第35卷 第10期 计 算 机 工 程 2009年5月
Vol.35 No.10 Computer Engineering May 2009
·软件技术与数据库· 文章编号：1000—3428(2009)10—0051—03 文献标识码：A 中图分类号：TP311

# B/S 系统性能测试的设计与实现

简 玲
(上海市公安局公共信息网络安全监察处，上海 200042)

**摘 要：**性能测试是性能调优中的重要环节。为了准确地获取系统的表现，采用“模拟真实用户体验”的原则，在一个典型的B/S系统上实施包括准备、设计、开发、执行等过程的性能测试。通过性能测试，发现该系统存在的问题，并通过调优结果证实了这种性能测试方法的正确性。

**关键词：**性能测试；B/S系统；调优

# Design and Realization of Performance Test on B/S System

JIAN Ling
(Public Information Network Safety Supervision Department of Shanghai Public Security Bureau, Shanghai 200042)

**【Abstract】** Performance test is an important part in the test tuning. In order to accurately access the system performance, this paper uses "simulating the real user experience" principle, realizes a serial of performance test on a typical B/S system, including the preparation, design, development and implementation. Through the performance test, the existing problems of the system are found. Tuning results confirm that the accuracy of the test methods.

**【Key words】** performance test; B/S system; tuning

随着桌面操作系统以及Internet、以太网的普及，越来越多的企业采用B/S架构部署信息系统。据统计，46%的客户会由于速度缓慢、网站浏览错误而离开他们原本经常光顾的网站[1]。例如，如果Gmail操作页面相应速度很慢，客户多半会转投其他电邮网站。为了保证业务系统的成功，每个系统投产前会提到一个共同的问题——性能调优，即尽力去消除系统中存在的性能瓶颈，而性能调优的基础就是性能测试。本文通过一个典型的案例，以性能调优为目的，以模拟真实用户体验为原则，设计并实现了对某B/S系统的性能测试。

## 1 性能测试的概念及分类

性能测试作为一种技术，是调优的衡量尺度。它通过对待测试系统施压，获取待测试系统技术参数。

按照测试目的分类，性能测试可以分为压力测试(stress test)、负载测试(load test)、基准测试(benchmarks test)等[1]。其中，压力测试主要用于考验系统的稳定性，找出瓶颈；负载测试是为了获得不同负载下系统的表现；基准测试则是为了比较系统在各种软硬件配置下的表现情况。本文选用基准测试作为手段，使用同样的测试案例，在多轮测试中将用户响应时间、服务器资源占用等指标互相比较，并与期望值相比，进行决策调优。

## 2 待测试系统介绍

本文涉及的待测试系统架构如图1所示。系统基于B/S架构的3层服务体系，由Http服务器、应用服务器、工作流服务器、数据库服务器4个部分组成。其中，Http服务器处于防火墙外，其他服务器都位于防火墙内。Http服务器接收来自Internet的Http请求，提供Http静态页面服务，并将JSP/Servlet请求转发到应用服务器。应用服务器和工作流服务器实现业务逻辑，其中，应用服务器提供基础的页面展示、信息管理、后台计算功能；而工作流服务器则实现较为复杂的业务审批流程服务。数据库服务器提供数据存取服务，支持业务逻辑处理。

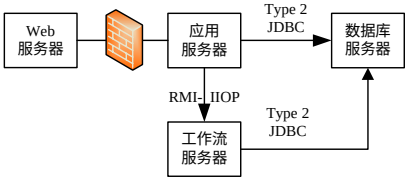


**图1 待测试系统架构**

在本系统中，预计有500个用户同时在线。在正常情况下，所有操作都在5 s内返回，最迟等待时间不能超过10 s。该系统在投入运营1年后，用户反映响应速度缓慢，而速度缓慢的原因无法精确定位，须进行性能测试来进行调优。为了找出根本缓慢的真实原因，在测试中的指导原则是以模拟真实用户体验为准。

## 3 性能测试的设计与实现

### 3.1 测试准备

测试的准备工作包括设备、应用特性分析2个部分。

#### 3.1.1 测试环境

建立一套与运营环境基本一致的待测试系统，包括软件、硬件，以及外围相关系统的一致性。如本系统还与其他一些信息系统存在实时交互的接口关系。在部署待测试环境时，尽可能模拟外部环境，必要的时候可以通过程序构造假的外部测试对象。而4台安装了LoadRunner的PC机部署在防火墙外，用于模拟发起用户的请求，其中1台作为总控，总体控制所有4台PC机向Web服务器发出请求。

