;
```

性能对比：

- 再进行1轮压测含动态含静态文件，TPS能够从1w达到2.7W，性能提高将近3倍，并且tomcat的线程从原来的200跑满，降到100附近并且线程没有持续跑满，2.6W TPS时候CPU在80%附近

  发现机器的核数都是2核，8G内存，对于CPU达到98%的情况，CPU是瓶颈，而对于应用来说比较浪费，所以将2核统一升级为4核。

  扩展机器资源，从目前的4台扩到6台，同时准备4台备份，以应对访问量较大的情况。

思考和风险：

- **异步请求处理:**客户所提供的url都是html静态，虽然页面当中含动态数据，但分析后发现动态数据都是通过jquery执行然后iframe嵌套的，所以不会随着html文件的加载而自动加载，需要分析所有的动态页面，同时压测，这是页面存在异步请求需要关注的地方。
- **Iframe:**Iframe嵌套页面的方式优点是静态资源调用方便、页面和程序可以分离，但是它的缺点也显而易见，包括样式、脚本额外注入，增加请求等等；还有搜索引擎搜素不到内容；iframe创建比其他元素慢1~2个数量级；资源重复加载；iframe会阻塞页面加载，阻塞onload事件；占用主页连接池；html5不再支持。所以建议尽量不要使用或者少使用。
- **[font='Times New Roman']:** 脚本录制和模拟实际用户访问。当用户的图片、javascript、CSS等静态资源和后端代码在同一台服务器上时，需要模拟用户的实际访问请求，压测脚本涵盖所有链接和资源。那么使用脚本录制功能就可以采集更全更完整的脚本。

6.2.3 第二阶段

找到几个页面的所有动态资源后，整合成为一个事务，串行访问，同时并发压测，从而对纯服务器端进行压测

，测试结果如下图：

性能指标：

场景名	并发用户数	事务名	性能指标		事务统计		
			TPS	RT(ms)	执行事务数	失败事务数	失败率
		hdjl	11.84	76.867	1776	0	0%
		ygzw	11.847	54.924	1777	0	0%
		bmfwDetail	11.847	34.136	1777	0	0%
		bmfw	11.854	24.27	1778	0	0%
	100.0	xzsp	0	0	1778	1778	100%
		frbsDetail	11.854	5520.955	1778	0	0%
		frbs	11.482	109.184	1837	0	0%
		grbsDetail	11.494	2212.604	1839	0	0%
		grbs	11.713	104.932	1874	0	0%
		index	11.725	47.127	1876	0	0%

性能分析：页面一和页面二的响应时间分别达到了5秒和2秒，性能较差，整体TPS只有11

- 性能分析：
  分析发现响应时间高的原因主要在RDS数据库上，数据库此时的CPU已经达到100%，处理较慢，怀疑跟sql有关，分析慢sql。
- 优化方法：
  数据库第一批优化完毕，优化了6条sql语句之前5s左右，优化后在150ms左右，数据库的QPS 从1k上升到6k。
- RDS优化内容包括：
  优化点主要是调整慢sql的索引，部分sql需要调整表结构和sql写法，需要应用配合才能完成优化，优化前QPS在1000左右，优化后QPS到达6000

前端响应时间从5秒降低到150毫秒，前台TPS由150提升到1500.总的TPS可以达到2000。

6.2.4 第三阶段

