清 华 大 学

**综 合 论 文 训 练**

题目：基于lkp-test的linux kernel性能分析

系 别：计算机科学与技术系

专 业：计算机科学与技术

姓 名：韩慧阳

指导教师：陈渝 副教授

2016年5月27日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签 名： 导师签名： 日 期：

# 中文摘要

在几乎所有的主流开源软件中，软件运行出现的问题大致分为两类：非性能缺陷和性能缺陷。其中后者通常具有难以检测、极为影响运行性能、可以通过少量代码更改解决等特点。因此，性能缺陷是开源软件当中（包括linux kernel）不可不引起重视的一点。

目前性能缺陷的精确检测尚没有成熟的技术，而使用lkp-tests(linux kernel performance tests)来进行内核分析，从而发现性能缺陷是一种很有效的方法。不过lkp-tests本身存在测试样例过多，测试冗余程度大等缺陷。

本文将从lkp-tests结果数据中进行一些分析，目的在于帮助改进lkp-tests的测试框架，减少冗余，并检测关键指标随着linux kernel版本的变化模式。lkp-tests是一个由因特尔开源技术中心高级工程师吴峰光建立的linux kernel性能检测框架，其从不同的benchmark、编译器、内核版本出发进行测试，使用不同的指标反映出内核的性能。

**关键词：**lkp-tests；指标；降维；相关性

# ABSTRACT

There are two main problems in almost all the OSS(open source software), non-performance bugs, which is also known as Functional bugs and performance bugs. And the latter usually are hard to detected, have considerable influence on the system performance and could be generally solved by only a few of lines of codes. That is why they should be taken into consideration seriously.

There is no way to detect performance bugs precisely till now, but lkp-tests(linux kernel performance tests) have shown that it can be very helpful in detection of performance bugs. However, lkp-tests is has too many extra tests which are not necessary, and they show quite limited results. It has limited the cost of the tests.

This paper will do some analysis base on results from lkp-tests, aiming to improve the lkp-tests framework and cut down testing cost. And the changing mode of key indicators along with different linux kernel versions will be detected, too.

LKP-Tests is a linux kernel performance testing framework authored by Fengguang Wu, , Senior Engineer @ Intel Open Source Technology Center. It tests a variety of indicators on benchmarks with different kernels and compilers.

**Keywords:** lkp-tests; indicators; Dimensionality reduction; correlation

# 目 录

[中文摘要 I](#_Toc452126848)

[ABSTRACT II](#_Toc452126849)

[目 录 III](#_Toc452126850)

[第1章 引言 IV](#_Toc452126851)

[1.1 研究背景 IV](#_Toc452126852)

[1.2 研究现状 IV](#_Toc452126853)

[1.2.1 降维 V](#_Toc452126857)

[1.2.2 相关性分析 VI](#_Toc452126858)

[第2章 lkp-tests概述 VII](#_Toc452126859)

[2.1 工作机理 VII](#_Toc452126861)

[2.2 结果格式 VII](#_Toc452126862)

[2.3 冗余测试简述 VII](#_Toc452126863)

[第3章 实验设计 VIII](#_Toc452126864)

[3.1 指标降维过程 VIII](#_Toc452126868)

[3.2 指标和配置的相关性分析 VIII](#_Toc452126869)

[3.2.1 单个benchmark的指标相关性分析 VIII](#_Toc452126870)

[3.2.2 不同benchmark之间配置的相关性分析 VIII](#_Toc452126871)

[3.3 指标变化模式的分析 VIII](#_Toc452126872)

[第4章 实验过程 1](#_Toc452126873)

[4.1 实验环境 1](#_Toc452126878)

[4.2 指标降维过程 1](#_Toc452126879)

[4.3 指标和配置的相关性分析 1](#_Toc452126883)

[4.3.1 单个benchmark的指标相关性分析 1](#_Toc452126884)

[4.3.2 不同benchmark之间配置的相关性分析 1](#_Toc452126885)

[4.4 指标变化模式的分析 1](#_Toc452126886)

[第5章 结论 2](#_Toc452126887)

[5.1 工作总结 2](#_Toc452126893)

[5.2 目前的问题与未来的工作 2](#_Toc452126894)

[插图索引 3](#_Toc452126895)

[表格索引 4](#_Toc452126896)

[参考文献 5](#_Toc452126897)

[致谢 6](#_Toc452126898)

[声明 8](#_Toc452126899)

[附录A 外文资料的书面翻译 9](#_Toc452126900)

[附录B 参考内容 10](#_Toc452126901)

