第 25 卷第 2 期 计算机应用 Vol. 25 No. 2
2005 年 2 月 Computer Applications Feb. 2005

文章编号:1001 - 9081(2005)02 - 0352 - 03

一种开放式性能测试框架的研究与应用

吴恒山,余华兵
(华中科技大学 计算机科学与技术学院,湖北 武汉 430074)
(szyhb810501. student@ sina. com)

摘 要:分析了一种基于 J2EE 的开放式性能测试框架 Grinder,阐述了它的工作原理,并对该框架做了两点改进:添加动态图形接口和从控制台向客户机发送脚本,最后基于改进的框架实现了 TPC-W 基准,并用它对国产数据库管理系统 DM4 进行了性能测试。

关键词: J2EE;Grinder;数据库;性能测试
中图分类号: TP311 **文献标识码**:A

Study and application of an open performance-test framework

WU Heng-shan, YU Hua-bing
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan Hubei 430074, China)

Abstract: This paper analyzed the principle of Grinder, an open performance test framework based on J2EE, and improved on this framework: adding a dynamic graph interface and sending scripts from console to clients. Finally we implement the TPC-W benchmark on the improved framework, and used it to test the national database management system-DM4.

Key words: J2EE; Grinder; database; performance test

0 引言

当前基于 J2EE 的应用非常广泛,但是当并发用户非常多时,容易出现性能瓶颈. 所以对于应用开发来说,需要在实际部署之前做好性能测试。为了找出系统的性能瓶颈,最常见的手段就是进行压力测试。压力测试一般有许多输入,或者需要在大的范围内改变输入,这就需要采用自动化的负载测试工具。Grinder 是一个自动化的负载生成和数据收集工具,和同类型工具如 ECperf 等相比,它具有许多优势:源代码完全开放;设置简单,容易使用;可扩展性好;可以测试任何 J2EE 应用程序;通过在 Java 虚拟机之间分布负载获得非常好的伸缩性。

1 Grinder 概述

1.1 运行原理

Grinder 由代理进程、工作进程和控制台组成。每台客户机运行一个单独的代理进程,代理进程创建指定数量的工作进程。每个工作进程启动多个工作线程,每个工作线程就相当于一个虚拟用户,工作线程执行用户指定的测试脚本,完成测试的内容。控制台用来协调工作进程的操作,给它们发送 start、reset 和 stop 命令。客户机上的工作进程定时向控制台发送统计报告。控制台有一个收集样本的线程定期更新显示。

1.2 脚本编写方法

在测试脚本中,用户需要定义 Test 对象(每个 Test 对象对应一个测试,如例 1 中的 test1,1 是测试编号,'test1'是测试描述),然后调用 Test 类的 wrap 方法,它能够把 Java 类和实例,Python 类、方法和函数包装成一个 TestPyInstance 或者 TestPyJavaInstance 对象,如例 1 中 testFunctionWrapper 就是封装全局函数 testFunction 后得到的 TestPyJavaInstance 对象。当调用 testFunctionWrapper 的时候,Jython 解释器就会调用 TestPyJavaInstance 对象的 invoke 方法,它间接调用 ThreadContext 类的 invokeTest 方法。每个工作线程都有一个相对应的 ThreadContext 对象,它最重要的作用是运行测试并且记录执行时间。在 ThreadContext 类的 invokeTest 方法中,调用 TestPyJavaInstance 对象父类的 invoke 方法执行全局函数 testFunction,并且记录执行时间。测试脚本还必须定义 TestRunner 类(推荐把 TestRunner 定义成类,但是严格地说,TestRunner 也可以定义成一个函数),为了使 TestRunner 实例可调用,__call__方法也是必须被定义的。

例 1:下面是 Jython 脚本示例:

```
def testFunction():
  ...                                          #用户要测试的内容
test1 = Test(1, 'test1')
class testRunner:
  testFunctionWrapper = test1. wrap(testFunction)
  def __call__(self):
  ...
  self. testFunctionWrapper()                  #执行测试 1
  ...
```

收稿日期:2004 - 07 - 16

作者简介:吴恒山(1953 -),男,湖北红安人,副教授,主要研究方向:数据库; 余华兵(1981 -),男,湖北松滋人,硕士研究生,主要研究方向:数据库.