通过性能测试模拟用户实际访问情况，包括所有静态资源，评估出当响应时间小于5秒的时候，最大支撑的并发用户数。

- 测试结果：

场景名	并发用户数	事务名	性能指标		事务统计		
			TPS	RT(ms)	执行事务数	失败事务数	失败率
		page6	556.452	426.506	155456	0	0%
		page5	556.442	4.99	155452	0	0%
		page4	556.421	5.457	155451	0	0%
	3000.0	page3	556.421	5.421	155450	0	0%
		page2	556.449	3724.746	155458	0	0%
		page1	556.132	1229.787	155398	0	0%

可以看到，所有的事务RT加起来小于5秒的情况下，并发用户数可以达到3000(6个事务，6个脚本，每个脚本500)，远远满足项目500个并发用户的目标。

- 总结：

压测项目	优化前			优化后		
页面一	并发用户数:100	平均响应时间:5s	CPU负载:负载上不去	并发用户数:3000	平均响应时间:5s	CPU负载:99%
动态页面	TPS	平均响应时间	CPU负载	TPS:1500	平均响应时:241ms	CPU负载:90%
RDS	QPS:1000			QPS:7000		

6.2.5 第四阶段

评估其他非主站应用的性能以及含静态页面的其他5个页面内容，包括：

- 搜索压测
- 操作压测
- 登录压测
- 证书登入
- 针对5个常用场景进行混合压测

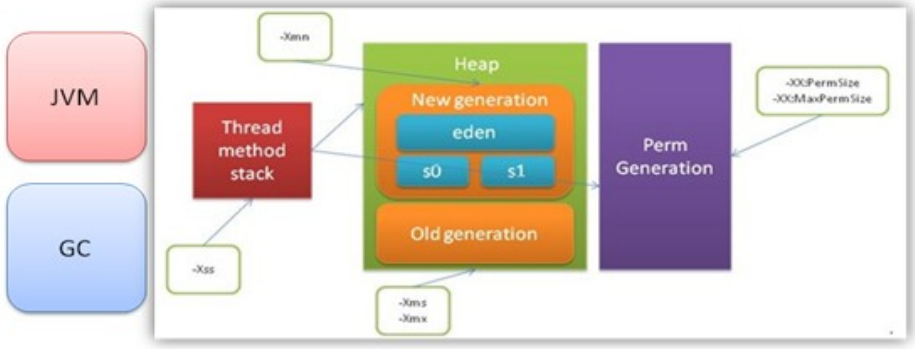
测试结果:

并发用户数	性能指标		负载				页面资源情况	
	TPS	响应时间（S）	网络(Mbps)	PPS(K)	QPS(K)	ECS CPU	请求数/页	页面Size（KB）
500	250	1.6	418	44	11	90%	75	500
	680	0.7	232	27.7	8.2	94%	12	83.5
	350	1.4	121	14	4	60%	24	125
	430	1	298	36	11	95%	35	350
	500	0.85	332	41	19.6	90%	65	300

发现的风险和问题：

- 测试发现，流量存在非常明显的波动，不经过某模块就无此问题，发现有大量的reset连接，会诊后总结：端口复用导致的问题。FULL NAT模式和LVS存在兼容性问题。最终结果: 由于存在兼容性问题，影响到网站的稳定性和性能，暂时加载该模块，待问题解决后再加。先使用另外一个模块代替
- .凌晨2点，针对单点用户登入进行了压测，发现100并发，该业务接口已宕机，分析结果:Cache缓存设置太小，1G内存容量导致内存溢出，已建议修改为4G。使用http协议性能不佳，早上4点30进行少量代码优化后，业务直接不可用，环境出现宕机无法修复，我们快速进行快照恢复，5分钟内恢复3台业务机，云产品的优势尽显。用户log日志撑满系统盘，并且一直不知这台云主机还有数据盘，产品上我们要做反思。帮助用户已进行挂盘及日志迁移至数据盘，减少单盘的IO压力。
- Web服务器数据同步，发现服务器io和cpu压力过大。加入inotify机制，由文件增量同步，变更为文件系统的变化通知机制。将冷备及4台备用web机器使用该方式同步，目前，查看内容分发主机IO和CPU使用率已回复正常范围同步推送时间，根据服务器的负载，进行调整同步时间。今天已修改为2分钟。 由于备份量大，晚上进行全量同步。新增4台备用机，已关闭apache 端口 自动从slb去除，作为冷备