**作者简介：**简 玲(1980－)，女，助理工程师，主研方向：计算机网络安全

**收稿日期：**2008-10-17 **E-mail：**madamjenny@sina.com

3.1.2　应用特性分析

应用特性分析用于找出能够模拟真实用户体验的简单方式，应先确定一些频发操作，然后组合为典型的应用场景。

(1)确定频发操作事件。本文的待测试系统比较复杂，包含上百个页面和上千个后台程序，而如果用 LoadRunner 等工具进行全面覆盖测试，工作量几乎与系统开发处于一个数量级。因此，只能选择频发操作来进行测试。通过分析系统原有操作日志，发现频发操作包括数据查询、数据收藏、关联搜索、数据删除等。

(2)确定典型应用场景。在性能测试中，技术人员常见的错误观点是，分别优化单个操作，就能优化系统的整体性能。而事实上系统运行时候，不同的操作之间往往对系统资源有互锁关系。为了模拟用户的真实体验，避免性能测试偏重于技术人员的主观想法，需要综合考虑整个系统的运行情况[1]。为了对系统性能有一个较全面地了解，必须同时考虑系统在不同的时间段可能承受到不同的压力负载情况，并分别测试这些情况下的系统表现。因此，必须确定典型应用场景，包括估算用户的种类、不同用户的操作过程、用户的使用时段等。通过分析系统原有操作日志，一种典型的应用场景为用户日常离散地进行数据查询、数据收藏、关联搜索这 3 种操作，另一种典型场景为在特定时间用户集中地进行数据删除操作。

### 3.2　测试用例设计与开发

测试用例开发是性能测试的设计阶段，接近于上文所提到的典型应用场景，有所不同的是，测试用例必须便于测试工具编码实现。

由于测试工具、资源所限，测试的复杂度不能太大，往往须对上一步获得的典型场景进行调整。待测系统某典型场景调整如图 2 所示，主要有 3 类用户进行操作，其中，用户组 1 不断重复进行操作 A, B；用户组 2 不断重复进行操作 B, C；用户组 3 不断进行操作 C, B。在长时间压力测试的情况下，从统计学角度来看，对于系统的压力就相当于用户组 1 重复操作 A 操作；用户组 2、用户组 3 分别进行 B, C 操作，把用户组进行的复杂操作，按此原理调整为逻辑简单的操作，就能在不影响测试精确度的情况下方便测试的实现。

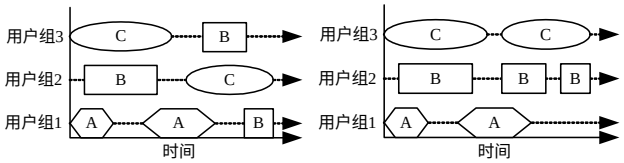


**图 2　待测系统某典型场景的调整**

测试用例的设计还须考虑负载的分配，包括用户的数量、组别、操作时间段、操作分布的百分比、持续的时间等。对于不明确的用户、组别进行简化；对于不明确的操作分布，按照“ 20-80 ”的原则进行分布，即有 80%的操作量都会集中在 20%的时间内完成，有 20%的操作种类访问频率达到 80%[2]。本文根据 2 个典型应用场景设计了 2 个测试用例。

### 3.3　测试开发

测试开发指用测试软件制作测试脚本，以便实现测试用例。测试脚本是一组程序代码，在执行时，可以模拟用户操作对服务器产生的请求数据，也可以接收并处理服务器返回的处理结果，或模拟用户的决策、等待等行为。

测试的开发分为以下几个步骤：

(1)用 LoadRunner 录制脚本，即根据测试用例，录制用户的操作情况。按照上文的分析，为了使测试易于实现，分别录制不同操作的脚本。在本例中，每次录制只记录 1 个操作，产生 1 个脚本。其中，登录、退出操作录制 1 次后，可以在后续脚本设计中复用。

(2)对于测试工具生成的脚本作 2 次开发。由于性能测试需要大量并发执行同种操作，而 B/S 系统中每个用户都拥有私有的 session 和 cookie，无法简单地将一个用户的操作克隆成许多用户的行为。而测试工具录制的脚本，比较单一化，并不适合直接用于测试，因此，须对脚本进行 2 次开发。脚本的 2 次开发中须进行以下工作：

1)考虑并发中会存在差异的值，比如用户 id，对策是将这些变量化，设置这些变量的取值范围、迭代方式等；2)考虑到业务分支情况，比如，将用户的选择操作通过随机数离散地分布到不同的后台数据上，这避免数据库操作与真实情况差异过大；3)考虑脚本之间并发时的数据相关性，比如数据查询用户组的登录脚本，须把 id 号传递给后续的操作脚本，以便后台程序进行身份检查。另外，本文还在脚本中加入随机的延时语句，模拟真实的用户等待时间。