# 引言

## 研究背景

随着互联网技术的高速发展，传统的面对面工作方式对于软件产业来说已经变得不再必需，多数开源软件的开发过程都涉及了大量的开发人员。这些开发者，通常都是通过版本控制系统(VCS)和email进行交流。因此，开源软件通常具有员工工作选择自由度大、整个软件没有严格的系统级设计、没有明确的项目规划时间表等常见问题的出现(2)。目前传统的软件问题测试几乎都是通过随机黑盒测试，而性能缺陷通常不会直接造成系统的崩溃，二是体现在降低吞吐量、增加延迟、浪费资源上面，所以既难以检测，又会造成大型项目的整体漏洞的出现(4)。

鉴于性能缺陷对系统整体影响巨大，有很多的性能缺陷检测的方法提出(5)(6)(7)，

纵观目前提出的这几种方法符号标记(5)、缺陷追踪(7)、负载测试(6)，能大规模推广的仍然是与传统黑盒测试相似的负载测试方法，而lkp tests正是针对linux kernel建立的测试框架，为各种研究linux kernel性能缺陷的工作提供了数据基础。

而正是由于前面所提到性能缺陷的各种特点，lkp tests目前仍然存在测试不准确且测试样例大量冗余的弊端（这是因为性能缺陷的位置不能严格检测，所以设计之初需要力求覆盖面全）。冗余测试的存在浪费了大量的时间和精力，但是我们并没有得到更多的有用的结果，因此测试样例的精简是亟待解决的一个问题。

我们主要根据lkp tests测试结果的数据特征，从数据挖掘的角度出发，通过聚类、降维、相关性分析等手段，寻找冗余的测试样例、测试指标、测试配置等，分析过程会涉及多次不同维度的数据处理，并且对结果进行具有现实意义的解释。除此之外，文章还检测了随着不同版本的linux kernel的变化，关键指标的变化模式，避免因指标本身的变化规律对测试结果造成干扰。

## 研究现状

目前，因为lkp tests框架本身比较新，针对其进行的优化工作几乎没有，但是根据前面的介绍我们知道我们需要的主要是以下几个方面的技术：



### 降维

降维是机器学习和数据挖掘中经常用到的一个概念，通常指的是减少需要考虑的随机变量的数目的过程，最终获得一组“不相关”的主成分变量。通常降维的过程会分为特征选择和特征提取两个主要步骤，后者是降维处理的重点。其实是一种高维数据转化为一种有意义的表示的过程，理想情况下，对应于源数据，降维后的数据通常会有一个固定的维数(8)。

随着大数据时代的到来，诸如语音信号、数字照片、核磁共振成像扫描等技术产生了大量高维数据，因此降维的技术也应运而生。当前比较流行的降维算法有针对线性变化数据的PCA(Principle Component Analysis)算法(8) 即主成分分析和LDA(Linear Discriminant Analysis)即线性判别分析；也有针对非线性变化数据的LLE(Locally linear embedding)即局部线性嵌入和LE(Laplacian Eigenmap)即拉普拉斯特征映射。

#### PCA

PCA又称主成分分析，其主要思想是将高维数据投影到一个低维空间，使投影到这个低维空间之后原数据的方差能得到最大化的保留，这样可以使用低维空间来表示原来的高维数据(10)。并且“与小波变换相比，PCA能够更好地处理稀疏数据”(10, p102)，这一点将成为我们后续选择的时候的重要依据。

#### LDA

LDA也是一种线性降维方法，它与PCA相比有相似之处，同样也是寻找一个可以投影的轴，但是PCA侧重于保留数据信息，LDA侧重于将不同类的数据分离开。所以忽略了一些信息，比起PCA来说，丢掉了更多的信息(15)。

#### LLE

和以上两种方法不同，LLE是一种非线性的降维方法，它是从图形的角度出发去分析数据特点，所以该方法的特点在于投影到低维空间之后，数据还能保留高维数据的流型结构。但是有的时候LLE是不适用的（比如数据平均分布在一个椭球面或者圆球面上），所以应用不是非常广泛(14)。

#### LE

LE的思想与LLE有相似之处，但是它的主要特征是降维之后，原来有关系的数据点还能够相近(13)。

### 相关性分析

相关性分析已经是一个很成熟的技术了，这里我们实际需要的是离散数据的相关性分析（而且这种离散的数据中通常有缺失值）。目前比较流行的衡量两组数据离散相关性的方法是使用Pearson相关系数和Spearman相关系数。两者的区别如下表所示 (16)：

|  |  |  |
| --- | --- | --- |
|  | Pearson | Spearman |
| 数据集要求 | 正态分布 | 没有要求 |
| 分析来源 | 数据的均值方差，属于积差相关值 | 排序值 |
| 衡量侧重点 | 线性相关性 | 非线性相关性 |
| 参数要求 | 参数统计法， | 非参数统计法，需额外选择度量参数 |