- 由于目前单点用户登入入口存在架构单点宕机风险，进行登入和未登入风险验证，确认，如用户已登入后，登入业务系统出现宕机，进行简单的页面点击切换，不受影响

- 内存优化



按照JVM内存管理模式，调整系统启动参数，如果一台ECS部署一台服务器，建议不要选择默认的JVM配置，应该设置内存为物理内存的一半，同时设置相应的YTC和FGC策略，观察Old区变化，避免大量Full GC，建议Full GC频率大于1小时，同时GC时间小于1秒钟。

6.3 架构优化

- 单点登录服务修改为SLB
- 检索 修改为 SLB
- 内容管理云平台云服务器实现行文件差异同步，同时冷备
- 新增4台web机器

7 总结

压测项目	优化前			优化后		
页面一	并发用户数:100	平均响应时间:5s	CPU负载:负载上不去	并发用户数:3000	平均响应时间:5s	CPU负载:99%
动态页面	TPS	平均响应时间	CPU 负载	TPS:1500	平均响应时:241ms	CPU负载:90%
RDS	QPS:1000			QPS:7000		

备注：服务器的CPU达到100% 这其实是一个好的现象，说明我们的逻辑全部已经走到了资源消耗上，而不是由外部其他逻辑限制或blocked，这种现象带来的好处就是，首先我们可以集中精力从减少代码的资源消耗上解决问题，带来性能的改善，其次，实在无法优化我们也可以增加机器台数，横向扩展来让性能成倍的提高，这也是用成本换性能的方法。当然，前提是架构上支持负载均衡的分布式架构。
总的来说就是，这种情况要不选择从纵向优化，或者选择从横向扩展，都可以增加。

8 遗留问题

- 时间原因，测试页面有限，有些页面没有测试覆盖到，比如后台页面。
- 登录系统存在内存风险，由于用户的缓存信息仍然存在单点问题，所以风险较大，同时系统压力不满

足要求，并发较高存在crash风险，未来得及发现瓶颈所在。彻底解决必须使用OCS等缓存系统改造，同时优化数据库。
- Iframe框架造成用户体验不好，需要改掉，换成异步js接口方式。
- 后台同步系统，对于资源消耗影响较大，需要持续优化。
- 平台应该提供开发和测试进行自主压测、分析、评估，同时提供统一的测试报告，保证各部分模块的性能，整合起来才能保障整个门户网站的性能。
- CPU优化还需要继续分析跟进，目前只是增加机器资源降低风险，成本较高。
- 上线发布应该具备统一的回归验证机制，并且日常需要持续优化，避免由于后期代码的改动导致性能下降。

访问性能测试控制台

访问性能测试控制台

大规模分布式压测

1 背景

需要临时扩容他们的机器来支持100W的QPS，每秒100W的请求，听起来还是挺恐怖的。什么概念呢，2013年双12的大秒系统的峰值QPS也就在42万多。从这样的数据来看，这个客户的需求高的离谱。但是既然用户有这个需求，我们还是需要满足客户的期望。

2 问题及挑战

遇到问题主要有：

- 100万QPS
- 被测系统如何搭建，需要多少台机器
- 选择何种压测工具，百万级QPS考验，压力机需要多少台机器

遇到的挑战主要有：

- 项目实施的时间只有5天，压力非常大
- 被测系统是否能承受如此大的压力
- 压力机自身是否能经受如此大的压力，是否稳定
- 短时间内如何快速部署被测系统及压力机环境
- 性能瓶颈在哪（程序、OS/硬件、网络设备等）

3 解决方案及评估

经过多次会议，确定解决方案是，采用阿里云环境自动化运维及弹性扩容来搭建环境，压测工具采用分布式压

测。
通过评估机器数量如下：

