**密级:**



**硕士学位论文**

**大数据应用可靠性测试框架设计与实现**

**作者姓名： 郑莹莹**

**指导教师: 叶丹 研究员**

**中国科学院大学软件研究所**

**学位类别: 工学硕士**

**学科专业: 软件工程**

**培养单位: 中国科学院大学软件研究所**

**2017年 4月**

**Design and Implementation of a reliability benchmarking framework for big data applications**

**By**

**Zheng Yingying**

**A Dissertation Submitted to**

**University of Chinese Academy of Sciences**

**In partial fulfillment of the requirement**

**For the degree of**

**Master of Software Engineering**

**Institute of Software**

**University of Chinese Academy of Sciences**

**April, 2017**

**独创性声明**

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明。

签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**关于论文使用授权的说明**

本人完全了解中国科学院软件研究所有关保留、使用学位论文的规定，即：中国科学院软件研究所有权保留送交论文的复印件，允许论文被查阅和借阅；中国科学院软件研究所可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。

（保密的论文在解密后应遵守此规定）

签名：\_\_\_\_\_\_\_\_\_\_ 导师签名：\_\_\_\_\_\_\_\_\_\_ 日期：\_\_\_\_\_\_\_\_\_\_

**大数据应用可靠性测试框架设计与实现**

**摘要**

随着互联网等的发展，越来越多的领域产生了海量、高速的数据。当前，Spark等分布式处理框架被广泛应用于这些数据的处理分析中，常见的应用场景有SQL查询、大规模图分析、机器学习等。然而，这些大数据应用在处理数据时，经常出现内存溢出、I/O异常、任务超时等多样及复杂的执行错误，这些可靠性问题会直接造成应用执行失败。本文主要关注系统故障或软件错误，即系统在特定配置下是否会发生执行出错或得不到正确结果等情况。然而，现有的测试基准主要提供性能以及扩展性测试，目前还没有一个通用的测试方法能够提前发现系统、应用和数据的潜在问题。针对大数据应用面临的运行时错误以及现有测试基准的不足，论文设计并实现了大数据应用的可靠性测试框架，支持异常数据生成和参数组合测试。

论文介绍了可靠性测试基准的设计，并针对以下关键技术进行研究。（1）提出了针对大数据应用的异常数据生成方法。首先，本文定义异常数据特征为：数据量大、数据倾斜、数据稀疏、数据维度高以及数据分布异常；然后，通过分析应用计算特性，提出应用计算特性与数据异常特征对应关系，并给出了特定应用的异常数据的生成规则；最后，通过异常数据规则生成相应的异常数据。（2）提出了针对大数据应用的参数组合测试方法。通过贪心算法对系统和应用参数进行组合空间削减测试；同时提出探测性参数验证方法，通过指数增长的慢启动方式来确定最差的资源占用的参数取值。

论文详细地介绍了大数据应用可靠性测试框架的设计，并在Spark系统上进行实现。通过对Spark上的基准应用进行可靠性测试，目前在6个基准应用中发现了三种类型的错误（内存溢出错误、运行超时以及计算结果错误），同时也验证了本文提出的异常数据生成以及参数组合测试方法的可用性。

**关键词：**可靠性，基准测试，异常，大数据系统，大数据应用

**Design and Implementation of a reliability benchmarking framework for big data applications**

**ABSTRACT**

With the development of the Internet, "massive" and "high speed" data has been produced in more and more areas. At present, Spark and other distributed processing frameworks are widely used in the processing and analysis of these data. These systems have high scalability, but the applications running atop them often generate runtime errors, such as out of memory errors, I/O exceptions, and task timeouts. These reliability issues can directly lead to application failures. Reliability is one of the six characteristics of software quality. This paper focuses on reliability in the event of system faults or software bugs. For the above reliability problems, existing benchmarks primarily provide performance testing, and often use conventional data as well as fixed configurations to test. This paper designs and implements a reliability benchmarking framework for large data system applications, in order to find out the reliability of the system or application in advance.