另外，根据Jan(16)等的分析，“不要把Spearman相关性系数作为一个关键性的衡量指标进行过度解读”(16，p6)。我们会在之后根据二者的特点和数据的特征进行挑选。

# lkp-tests概述

本文的研究工作都是基于lkp-tests的测试结果进行的，在这里我们先对其进行一些基本的介绍。



## 工作机理

lkp-tests选择了一系列的benchmark作为自己测试的工具，在每个benchmark中又会分为不同的硬件配置、compiler、内核版本等等。最终的测试是用不同侧重点的指标来展现的。一次测试通常需要经过以下流程：

## 结果格式

lkp-tests的测试结果由三部分组成，matrix.json、job.yaml还有一个文本文件。三个结果文件的存储路径为（以ebizzy的一次测试结果为例）：

|  |
| --- |
| /result/ebizzy/100%-10x-10s/lkp-ws02/eywa-rootfs/x86\_64-rhel/gcc-4.9/6a13feb9c82803e2b815eca72fa7a9f5561d7861/ |

表1

上面路径中的每一个目录都有其存在的实际意义，依次为：

|  |
| --- |
| 结果根目录/benchmark名/硬件配置1/硬件配置2/硬件配置3/内核版本/编译器版本/commit ID |

表2

其中的文本文件是对本次测试的一次描述，而job.yaml是本次测试运行的工作，所以实际的测试结果只存在matrix.json中。下面我们用一个样例文件来介绍结果的具体格式：

|  |
| --- |
| {  "uptime.boot": [  179.89,  240.59,  278.72  ],  "uptime.idle": [  1868.05,  3320.11,  4214.25  ],  "numa-numastat.node0.numa\_hit": [  277959267,  273755067,  289353920  ],  } |

表3

如上表所示，是一个原始数据中的matrix.json的片段。数据采用json的格式存储，键是一个字符串，对应的是一个测试指标。值是一个列表，对应的是几次重复的测试，其中列表的规模是重复测试的次数，上面的例子中来自ebizzy（其中一个benchmark），代表的意思是，三个指标被重复测试了三次，分别得到了三组测试结果。测试结果是一个浮点值。

## 冗余测试简述

我们前面已经说过，本文的其中一部分工作就是减除冗余的数据，那么什么是冗余的数据呢？

根据数据格式，冗余数据可以分为指标性的冗余和配置性的冗余。

### 指标性冗余

一次测试中可能包含很多不同的指标（作为单独一个benchmark，ebizzy测试指标超过13000个），那么是不是每个指标都是有意义的就是一件值得思考的问题，也许有很多指标都在测试系统的本地I/O，他们之间存在很强的相关关系，比如一个指标测试的是系统一分钟接受的任务数目，另一个指标测试的是系统十分钟接受的任务数目，两者是完全成比例的，也就是说我们知道其中一个就可以计算出另外一个。这时我们就可以认为其中一个是冗余的。

### 配置性冗余

另外根据之前对路径的解释（表1、表2），我们可以看到，一次完整的测试对应的是一个完整的配置（benchmark、三个硬件配置、内核版本、编译器版本、commit ID）。对于同一个benchmark而言，如果两个配置的测试结果几乎没有差别，那么我们可以认为其中一个配置是冗余的。比如说，假如一个benchmark主要测试的是系统的本地I/O，两个配置出了commit ID意外都是一样的，但是两次commit的差别仅仅是改变了网络应用的速度，对于本地的I/O没有任何影响，也就是说两次测试结果完全是一样的，那么我们可以将其中一个配置认定为是冗余配置。

除此之外，不同benchmark之间也会存在这样的冗余配置，比如两个benchmark测试侧重点不同，但是都有本地I/O速率的测试，两个配置仅仅表现为benchmark的不同，其实内部测试都是一样的，那么这两个配置中可以认定其中一个为冗余配置。具体哪一个为冗余还需要具体的观察才能判断。