- 被测系统环境：2台SLB,300台ECS(4核CPU 8G内存)，批量部署应用
- 压力机环境：300台ECS（4核CPU 8G内存），批量部署性能测试

整个团队包括客户、SLB和ECS机器维护人员、环境部署人员以及性能测试团队充分密切配合。

4 目标

QPS：100万，稳定运行1分钟左右。

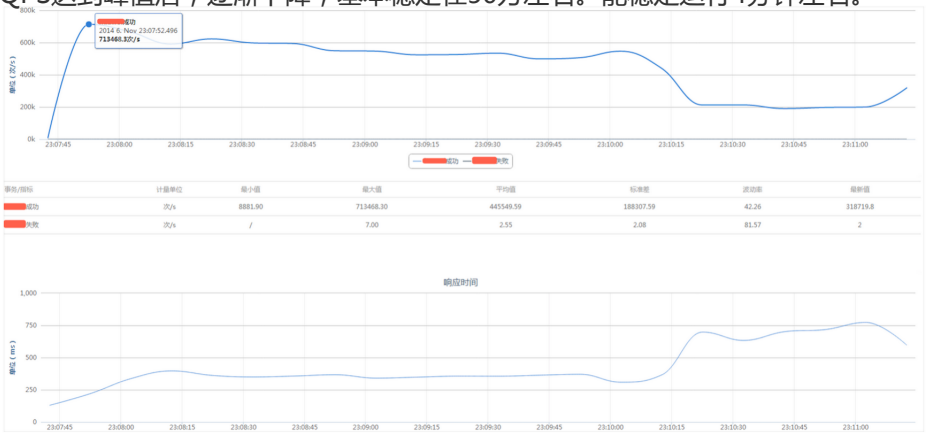
5 典型业务

秒杀活动，一个比较复杂的带Header, Body, Cookie 的http 请求。

6 测试结果

6.1 结果

- QPS峰值最高达到71.5W，后端ECS CPU利用率最高75%，网络峰值流量达到25Gb。
- QPS达到峰值后，逐渐下降，基本稳定在50万左右。能稳定运行4分钟左右。



6.2 分析

QPS下降的原因经过各技术专家诊断一致认为是SLB丢包导致，SLB压力已到极限，因此建议需要配置3台SLB，每台SLB挂100台ECS，才有可能满足100万QPS 。

6.3 结论

经过与客户会讨，峰值71.5万QPS，稳定运行4分钟50万QPS，能满足目前现在的业务需求。 如果需要支持

100万QPS，需要扩容SLB，至少是3台SLB以上。

7 总结

7.1 分布式压测

7.1.1 稳定性

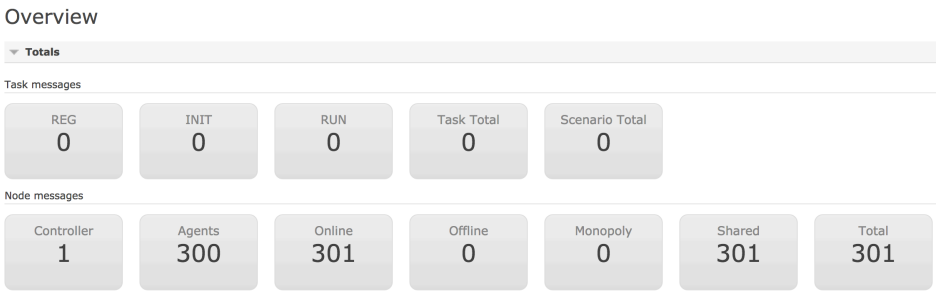
在测试过程中，性能测试经受住了大规模压力的考验，并且从未出现过异常问题，由此可知，性能测试产品非常稳定。

7.1.2 百万级QPS支持

在测试的过程中，性能测试能支撑百万级QPS的压力发起，这是目前其他压测工具所不能支持的。

7.1.3 资源消耗少

虽然性能测试压测机申请了300台ECS机器，但在测试过程中，消耗的机器资源非常少，CPU利用率不到0.1%，并且每台机器负载均衡，实际上100台ECS就足够了。



7.2 环境搭建

在调研阶段，性能测试团队就大规模压力发起进行了充分的调研，并且通过测试验证单台机器能发起的压力以及弹性扩容，预估出需要的机器数量，才能保证项目的顺利进行。

另外阿里云批量自动化环境搭建节省了环境的部署时间，在1天内完成所有工作。

7.3 团队合作

这次大规模压力压测在5天内顺利完成，离不开整个团队所有人员密切配合，重点关注，才能让如此大的项目在短时间内成功实施。因此团队合作在项目实施的过程中有起着举足轻重的作用。

访问性能测试控制台

访问性能测试控制台

移动APP项目实施案例分析及调优

1 背景

随着客户业务发展，目前系统架构已不能满足业务发展需要，因此急需将服务器托管到阿里云上，并进行扩容；迁移到阿里云上以后，系统资源消耗是否比目前线上环境结果要好。因此在上线前需要进行性能测试，测试是否满足各项性能指标。

2 测试目标

本次测试目标如下：

- **容量测试**：核心业务(核心业务1+核心业务2)+非核心业务基线（非核心业务1+非核心业务2+非核心业务3+非核心业务4+非核心业务5+非核心业务6）混合交易容量
- **稳定性测试**：混合交易稳定性
- **突变测试**：非核心业务突变3倍，对核心业务的影响
- **对比测试**：和线上同等压力下，线上和线下资源消耗和响应时间对比。
- **恢复性测试**：模拟网络攻击

3 架构

系统架构主要有如下服务器：

- **HTTP服务器**：核心业务1和核心业务2业务
- **TCP服务器**：核心业务使用人员终端心跳业务
- **MongoDB服务器**：非结构化数据库存储
- **Redis服务器**：信息推送