(3)准备测试数据。由于需要模拟大量操作并发执行，因此要准备好合适的后台测试数据。在此用例中，事先设计了 500 个虚拟用户，分为 4 个组，每个组用于执行不同的操作；在后台准备了上千万条用于测试查询性能的业务数据，这些数据的主键连续，但是物理上离散分布；另外还有用于测试插入、修改性能的数据量很大的数据库表。待测试的数据规模是在运营数据库的规模上加上 1 年增长预估的情况下计算出来的。

(4)将脚本组装起来，按照测试用例的操作序列、比例、运行时间、并发数量设置。为了模拟测试用例 1，组装了 3 组脚本，分别对应 3 组虚拟用户，分别用于执行数据查询、数据收藏、关联搜索操作，设置持续运行时间为 1 min。测试脚本组装后，进行简单验证，确保脚本在无并发情况下能够正常运行。

### 3.4　测试执行

由于本文的测试是以优化目标驱动的，因此是迭代进行的。在迭代测试开始之前，先要备份测试现场，即把相关程序、测试脚本、测试数据、配置项记录备份下来，作为后续测试的基线，以便后续迭代时能够恢复到同样的起点。

测试执行流程如图 3 所示。

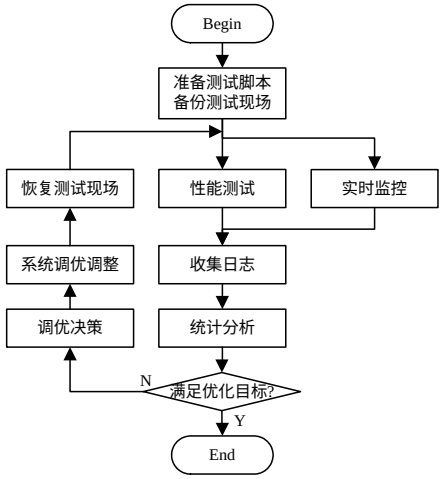


**图 3　测试执行流程**

后续的活动就是执行测试案例，收集测试结果并分析，

根据测试结果与优化目标的比较，决定是否结束调优[3]。如果不满足测试目标，则须研究后对系统进行调整，恢复测试现场后进入另一轮性能测试。每次调整后，继续测试所有案例，以防解决老问题带来新问题。

表 1 是从 LoadRunner 的测试报告中浓缩的摘要，可以看出，在第 1 次测试时，数据查询等操作的平均端对端响应时间高于 10 s，因此，必须从实时监控数据中进行分析。

**表 1　平均端对端响应时间**　　s

| 操作名称 | 第 1 次测试的响应时间 | 最后 1 次测试的响应时间 |
|---|---|---|
| 数据查询 | 10.044 | 4.248 |
| 数据收藏 | 15.864 | 4.932 |
| 关联搜索 | 11.115 | 1.836 |
| 数据删除 | 8.778 | 1.819 |

在测试中 JVM 的垃圾收集统计情况如表 2 所示。

**表 2　JVM 分类垃圾收集次数**

| 测试轮次 | 垃圾收集次数 | | | |
|---|---|---|---|---|
| | Scavenge | Old Full | Perm Full | System.gc |
| 第 1 次 | 251 | 121 | 11 | 113 |
| 最后 1 次 | 1 043 | 47 | 0 | 0 |

在第 1 次持续 1 h 的测试中，发生了 11 次 Perm Full 类垃圾收集。PERM 区存放一些永远不变的对象，用于管理 JVM 执行，这个层次的垃圾收集是异常的，并且非常消耗时间，很可能是由于 JVM 初始参数设置不当引起的。在系统刚刚投入运营的时候，因为程序代码相对较少，PERM 区不会因为空间用完而发生垃圾收集，但在运营 1 年后，随着程序版本变更，代码量增大可能就带来了 PERM 区的垃圾收集。因此，调大了 PERM 区的大小，后续又调大了 NEW 区，打开了多线程垃圾收集。

在最后 1 次测试中，Perm Full 类垃圾收集消失了，而 Scavenge, Old Full 这 2 类正常垃圾收集次数增多，JVM 运行正常。JVM 的参数调整，对程序运行速度有一定改善。除此之外，本文也分析了吞吐量、端对端响应时间、数据库情况等方面的监控结果，如表 1 所示，最后 1 次测试发现系统性能得到了很大的改善。