The paper introduces the design of reliability benchmarking and studies the following key technologies. Firstly, this paper proposes an exception data generation method for large data applications, defines the concept of abnormal data and a variety of random distributions of data. At the same time, this paper proposes the corresponding relationship between the application characteristics and the abnormal data, as well as proposes the generation rules of abnormal data for particular application. Secondly, this paper presents a combination test method, and uses greedy algorithm for the combination of system and application configurations; at the same time proposes probe parameter verification method, through the exponential growth slow start mode to determine the value of the parameters that the worst resource occupies.

This paper introduces the design of reliability benchmarking in detail and implements it on Spark system. By testing the benchmark applications on Spark, three types of errors (out of memory errors, IO exceptions, and task timeouts) were found in six applications, and the abnormal data generation and configuration combination test presented in this paper were also validated availability.

**Keywords:** reliability, benchmark, error, large data system, large data application

目录

[第一章. 绪论 1](#_Toc479627535)

[1.1 研究背景 1](#_Toc479627536)

[1.2 研究内容 3](#_Toc479627537)

[1.3 论文组织 4](#_Toc479627538)

[第二章. 大数据应用可靠性相关工作 5](#_Toc479627539)

[2.1 大数据系统及应用 5](#_Toc479627540)

[2.1.1 大数据系统 5](#_Toc479627541)

[2.1.2 大数据应用 8](#_Toc479627542)

[2.2 可靠性问题研究现状 9](#_Toc479627543)

[2.3 测试基准框架研究现状 10](#_Toc479627544)

[2.4 小结 12](#_Toc479627545)

[第三章. 可靠性测试框架设计 13](#_Toc479627546)

[3.1 设计需求及框架组成 13](#_Toc479627547)

[3.1.1 设计需求 13](#_Toc479627548)

[3.1.2 框架组成 15](#_Toc479627549)

[3.2 典型应用选取 18](#_Toc479627550)

[3.2.1 应用类型 18](#_Toc479627551)

[3.2.2 工作负载 18](#_Toc479627552)

[3.3 异常数据生成 22](#_Toc479627553)

[3.3.1 数据格式 22](#_Toc479627554)

[3.3.2 生成方式 23](#_Toc479627555)

[3.4 组合参数测试 24](#_Toc479627556)

[3.4.1 配置参数 24](#_Toc479627557)

[3.4.2 组合测试 25](#_Toc479627558)

[3.5 测试报告生成 27](#_Toc479627559)

[第四章. 可靠性测试关键技术 29](#_Toc479627560)

[4.1 基于应用特征分析的异常数据生成方法 29](#_Toc479627561)

[4.1.1 应用特征分析 29](#_Toc479627562)

[4.1.2 数据分布方式 30](#_Toc479627563)

[4.1.3 异常数据生成 32](#_Toc479627564)

[4.2 基于贪心算法的参数组合空间削减方法 37](#_Toc479627565)

[4.2.1 组合测试 37](#_Toc479627566)

[4.2.2 参数组合空间削减 39](#_Toc479627567)

[第五章. 可靠性测试框架实现 44](#_Toc479627568)

[5.1 系统架构 44](#_Toc479627569)

[5.2 系统设计与实现 45](#_Toc479627570)

[5.2.1 Web模块设计与实现 46](#_Toc479627571)

[5.2.2 基准模块设计与实现 50](#_Toc479627572)

[5.3 可靠性测试应用验证 59](#_Toc479627573)

[5.3.1 实验环境 59](#_Toc479627574)

[5.3.2 参数配置 59](#_Toc479627575)

[5.3.3 实例及分析 60](#_Toc479627576)

[第六章. 结束语 66](#_Toc479627577)

[6.1 论文贡献 66](#_Toc479627578)

[6.2 未来工作展望 66](#_Toc479627579)

[参考文献 68](#_Toc479627580)

[发表文章 72](#_Toc479627581)

[致谢 74](#_Toc479627582)