- **MySQL服务器**：结构化数据库存储

4 测试指标

- **容量测试**：核心业务1 TPS>=600笔/秒,核心业务2 TPS>=1200笔/秒
- **稳定性测试**：至少在核心业务1 TPS等于300笔/秒和核心业务2 TPS等于600笔/秒能稳定运行8小时
- **突变测试**：非核心业务突变3倍，基本对核心业务无影响
- **线上线下资源消耗对比测试**：在跟线上核心业务1 TPS等于150笔/秒和核心业务2 TPS等于120笔/秒同等压力下，测试环境的MonogoDB和Redis CPU Load小于0.5%,磁盘利用率小于0.1%
- **线上线下存储访问时间对比测试**：在核心业务1 TPS等于200笔/秒和核心业务2 TPS等于400笔/秒的情况下，应用观察到的存储访问平均耗时不超过4ms，最大耗时不超过100ms。
- **恢复性测试**：系统能恢复，TPS无变化

5 业务模型

5.1 分析

通过生产上高峰业务量分析得出，核心业务1和核心业务2除了双12外，比例占比1：1.5左右，通过系统整个趋势观察，发现核心业务2业务量有明显增长趋势，因此核心业务1和核心业务2的占比为1：2。高峰时候核心业务总的TPS只有50~200笔 / 秒。

核心业务量：

时间点	业务	合计	占比	平均	占比	最大值	占比	最大值 TPS
2014/12/12 17：00~18：00	核心业务 1	351348	0.601096299	5855	0.605043	6403	0.604513	106.716667
	核心业务 2	233164	0.398903701	3822	0.394957	4189	0.395487	69.8166667
	合计	584512		9677		10592		
时间点	业务	合计	占比	平均	占比	最大值	占比	
2015/01/04 07：00~08：00	核心业务 1	143323	0.245201125	2349	0.242741	3425	0.430385	57.0833333
	核心业务 2	245052	0.419242034	4084	0.422032	4533	0.569615	75.55
	合计	388375		6433		7958		
时间点	业务	合计	占比	平均	占比	最大值	占比	
2015/01/06 07：00~08：00	核心业务 1	126989	0.217256446	2116	0.218663	3123	0.360499	52.05
	核心业务 2	257128	0.439902004	4215	0.435569	5540	0.639501	92.3333333
	合计	384117		6331		8663		
时间点	业务	合计	占比	平均	占比	最大值	占比	
2014/11/11 17：00~18：00	核心业务 1	81702	0.13977814	1361	0.140643	1632	0.400294	27.2
	核心业务 2	111120	0.190107303	1852	0.191382	2445	0.599706	40.75
	合计	192822		3213		4077		

非核心业务量：

非核心业务1+非核心业务2+非核心业务3+非核心业务4+非核心业务5+非核心业务6

编号	业务	TPS	占比
1	非核心业务1	400	16.4%
2	非核心业务2	500	20.5%
3	非核心业务3	833	34.2%
4	非核心业务4	210	8.6%