1.3 重要特性

控制台的采样方法分为周期方法和快照方法。周期是一个模拟用户对一个测试脚本的完整执行。所谓周期方法就是指定测试脚本执行的周期数。所谓快照方法,就是由用户指定所收集的样本数量,而测试时间是固定的。

Grinder 按照正态分布改变相邻两个请求之间的实际思考时间。一般设置该分布的偏差是使 99.75% 的思考时间都位于该请求指定的思考时间的 20% 的偏差范围内。在某些场合可以使用 0% 的偏差,使用 0% 的偏差的测试运行称为精确的测试运行。

Grinder 使用的统计量包含在一组"原始统计量"中。用户可以定义附加的原始统计值,但是必须使用 userLong0 ~ userLong4 中的一个名字(对于 long 型值),或者 userDouble0 ~ userDouble4 中的一个名字(对于 double 型值)。每个测试的平均响应时间和吞吐量以及总计平均响应时间和总吞吐量都是从这些原始统计量导出的。Grinder 把每个测试调用称为一个事务处理,它用每秒钟的事务处理数量(TPS)来描述吞吐量。

2 对 Grinder 的改进

2.1 提供动态图形接口

使用未经改进的 Grinder,用户无法知道系统性能在整个测试期间的变化情况,也不知道系统何时开始稳定。解决这个问题的办法是添加动态图形接口,让用户利用这个接口定制自己的统计图。

在控制台,每个测试都对应一个样本累加器(SampleAccumulator),还有一个总计样本累加器用来记录所有测试的统计结果总和,每个样本累加器都有一个样本监听器(SampleListener)队列。当客户机向控制台注册测试的时候,就向相应样本累加器的样本监听器队列添加监听器。样本监听器只有一个方法更新,这个方法有两个参数:一个参数记录了在采样时间区间的统计结果,另一个参数则记录了自从收集样本以来的累积统计结果。一个收集样本线程(Sampler)负责定期采集样本,每隔一定的时间,就调用每个样本累加器的所有样本监听器的更新方法更新显示。研究发现:在忽略样本期间,样本累加器不会接收工作进程发送的统计结果。为了获取系统自从运行以来的性能信息,本文采用的办法是添加另一套样本累加器,用于在忽略样本期间接收工作进程发送的统计结果。

具体的实现办法是:在控制台添加画图面板(CurveGraphPanel)用于显示统计图,定义数据接口 DataInterface 和画图接口 DrawInterface。DataInterface 接口用于获取自定义的样本累加器的统计数据,并给 DrawInterface 接口提供绘图数据。CurveGraph 类是对数据接口 DataInterface 和画图接口 DrawInterface 的封装。CurveGraphFactory 类负责创建 CurveGraph 对象,每个 CurveGraph 对象对应一幅统计图。CurveGraphFactory 类提供了一个散列表,保存图名和 CurveGraph 对象的映射关系,这样,当用户在画图面板的图名下拉菜单中选择某一图名的时候,就可以查找到相应的 CurveGraph 对象,并调用它的绘图方法绘制统计图。CurveGraphFactoryManager 类有 initialize 和 addSampleListener 两个公有静态方法。initialize 调用 CurveGraphFactory 静态实例,创建统计图层;addSampleListener 用于向相应样本累加器的样本监听器队列添加监听器,从样本累加器获取统计数据,向数据接口填充绘图数据。

当使用上述动态图形接口的时候,用户需要做的工作是:实现画图接口 DrawInterface 的方法 draw,在 CurveGraphFactoryManager 的 initialize 方法中创建 CurveGraph 对象,并在 addSampleListener 方法中添加自定义的样本监听器,即实现它的更新方法。