# 绪论

本章介绍论文的研究背景、主要工作和论文组织方式。

## 研究背景

随着互联网、移动互联网、物联网的发展，越来越多的领域产生了海量和高速的数据[1]。例如，在金融领域，其日常运营过程中会产生大量的数据，这些数据产生速度快，并且时效性短[2]。又如，在移动通信领域，一个大型城市中每分钟都会有超过8万条的位置更新数据，每天的网络承载流量高达100TB[1]。这些数据具有数据规模大，产生速度快的特点，而且在这些数据中也隐藏着巨大的价值。在金融领域，通过对这些海量数据进行计算和分析，发现隐藏在其中的特征，可以帮助金融行业进行实时决策，从而更好得进行风险管理以及实现商业智能化。在移动通信领域，如果可以对这些海量数据进行实时的挖掘分析，可以降低电话诈骗造成的损失。因此，对大数据应用场景下的海量数据的处理分析已经成为一个迫切的需求。

为了应对海量数据以及发现其背后隐藏的巨大价值，许多大数据系统及大数据应用应运而生。常见的大数据系统有Apache Storm[4]、Apache Hadoop[5]、Apache Spark[6]、Apache Flink[7]等。这些分布式处理框架被广泛应用于社交网络、搜索引擎、数据采集及数据查询等应用场景中，而这些应用场景又衍生出了一系列用于处理特定领域的大数据应用。常见的大数据应用有SQL查询、大规模图分析、机器学习以及流式应用等。然而，现有的大数据应用在处理数据时，经常出现内存溢出、I/O异常、任务超时等运行时错误，以及在流处理过程中的计算结果错误等。这些可靠性的问题将会直接造成应用执行失败，甚至产生更加严重的后果。

ISO 9126[50][51]定义了六个质量特性，Reliability（可靠性）就是其中之一，并且包含了以下子特性：Maturity，Fault Tolerance，Recoverability。同时定义可靠性为[52]“软件产品在规定的时间内、在规定的条件下，保持其性能水平的能力”（”The capability of the software product to maintain its level of performance under stated conditions for a stated period of time.”）。IEEE 610.12-1990[28][53]将可靠性定义为“系统或组件在规定的时间内、在规定的条件下，执行其所需功能的能力”（“The ability of a system or component to perform its required functions under stated conditions for a specified period of time.”）。同时根据《Software Engineering A practioner’s Approach》[55]一书中提出的对可靠性的看法，本文总结可靠性为以下三个方面：系统故障频率（Frequency of system failure）、无错系统功能（Error-free system functions）和存在故障或软件错误（Presence of faults/software bugs）。对于大多数的项目和软件开发者来说，可靠性等于正确性[28]，即它们正在测试并发现和修复 “bug”的数量。然而，在大数据系统测试环境下，本文主要关注的是系统是否存在故障或软件错误，而不是系统故障频率。

在上述基础上，本文将可靠性定义为“在给定应用（包括数据、代码、参数配置等）的情况下，大数据系统在指定的数据和参数配置下，系统是否存在运行时错误、数据完整性或计算结果错误等故障或软件错误”。在软件运行的可靠性测试中，已有的方法通常使用平均无故障时间（MTBF）和平均失效时间（MTTF）来衡量系统的可靠性[28]。然而，本文强调的是系统在特定配置下能否正常运行，即是否会发生执行出错或得不到正确结果的情况。Gillies等人[54]也指出，由于每个系统以及系统的不同部分的目的是不同的，因此评估可靠性的方式也会有变化。因此，本文将以下故障或软件错误称为系统不可靠：（1）性能出现异常。如，用户给定的时间内，系统出现无响应、假死等现象。（2）资源使用异常。如，出现I/O异常、内存溢出、磁盘异常等现象。（3）计算结果异常。如，计算结果错误、计算数据丢失或重复计算等现象。