5	非核心业务5	300	12.3%
6	非核心业务6	190	8%
合计		2433	100%

5.2 模型

5.2.1 模型1

编号	业务类型	业务	占比	备注
1	核心业务	核心业务 1	33.3%	采用梯度施压测试，测出容量
2		核心业务 2	66.7%	
3	非核心业务	非核心业务 1	16.4%	非核心基线，总的 TPS 为 2433 笔/秒，按照占比进行分配
4		非核心业务 2	20.5%	
5		非核心业务 3	34.2%	
6		非核心业务 4	8.6%	
7		非核心业务 5	12.3%	
8		非核心业务 6	8%	

此模型用于容量测试、稳定性测试和恢复性测试。

5.2.2 模型2

编号	业务类型	业务	占比	备注
1	核心业务	核心业务 1	33.3%	按照测试出来的容量的 50%压力运行
2		核心业务 2	66.7%	
3	非核心业务	非核心业务 1	16.4%	非核心基线突变 3 倍，总的 TPS 为 7299 笔/秒，按照占比进行分配
4		非核心业务 2	20.5%	
5		非核心业务 3	34.2%	
6		非核心业务 4	8.6%	
7		非核心业务 5	12.3%	
8		非核心业务 6	8%	

此模型用于突变测试。

5.2.3 模型3

编号	业务类型	业务	占比	备注
1	核心业务	核心业务 1	55.5%	按照核心业务 1 150TPS 和核心业务 2 120TPS 情况，资源消耗对比
2		核心业务 2	44.5%	
3	非核心业务	非核心业务 1	16.4%	非核心基线，总的 TPS 为 2433 笔/秒，按照占比进行分配
4		非核心业务 2	20.5%	
5		非核心业务 3	34.2%	
6		非核心业务 4	8.6%	
7		非核心业务 5	12.3%	
8		非核心业务 6	8%	

此模型用于线上线下资源消耗对比测试。

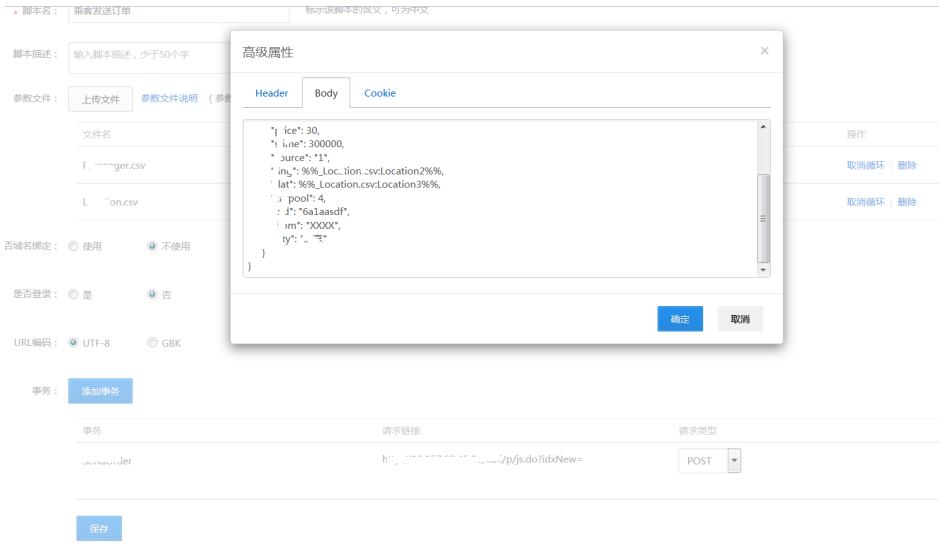
5.2.4 模型4

编号	业务类型	业务	占比	备注
1	核心业务	核心业务 1	33.3%	总的 TPS 为 600 笔/秒，方法耗时
2		核心业务 2	66.7%	
3	非核心业务	非核心业务 1	16.4%	非核心基线，总的 TPS 为 2433 笔/秒，按照占比进行分配
4		非核心业务 2	20.5%	
5		非核心业务 3	34.2%	
6		非核心业务 4	8.6%	
7		非核心业务 5	12.3%	
8		非核心业务 6	8%	

此模型用于线上线下存储访问时间对比测试

6 脚本设计

经过调研，发送后台的业务均是URL+自定义Body方式，因此在性能测试里面，新增一个脚本，上传参数化文件，定义事务，设置连接和Body就行了，注意尽可能多的进行参数化。



7 测试结果

7.1 容量测试

7.1.1 测试场景

按照模型 1 ，设置用户数比例和步调时间(保持业务占比，不偏模型)，运行20分钟，进行负载测试。

7.1.2 测试结果及分析

- 第一轮测试
  按照核心业务1 1000笔/秒和核心业务2 2000笔/秒目标发起压力，发现不能达到目标，TPS曲线不稳定，运行到 1 分钟的时候，下降非常厉害，抖动也非常厉害，通过监控，发现FULL GC非常频繁，达到 1 秒 1 次，经过与架构师沟通，这是由于实现机制导致的，核心业务1的机制是将内容放到队列里面，队列长度是2147483647,后台只有64个线程（不能修改）在消化，消费者（消化）处理速度的比生产者（核心业务1）慢，导致队列长度越来越大，内存很快被消化完了，导致FULL GC频繁，这属于架构问题，不能进行修改。