2.2 测试脚本的发送

使用未经改进的 Grinder,用户必须在一台客户机上编辑 grinder. properties 文件和测试脚本,然后把它们复制到其他客户机。改进以后,用户在控制台编辑 grinder. properties 文件和测试脚本,然后向所有客户机发送准备接收文件的消息,这个消息封装了目的客户机的 IP 地址,接着启动发送文件的服务器线程,客户机收到消息以后,就把消息中的 IP 地址和自己的 IP 地址进行对比,如果相同,就启动接收文件的客户端线程。这是因为考虑到不同客户机可能执行不同的测试脚本,所以必须能够把测试脚本发送到指定 IP 地址的客户机。当客户机的数量较多的时候,不仅能节省人力,而且方便集中管理和维护,提高测试的效率。

3 基于 Grinder 的 TPC-W 性能测试

3.1 TPC-W 概述

TPC-W 是由事务处理性能委员会(TPC)制定的针对电子商务应用的评测规范,它模拟一个具有顾客注册、浏览和定购功能的网上书店,总共有 14 个网页。TPC-W 通过改变浏览和购买的比例定义了三种网络交互组合:WIPS(代表一般情况)、WIPSb(偏重于浏览)和 WIPSo(偏重于购物)。对于 WIPS 组合,所访问的网页中 80% 是浏览网页,20% 是定购网页;对于 WIPSb 组合,所访问的网页中 95% 是浏览网页,5% 是定购网页;对于 WIPSo 组合,所访问的网页中浏览网页和定购网页各占一半。TPC-W 的性能度量标准是每秒钟的网络交互次数(WIPS)和每个网络交互的价格($/WIPS)。

3.2 基于 Grinder 的 TPC-W 测试方案

3.2.1 定义性能度量标准

实验的性能度量标准是最大可接受响应时间。按照 TPC-W 规范,很容易确定这 14 个页面的最大可接受响应时间。TPC-W 规范还要求每个网页多次交互中的 90% 的响应时间小于该页面的最大可接受响应时间。

3.2.2 编写测试脚本

实现 TPC-W 测试规范需要定义 14 个测试(写法参见例 1),分别对应 14 个网页。在测试脚本中自定义一个统计量,计算每个网页的多次交互中响应时间小于最大可接受响应时间的交互所占的百分比。首先必须定义一个附加的原始统计值,用 userLong0 记录响应时间小于最大可接受响应时间的事

务数目，如下所示，如果响应时间小于 3 000 毫秒，就把 userLong0 的值设置为 1，否则 userLong0 的值为 0。

```
if grinder.statistics.time < 3000:
  lessThanMARTIndex = StatisticsIndexMap.getInstance().
    getIndexForLong("userLong0")
  grinder.statistics.setValue(lessThanMARTIndex, 1)
```

为了让 Grinder 显示每个网页的多次交互中响应时间小于最大可接受响应时间的交互所占的百分比，首先创建一个新的 StatisticsView 对象，再创建 ExpressionView 对象，它有一个名称、一个控制台（可以用来使该名称国际化的键），以及一个表达式。再把它添加到 StatisticsView 中。最后把这个 StatisticsView 注册为“概要视图（summary view）”，这意味着它将出现在日志文件的概要表格以及控制台的“Results”面板中。如下所示：

```
summaryView = StatisticsView()
summaryView.add(ExpressionView("90% wirt", "",
  "(* 100 (/ userLong0 timedTransactions))"))
grinder.registerSummaryStatisticsView(summaryView)
```

该表达式使用后缀格式，等价于 100 *（userLong0/timedTransactions）。其中，timedTransactions 是计时事务的数量。（使用 Grinder 时，可以让一部分客户机记录时间，而让另一部分客户机不记录时间。在记录时间的客户机上执行的事务称为计时事务，在不记录时间的客户机上执行的事务称为不计时事务。）

在正常情况下，一个测试执行完后将自动报告测试结果。但是，如果要在测试完成后修改统计，在执行测试前必须延迟报告。接下来执行测试并且修改统计，然后调用 report 报告测试结果。如下所示：

```
grinder.statistics.delayReports = 1
...                                    #执行测试，修改统计
grinder.statistics.report()
```

3.2.3 定义采样方法

采用快照方法收集数据，样本大小为 1 秒（这意味着每 1 秒钟记录一次性能测度）。考虑到系统稳定所经历的初始测试周期，忽略前 600 个样本（10 分钟），收集 1 800 个样本（30 分钟）。