通过开源论坛、社区、以及相关论文的研究发现，本文分别分析了产生可靠性问题的原因。（1）首先，对于I/O异常、内存溢出及任务超时等运行时错误，其产生原因[8][9][10]包括不恰当的配置参数（输入的数据块过大，分区数目过小，分区函数不均衡等）、异常数据（数据维度过高、数据倾斜等）以及用户代码缺陷（内存泄漏、较高的时间或空间复杂度等）等。（2）其次，对于数据错误（如数据丢失、数据重复等）以及状态错误（状态丢失、快照错误等）等数据和计算完整性问题，根据博客、论坛等的实证分析[11][12][13]，发现其产生原因主要有数据流速过快、Task失效以及快照恢复机制不完善等。在上述分析的基础上，本文将运行时错误以及计算结果错误等问题产生的原因总结为以下三点：（1）系统错误，包括硬件错误（如CPU、内存、网络以及硬盘等错误）和软件错误（如设计缺陷、逻辑错误、实现缺陷等）。（2）应用错误，如参数配置不当或代码缺陷等。（3）数据异常，如数据维度过高或数据倾斜等。

对于用户来说，他们希望了解部署好的应用是否存在潜在的应用错误；对于系统设计者和管理者来说，他们希望了解部署的或更新的系统是否有潜在的系统错误。目前，针对这些应用错误和系统错误的解决方案[14][15]通常是在出现问题之后，再针对某一类应用及问题进行分析诊断。通过测试来发现错误是一种常用的方法，然而当前还没有一个通用的检测方法能够提前发现系统、应用和数据存在的潜在问题。

通过调研发现，现有的大数据平台测试基准，如HiBench[16]、BigSQL benchmark[17]、Spark-perf[18]、Graphalytics[19]以及SparkBench[20]等，关注的焦点通常是特定的大数据平台上的性能或扩展性的基准测试；并且，主要使用常规的输入数据（固定的真实数据集或其简单合成）以及固定的参数配置进行测试。这些测试基准都不能直接用于检测潜在错误，因而对于大数据系统的可靠性测试，目前还没有一套类似的测试基准提供支持。

大数据系统在处理海量和高速数据时，是否可以高可靠的应对高负载的场景已经成为一个亟待研究的课题。针对大数据系统面临的可靠性问题以及现有测试基准的不足，本文将研究如何开发一个针对大数据系统应用的可靠性测试基准框架。该框架的系统目标主要包括以下几点：

1. 构建一个大数据系统的可靠性测试基准，选取SQL查询、大规模图分析以及机器学习应用领域中使用广泛的典型应用作为基准应用；提供包含异常数据的丰富多样的测试数据；提供支持参数组合的测试。
2. 集成数据生成、组合测试以及测试报告生成模块，构建一站式可靠性测试基准平台，将测试人员从测试脚本编写以及手动执行测试等繁琐工作中抽离出来。
3. 提供基于Web的可视化界面，可配置的数据生成以及参数组合测试界面，降低测试人员的使用难度，加快测试速度；自动生成的测试报告，降低分析人员的分析成本。

## 研究内容

针对大数据系统在特定的应用领域面临的运行时错误等可靠性问题，本文主要研究如何通过提前测试的方法来发现系统、应用和数据存在的潜在问题。根据现有测试基准存在的不足，本文将研究如何设计并实现一个大数据系统应用可靠性测试基准框架。

首先，可靠性测试基准需要选取SQL查询、大规模图分析以及机器学习等应用领域中使用广泛并且具有计算特征的典型应用作为基准应用。其次，针对这些基准应用，可靠性测试基准需要提供测试数据；为了检测潜在的应用错误或系统错误，测试数据除了提供常规数据（通常使用的、按照一般规则生成的数据），还应根据应用计算特性生成符合异常数据特征的异常输入数据。最后，在使用测试数据对基准应用进行测试过程中，可靠性测试基准需要提供参数组合测试来检测极端配置参数可能会造成的错误；考虑到组合测试中参数组合空间过大的问题，可靠性测试基准需要基于参数特征减少参数组合空间。