核心业务2：

第二轮测试

按照核心业务1 600笔/秒和核心业务2 1200笔/秒发起压力，运行20分钟，TPS基本保持稳定，通过监控发现，order应用连接MongoDB连接数报已满的异常错误、logserver IO过高、MongoDB locked DB值高于75%。 按照核心业务1 800笔/秒和核心业务2 1600笔/秒目标发起压力，不能达到此目标，TPS曲线非常不稳定。

第三轮测试

mongoDB 只有表锁没有行锁，导致locked值非常高，这个是产品问题，没办法进行调优。

将order应用MongoDB连接数从250调到1000;logserver 磁盘换成效率更高SSD磁盘；重新按照核心业务1 800笔/秒和核心业务2 1600笔/秒目标发起压力，运行20分钟，TPS曲线基本稳定。

核心业务1：

核心业务2：

7.1.3 测试结论

系统的容量为核心业务1 800笔/秒和核心业务2 1600笔/秒，满足核心业务1 600笔/秒和核心业务2 1200笔/秒目标要求。

7.2 线上线下资源消耗对比测试

7.2.1 测试场景

按照模型3发起压力，在核心业务1 150TPS和核心业务2 120TPS压力情况下，运行20分钟，资源消耗对比。

7.2.2 测试结果及分析

MongoDB 和 Redis CPU Load均小于0.5, CPU 利用率均小于10%,磁盘利用率均小于0.1%, 这些指标结果比线上资源消耗结果略好。

7.2.3 测试结论

在跟线上同等压力的情况下，阿里云环境各项指标结果略好于目前线上环境资源消耗。

7.3 线上线下存储访问时间对比测试

7.3.1 测试场景

按照模型 4 发起压力，在核心业务1 200笔/秒和核心业务2 400笔/秒的压力下，运行20分钟，观察存储访问的时间。

7.3.2 测试结果及分析

在xflush上面观察到的存储耗时值小于3ms，最大值不超过100ms

7.3.3 测试结论

满足目标平均耗时不超过4ms，最大耗时不超过100ms的需求。

7.4 突变测试

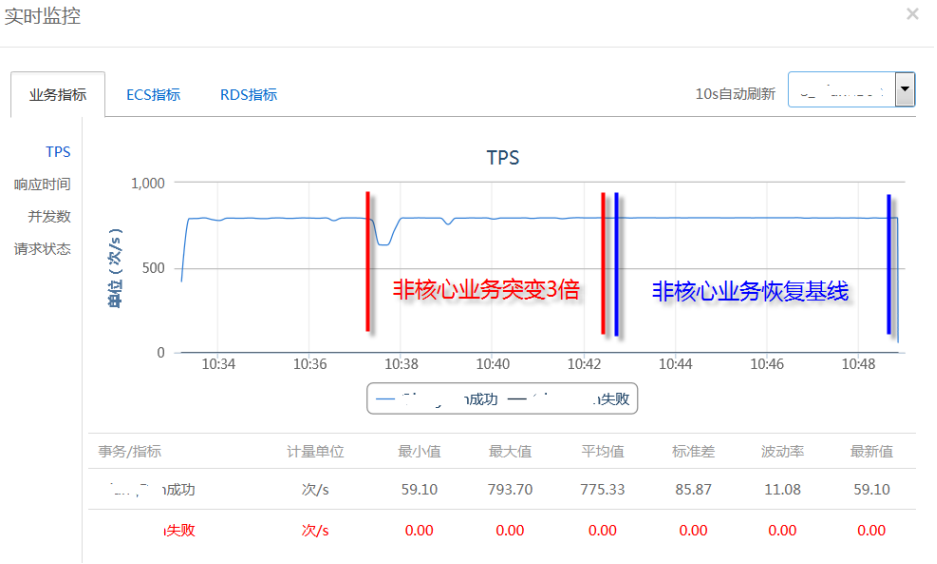
7.4.1 测试场景

按照模型 2 ，在核心业务1 TPS 400笔/秒和核心业务2 TPS 800笔/秒的情况下，平稳运行 5 分钟后，将非核心业务按照基线的 3 倍进行突变，运行 5 分钟，观察核心业务TPS曲线的变化，然后将非核心业务恢复到基线，观察核心业务TPS曲线的变化。

7.4.2 测试结果及分析

核心业务1：

核心业务2：



从图中可以看出，当非核心业务突变3倍以后，对核心业务1和核心业务2有轻微的影响（核心业务1和核心业务2 TPS下降），但马上能恢复，突变的整个过程对核心业务基本无影响。

7.4.3 测试结论

非核心业务突变 3 倍对核心业务基本无影响，满足目标要求。

7.5 恢复性测试

7.5.1 测试场景

按照模型 1 ，在核心业务1 800笔/秒和核心业务2 1600笔/秒的压力下，平稳运行 5 分钟后，断开所有mysql服务网络5秒，观察核心业务TPS曲线变化，然后恢复mysql网络，观察核心业务TPS曲线变化，接着断开所有

MongoDB服务网络5秒，观察核心业务TPS曲线变化，然后恢复所有MongoDB服务网络，观察核心业务TPS曲线变化。

7.5.2 测试结果及分析

核心业务1：

事务/指标	计量单位	最小值	最大值	平均值	标准差	波动率	最新值