3.2.4 搭建 J2EE 测试环境

测试环境如图 1 所示。使用七台连接在 100Mbps 快速以太网交换机上的计算机：一台作为控制台，三台客户机，两台 Web 服务器，一台 DM4 数据库服务器。操作系统为 Windows 2000 高级服务器版。

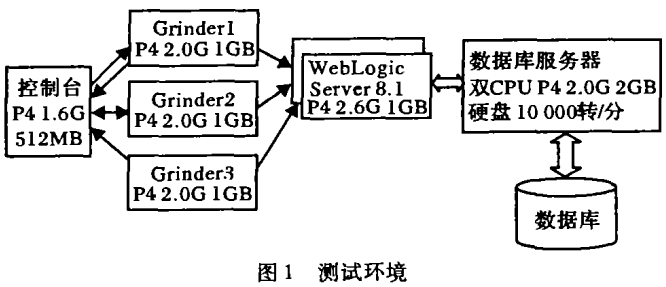

图 1 测试环境

安装 DM4 数据库服务器时，设置数据文件页大小为 8KB，段大小为 16。在装载 TPC-W（V1.8）数据库时，确定模拟浏览器（EBS）个数为 3 200，书的数量（ITEM）为 10 000。装载完以后建立索引。

3.2.5 设置并执行测试

首先设置数据库服务器：内存池为 200MB，缓冲区块数为 70 000，工作线程为 4，检查点时间间隔为 15 分钟。然后设置 Web 服务器：工作线程数为 1 000，最大连接数为 160。再修改 Grinder 的属性文件 grinder.properties：工作线程数为 100，设置控制台的 IP 地址和端口，起初启动 1 个工作进程，以后每隔 20 秒启动 1 个工作进程，思考时间偏差为 0%，指定测试脚本。最后设置测试脚本中的全局变量：14 个网页对应的 Web 服务器 IP 地址和端口，不传送图片，书的数量（ITEM）为 10 000。每次测试前，都必须修改属性文件中的工作进程数和测试脚本中的顾客数量、网络交互组合类型。

在测试过程中，需要反复试探，不断增加工作进程数，直到至少一个网页的多次交互中响应时间小于最大可接受响应时间的交互所占的百分比小于 90%。表 1 就是最终得到的最大并发用户数和每秒网络交互次数。

表 1 测试结果

网络交互组合	最大并发用户数	每秒网络交互次数
WIPS	2 100	286
WIPSb	1 600	208
WIPSo	3 200	432

图 2 是利用动态图形接口创建的统计图，它反映每秒钟网络交互次数的变化情况。从图中可以直观地看到系统稳定的部分（水平的部分）。

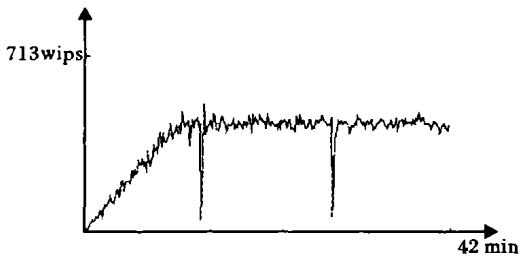

图 2 WIPSo wips 图（并发用户数为 3 200）

4 结语

Grinder 提供了一个灵活的测量任何 J2EE 应用程序性能的框架。本文对 Grinder 进行了两点改进，增强了它的实用性，最后给出了一个用改进的 Grinder 对 DM4 进行 TPC-W 性能测试的具体方法和实现。

参考文献：

[1] ZADROZNY P. J2EE 性能测试[M]. 张文耀，叶茂盛，陈爱国，译. 北京：电子工业出版社. 2003.

[2] TPC. TPC Benchmark W Specification Version 1.8. TPC[Z], 2002.2.19.

[3] DM4 系统管理员手册[Z]. 武汉华工达梦数据库有限公司，2003.

[4] 萨师煊，王珊. 数据库系统概论（第三版）[M]. 北京：高等教育出版社，2002.

[5] ZUFFOLETTO J. BEA WebLogic Server 宝典. 邱巍峰，袁建洲，张海峰，译. 北京：电子工业出版社，2003.