根据上述研究内容，该大数据系统可靠性测试基准框架开发中的关键技术，主要包括以下几点：

1. 提出了针对大数据系统应用的数据生成方法，提供异常数据的生成。首先定义了异常数据的概念，并通过分析应用程序特征，给出了特定应用的异常数据的生成规则，进而生成相应的异常数据（如分布异常、维度过高等特征）。
2. 提出了一种针对大数据系统应用的参数组合测试方法。通过参数组合，并应用贪心算法对系统和应用参数进行组合空间削减。同时，通过指数增长的慢启动方式对参数相关性进行探测性验证，从而确定最差的资源占用的参数组合情况。

最后，论文介绍了大数据系统的可靠性测试基准框架的实现，并将该可靠性测试基准框架应用在Spark系统上。通过对Spark系统中的基准应用进行可靠性测试，已经在6个应用中发现了三种类型的错误（如，内存溢出错误、运行超时以及计算结果错误），具体问题如下：（1）在小表内连接大表时，如果提供倾斜数据会出现内存溢出错误；（2）在一个表同时参与多次join操作但不重新命名时，Spark会将列名弄混导致结果出错；（3）RandomForest在数据维度高、多实例数、数据混合分布时，会出现内存溢出的错误；（4）LogisticsRegression在数据规模较大、数据维度高、以及倾斜数据时，会出现内存溢出或运行超时的错误；（5）ALS在数据规模较大以及迭代次数多时，会出现内存溢出错误；（6）PageRank在数据规模较大以及数据倾斜时，会出现内存溢出错误。同时，通过这些错误验证了可靠性测试基准的异常数据生成方法以及组合参数测试方法的可用性。

## 论文组织

论文的后续章节组织方式如下：

第二章介绍大数据系统及应用可靠性相关工作的研究。介绍了大数据系统和大数据应用的研究现状、大数据系统及应用面临的可靠性问题、测试基准框架的研究现状，并结合现有工作分析了可靠性测试基准框架的需求。

第三章介绍可靠性测试基准的设计。首先，给出了基准设计需求；然后，介绍了基准应用的选取；接着，介绍了测试数据的设计；最后，介绍了基准执行。

第四章介绍可靠性测试基准的关键技术。首先，定义了异常数据概念以及数据概率分布形式，并介绍了异常数据生成方法；然后，介绍了参数配置，组合测试以及参数组合空间削减策略。

第五章介绍大数据应用可靠性测试框架实现。介绍了系统架构，并分别从Web模块和基准模块介绍了系统的设计和实现；最后，对实现的可靠性测试基准框架进行应用验证，介绍了实验环境、参数配置和可靠性测试的实例及分析。

第六章总结了全文，主要包括论文贡献以及对未来工作的展望。

# 大数据可靠性相关工作

本章介绍了大数据系统及应用的发展现状，以及大数据系统应用面临的可靠性问题，最后介绍了测试基准框架的研究现状。

## 大数据系统及应用

随着社交网络、移动互联网、电子商务等技术的不断发展，以及云计算、物联网、互联网+等新兴技术的兴起，全球数据量正在急剧增长，大数据已经成为了当今时代的主旋律。这些网络通信、电子设备以及个人消费等产生的大数据信息，对于企业进行风险管理、分析用户需求以及政府管理交通治安等，都有着极大的帮助。为了处理和分析这些大数据，同时挖掘其背后隐藏的巨大价值，许多大数据系统和大数据应用迅速地发展起来。