成功	次/s	380.20	794.20	781.38	54.20	6.94	380.20
失败	次/s	0.00	0.00	0.00	0.00	0.00	0.00

核心业务2：

事务/指标	计量单位	最小值	最大值	平均值	标准差	波动率	最新值
成功	次/s	781.70	1584.80	1555.19	104.78	6.74	797.10
失败	次/s	0.00	0.00	0.00	0.00	0.00	0.00

从图中可以看出，断开MySQL和MongoDB网络5秒的瞬间，核心业务1和核心业务2的TPS有轻微的下降，随后能恢复到正常水平，因此对核心业务基本没有影响。

7.5.3 测试结论

模拟网络攻击，对核心业务基本没有影响，满足目标要求。

7.6 稳定性测试

7.6.1 测试场景

按照模型1和最大容量的80%左右发起压力(核心业务1:600笔/秒和核心业务2：1200笔/秒)，运行8小时，观察系统是否能稳定运行。

7.6.2 测试结果及分析

核心业务1：

事务/指标	计量单位	最小值	最大值	平均值	标准差	波动率	最新值
成功	次/s	66.80	596.10	497.44	149.33	30.02	66.80
失败	次/s	0.00	0.00	0.00	0.00	0.00	0.00

核心业务2：

事务/指标	计量单位	最小值	最大值	平均值	标准差	波动率	最新值
成功	次/s	135.40	1189.90	1012.79	278.74	27.52	135.40
失败	次/s	0.00	0.00	0.00	0.00	0.00	0.00

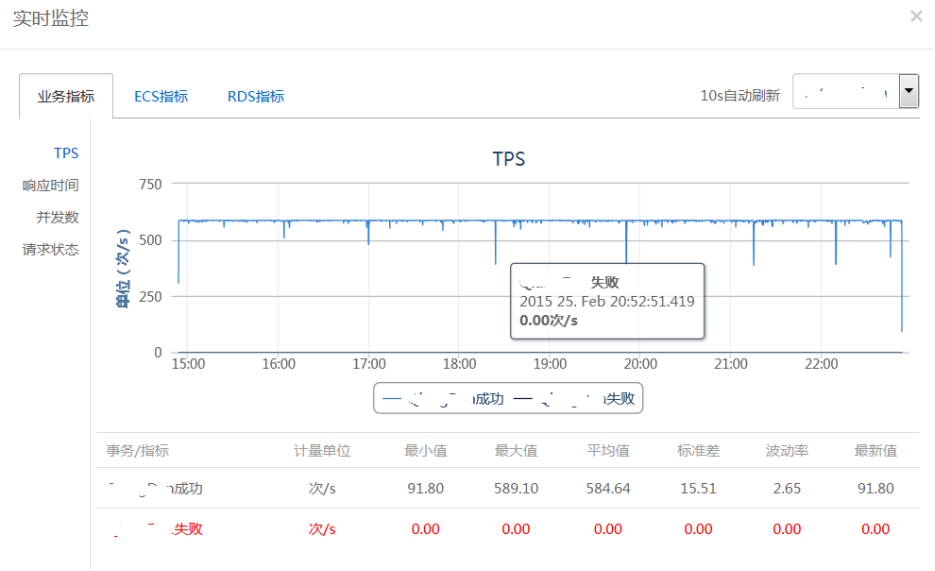
运行到35分钟后，核心业务1和核心业务2 TPS开始有轻微大幅度波动，运行到45分钟后，核心业务1和核心业务2 TPS开始大幅度波动，比较频繁，并且不能恢复到初始水平（过一段时间，TPS逐渐在下降），经过分析发

现是FULL GC导致，详见7.1.2测试结果及分析。 因此将压力降为一半（核心业务1：300笔/秒，核心业务2：600笔/秒），重新运行稳定性测试。

核心业务1：

事务/指标	计量单位	最小值	最大值	平均值	标准差	波动率	最新值
成功	次/s	33.90	308.80	293.51	8.40	2.86	33.90
失败	次/s	0.00	0.00	0.00	0.00	0.00	0.00

核心业务2：



事务/指标	计量单位	最小值	最大值	平均值	标准差	波动率	最新值
成功	次/s	91.80	589.10	584.64	15.51	2.65	91.80
失败	次/s	0.00	0.00	0.00	0.00	0.00	0.00

系统在核心业务1 300笔/秒和核心业务2 600笔/秒的压力下，基本能稳定运行8小时，但随着时间推移，Full GC次数越来越多，长时间运行下去将会导致系统处理能力大幅度下降（详见7.1.2测试结果及分析）

7.6.3 测试结论

在核心业务1 300笔/秒和核心业务2 600笔/秒的压力下，系统基本能稳定运行8小时，满足目标要求。

8 风险及建议

经过多次分析、调优及测试，基本能满足各业务场景的目标要求，但系统处理能力不能再继续上升的瓶颈主要体现在系统架构上，因此随着未来业务量猛增，超过系统处理能力的时候，将会产生处理能力急需下降、客户体现差甚至宕机的风险，建议针对系统架构进行修改，并进行架构类性能测试，满足日益增长的业务需要。

访问性能测试控制台