通过 8 轮迭代性能测试的帮助，可以发现 JVM 参数设置不合理、数据库索引不健全、应用服务器线程池过大的问题。优化系统后，系统业务吞吐量提升了 3 倍，平均端对端响应时间从 29.3 s 提升到 5.8 s，接近了原先的设计指标。这些提升证明，本文的测试能够有效地发现系统的问题。本文将测试的原则、过程、设计与开发概要、每一轮测试的结果以及调优建议等归档，作为后续解决性能问题及再次启动测试工作的参考[3]。

## 4　结束语

本文描述了在一个 B/S 架构上实现系统性能测试的实例，包括搭建测试环境、分析数据、统计业务特性，以及设计测试案例、开发执行、收集测试结果的全过程。文中性能测试的设计原则是以模拟真实用户体验为准。通过选取频发操作以及模拟典型应用场景的方法进行测试，结果真实地反映了系统存在的问题，较好地指导了调优工作。

## 参考文献


[1] Barber S. User Experience, Not Metrics[Z]. (2004-05-05). http://www.ibm.com/developerworks/rational/library/4228.html.

[2] 董　敏, 陈　勤, 呼秀婷, 等. 软件应用系统性能测试案例分析[J]. 现代邮政, 2006, (11): 52-54.

[3] Barber S. Beyond Performance Testing[Z]. (2004-06-24). http://www.ibm.com/developerworks/rational/library/4169.html.



编辑　顾姣健



(上接第 50 页)


可以看出，对于本文的数据集，扩展词选择 20 能取得最大的 *MAP* 值。

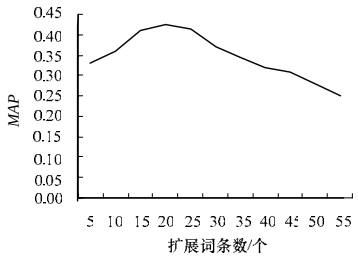


**图 1　扩展词个数和 *MAP* 关系图**

### 3.3　实验结果

表 1 为在此数据集上不进行扩展和分别用近义词法(即仅对原始查询进行同义词扩展)、两步伪相关反馈法(TSPRF)和 LSA 双语空间扩展法(LSAQE)进行跨语言查询扩展后的检索性能。其中，NOE 表示不进行查询扩展时的性能，其他表示相应扩展方法的查询性能及与 NOE 相比时改进的百分比。

**表 1　各种查询扩展方法比较**

| 方法 | *MAP* | Prec@10 | Prec@20 |
|---|---|---|---|
| NOE | 0.313 7 | 0.520 | 0.509 |
| SYNONYMY | 0.337 5(+7.5%) | 0.558(+6.8%) | 0.530(+4.1%) |
| TSPRF | 0.406 4(+29.6%) | 0.629(+21.0%) | 0.613(+20.4%) |
| LSAQE | 0.442 8(+41.2%) | 0.633(+21.7%) | 0.615(+20.8%) |

由表 1 可知，与未进行查询扩展相比，3 种扩展方法的检索性能均有明显的提高，其中，LSAQE 的 *MAP* 值提高的幅度最大，从 Prec@10 和 Prec@20 来看，TSPRF 和 LSAQE 提高的幅度不相上下，且均较大于 SYNONYM。

## 4　结束语

本文提出一种基于潜在语义分析的跨语言查询扩展方法，使最后的目标语言查询式中包括部分无需翻译的扩展词汇，削弱了翻译歧义对检索性能的负面影响。实验结果证明，该方法比传统的查询扩展方法在检索性能上有明显的提高。

## 参考文献


[1] Gao Jianfeng, Nie Jianyun, Zhang Jian, et al. TREC-9 CLIR Experiments[C]//Proc. of the 9th Text Retrieval Evaluation Conference. Gaithersburg, Maryland, USA: [s. n.], 2000.

[2] 盖　杰, 王　怡, 武港山. 基于潜在语义分析理论及其应用[J]. 计算机应用研究, 2004, 21(3): 9-20.

[3] Dumais S. Improving the Retrieval of Information from External Sources[J]. Behavior Research Methods Instruments & Computers, 1991, 23(2): 229-236.

[4] 万小军, 杨建武, 陈晓鸥. 文档聚类中 k-means 算法的一种改进算法[J]. 计算机工程, 2003, 29(2): 102-104.

[5] Salton G. The Smart Retrieval System-experiments in Automatic Document Processing[M]. New Jersey, USA: Prentice-Hall Inc., 1971.



编辑　陈　文