# 实验设计



## 指标降维过程

## 指标和配置的相关性分析

### 单个benchmark的指标相关性分析

### 不同benchmark之间配置的相关性分析

## 指标变化模式的分析

# 实验过程



## 实验环境

## 指标降维过程



## 指标和配置的相关性分析

### 单个benchmark的指标相关性分析

### 不同benchmark之间配置的相关性分析

## 指标变化模式的分析

# 结论



## 工作总结

## 目前的问题与未来的工作

# 插图索引

# 表格索引

# 参考文献

1. Adrian Nistor1, Linhai Song2, Darko Marinov1 and Shan Lu. ***Toddler: Detecting Performance Problems via Similar Memory-Access Patterns.***] In ICSE, 2013.
2. Yepang Liu, Chang Xu and Shingchi Cheung. ***Characterizing and Detecting Performance Bugs for Smartphone Applications.*** In ICSE, 2014.
3. Michael.J.Fischer, Martin Pinzger, Harald Gall. ***Populating a Release History Database from version control and bug tracking systems.*** In ICSM, 2003.
4. Nachiappan Nagappan, Thomas Ball and Andreas Zeller. ***Mining Metrics to Predict Component Failures.*** In ICSE, 2006.
5. T.Zimmermann, Rahul Premraj and Andreas Zeller. ***Predicting Defects for Eclipse.*** In MoDELS, 2007.
6. Sunghun Kim, Thomas Zimmermann, E.James Whitehead and Andreas Zeller. ***Predicting Faults from Cached History.*** In ICSE, 2007.
7. Sunghun Kim, Thomas Zimmermann, E.James Whitehead and Andreas Zeller. ***Predicting Faults from Cached History.*** In ICSE, 2007.
8. Patrice Godefroid, Nils Klarlund and Koushik Sen. ***DART: directed automated random testing.*** In pldi, 2005.
9. Cristian Cadar, Daniel Dunbar and Dawson R Engler. ***KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs.*** In OSDI, 2008.

9. Dawson R Engler, David Yu Chen, Seth Hallem, Andy Chou and Benjamin Chelf. \_\*\*[Bugs as deviant behavior: a general approach to inferring errors in systems code.](http://research.cs.queensu.ca/~ahmed/home/teaching/CISC880/F10/papers/BugsAsDeviantBehavior\_SOSP2001.pdf)\*\*\_ In sosp, 2001.

10. Cristian Cadar and Koushik Sen. \_\*\*[Symbolic execution for software testing: three decades later.](http://www.eecs.berkeley.edu/~raluca/cs261-f15/readings/symb.pdf)\*\*\_ In Communications of the ACM, 2013.

11. Adrian Nistor, Tian Jiang and Lin Tan. \_\*\*[Discovering, reporting, and fixing performance bugs.](http://mir.cs.illinois.edu/~nistor1/pubs/msr13.pdf)\*\*\_ In MSR, 2013.

12. Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz and Anand Swaminathan. \_\*\*[Mining email social networks.](http://research.microsoft.com/pubs/138221/bird2006mes.pdf)\*\*\_ In MSR, 2006.

13. Luis Lopezfernandez, Gregorio Robles, Jesus M Gonzalezbarahona, U Rey and Juan Carlos. \_\*\*[Applying Social Network Analysis to the Information in CVS Repositories.](http://gsyc.es/~jgb/libresofteng/sna-repositories-2004/sna-repositories-2004.pdf)\*\*\_ In MSR, 2004.

14. Jaime Spacco, David Hovemeyer and William Pugh. \_\*\*[Tracking defect warnings across versions.](http://www.irisa.fr/lande/lande/icse-proceedings/msr/p133.pdf)\*\*\_ In MSR, 2006.

15. Abram Hindle, Daniel M German and Ric Holt. \_\*\*[What do large commits tell us?: a taxonomical study of large commits.](http://turingmachine.org/~dmg/papers//dmg2008\_msr\_largeCommits.pdf)\*\*\_ In MSR, 2008.

16. Daniel M German. \_\*\*[Mining CVS repositories, the softChange experience.](http://2004.msrconf.org/papers/German.pdf)\*\*\_ In MSR, 2004.

17. Sunghun Kim, E J Whitehead and Yi Zhang. \_\*\*[Classifying Software Changes: Clean or Buggy?.](https://users.soe.ucsc.edu/~ejw/papers/cc.pdf)\*\*\_ In IEEE, 2008.

18. S.M.A. Zaman, Bram Adams and Ahmed E.Hassan. \_\*\*[A qualitative study on performance bugs.](http://sail.cs.queensu.ca/Downloads/MSR2012\_AQualitativeStudyOnPerformanceBugs.pdf)\*\*\_ IN MSR, 2012.

19. Godfrey and Qiang Tu. \_\*\*[Evolution in open source software: a case study.](http://plg.uwaterloo.ca/~migod/papers/2000/icsm00.pdf)\*\*\_ In ICSM, 2000.

20. Kunrong Chen and Vaclav Rajlich. \_\*\*[Case study of feature location using dependence graph.](http://www.cs.wayne.edu/~vip/publications/Chen.IWPC.2000.FeatureLocation.pdf)\*\*\_ In ICPC, 2000.

21. Thomas Eiter and Heikki Mannila. \_\*\*[Computing Discrete Fr´echet Distance](http://www.kr.tuwien.ac.at/staff/eiter/et-archive/cdtr9464.pdf)\*\*\_ In CD-TR, 1994.

# 致谢

# 声明

# 附录A 外文资料的书面翻译

# 附录B 参考内容