### 大数据系统

目前，大数据的处理模式主要包括对静态数据的批量处理以及在线数据的实时处理[21]。因此，现有的大数据处理系统主要分为了批处理系统、流处理系统以及混合处理系统。

* 批处理系统

当前比较流行的批处理系统有Apache Hadoop[5]等。Hadoop是一种以MapReduce[3]作为处理引擎、以HDFS作为存储的批处理框架。MapReduce是一种分布式的编程模型，用于处理和生成大型数据集，并通过共享大规模系统集群，提供良好的可伸缩性。MapReduce将分布式编程分为了map和reduce两个阶段：（1）map阶段，用户通过指定*map()*函数来处理键/值对，以生成一组作为中间结果的key/value对；（2）reduce阶段，通过*reduce()*函数，将中间结果key值相同的key/value对合并。HDFS是一种提供可扩展的、可靠的、用于数据存储的分布式文件系统，通过使用大量低配置、低成本的服务器代替高配置、高成本的大型单机服务器，以及通过key/value对代替关系表[22]，为MapReduce处理产生的大型数据集提供了可靠的存储策略。

* 流处理系统

当前使用比较广泛的流处理系统有Apache Storm[23]等。Storm具有高度可扩展性，易于使用，并提供低延迟、有保证的数据处理，因此被广泛的应用于数据的实时处理中。Storm支持开发人员以任何编程语言实现他们的逻辑，同时还具有以下特点：（1）编程模型简单，Storm提供了一个非常简单的、类似于MapReduce的操作方式来提供实时处理。（2）快速可靠的消息处理，Storm提供“至少一次”的语义保障，一旦任务失败，可以从数据源恢复数据。（3）良好的扩展性，Storm提供多线程并发的进行流式计算，有良好的水平扩展的能力。

* 混合处理系统

当前应用较广的混合处理系统有Apache Spark[24]、Apache Flink[7]等。Spark和Flink同时提供了批处理和流处理两种大数据处理模式，并提供了更加丰富的生态系统，如图2-1和2-2所示。

Spark是一个基于内存的可扩展的大数据处理系统，它引入了一个称为弹性分布式数据集（RDDs）的抽象概念。Spark利用内存进行快速的数据计算，同时具有快速查询的优势，弥补了MapReduce中网络延迟及磁盘I/O过重的劣势[25]，这使得Spark在机器学习等应用中要比Hadoop快10到100倍。同时Spark Streaming可以亚秒级增量地对流进行缓冲，然后将这些缓冲作为小规模的固定数据集进行批处理[26]。这种使用微批处理的方式来处理流式数据的缓冲机制，提高了系统的吞吐率，但同时也带来了较高的延迟。因此，Spark Streaming并不适合处理对延迟要求过高的流处理场景。



图 2-1 Spark生态系统

Flink也是基于内存的大数据处理框架，与Spark不同的是，Flink是一种支持批处理任务的流式数据处理框架。Flink基于同一个Flink运行时（Flink Runtime），提供流处理和批处理两种应用功能的支持，分别使用DataStream API和DataSet API。Flink提供了自己的内存管理，而不是完全依赖于JVM的内存管理机制，可以避免频繁GC（垃圾回收）带来的性能波动。Flink通过构建的DAG（有向无环图）来执行任务，其提供的Logical Plan可以实现自动优化迭代，同时在数据流运行时，实现了高吞吐速率和低延迟。



图 2-2 Flink生态系统

图2-3和2-4分别展示了Flink和Spark的计算迭代流程。Flink提出了增量迭代的计算模型，在迭代中可以显著地减少计算。因此，Flink比Spark的流处理速度更快，内存利用率更高，更加适合流式计算以及迭代计算。



图 2-3 Flink计算迭代流程



图 2-4 Spark计算迭代流程

针对上述介绍的大数据系统，本文针对大数据系统的处理类型、计算模式、处理延迟以及处理速度等进行了总结，如表2-1。

表 2-1 大数据系统对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **大数据系统** | **处理类型** | **计算模式** | **处理延迟** | **处理速度** | **成熟度** |
| **Hadoop** | 批处理 | 基于磁盘 | 高 | 慢 | 强 |
| **Storm** | 流处理 | 基于磁盘 | 低 | 快 | 强 |
| **Spark** | 批/流处理 | 基于内存 | 中 | 快 | 强 |
| **Flink** | 批/流处理 | 基于内存 | 低 | 快 | 中 |

在上述大数据系统中，Spark和Flink不仅提供了丰富的处理类型，具有更加丰富的生态系统，同时针对机器学习、交互式查询、大规模图计算等应用场景提供了良好的支持。Spark发展迅速，社区活跃，并广泛应用在了工业界；Flink以其高效的内存管理和增量迭代，在流式计算中有着很强的优势，并且也在飞速地发展着。对于被广泛应用于工业界的大数据系统，其可靠性更应引起广泛的关注。

### 大数据应用

互联网、云计算、物联网等应用场景中都会产生大量数据。这些应用场景主要包括社交网络、搜索引擎、数据采集及数据查询等，并根据这些应用场景衍生出了一系列用于处理特定领域的应用类别，如SQL查询、大规模图计算以及机器学习等。

* SQL查询

SQL是一种比较持久的查询语言，目前已经有广泛的工具支持它。基本的SQL查询类型主要包括原子查询、中间结果查询以及复杂查询。另外，Big Data benchmark，TPC-H以及TPC-DS等主流的数据库评测基准提供了更加丰富的SQL处理语句。

目前的大数据处理系统中，Spark提供的Spark SQL以及Flink提供的Table API和SQL都支持SQL查询的应用场景。

* 大规模图分析

当前的社交网络发展越来越迅速，Facebook、微博等建立了大量的社会关系网络，并且可以用图来表示人与人之间相互通信的关系。社交网络中复杂的人际关系形成了一个个具有不同属性的社区，也因此衍生出了基于图的社区发现等算法。在搜索引擎应用场景中，可以用图来表示网页之间的超链接关系。为了衡量特定网页相对于搜索引擎索引中的其他网页的重要程度，Google提出了PageRank算法。在交通领域的应用场景中，可以用图来表示车辆或行人的出行轨迹。收集的位置信息构成的图，可以用来在动态的交通网络中查找最短路径，或进行行程规划等。

社交网络、搜索引擎以及交通领域等产生的数据可以用图来表示其中的关联关系。图中的点和边存在着一定的关联性，为了挖掘其中有价值的信息，需要对图数据进行一系列的操作。这些常见场景中的大规模图数据处理已经成为了大数据系统需要支持的主流应用。目前Spark中的GraphX以及Flink中的GElly等都提供了对图算法的支持。

* 机器学习

机器学习[27]是一门人工智能的科学，通常可以分为监督学习、无监督学习和增强学习三类。（1）监督学习是从给定的训练集中学习出一个函数，当有新的数据到来时，可以根据这个函数预测结果。监督学习的训练集包括特征和目标，并且其中的目标是由人标注的。常见的监督学习有回归分析、统计分类等算法。（2）无监督学习中，训练集没有人为标注。常见的无监督学习包括[聚类](https://zh.wikipedia.org/wiki/%E8%81%9A%E7%B1%BB)、降维等算法。（3）增强学习通过观察来学习动作，是一种试错学习。由于没有直接的指导信息，因此，学习对象需要不断与环境进行交互，通过试错的方式做出判断。

目前，Spark中的MLlib以及Flink中的ML都对机器学习中应用比较广泛的模型（如聚类、回归等）提供了支持。

* 流式应用

随着信息技术网络化、智能化的发展，以及数据规模大、来源丰富、类型复杂、变化迅速等诸多特征，使得高时效、可扩展的数据处理成为保障大数据系统服务质量的必要条件。当前，以智慧城市、智能装备、智能制造、极端交易等为代表的具有大数据、实时流式处理等特征的数据处理场景大量涌现。

流式应用可以从三个维度进行表征[20]：延迟，吞吐量和状态大小。流式应用对大数据系统的处理速度以及处理延迟都有较高的要求。目前，Spark Streaming以微批的方式处理流式数据，但有一定的处理延迟；Flink提供的DataStream API为流式应用提供了接口支持。

## 可靠性问题研究现状

当前Hadoop、Spark、Flink等分布式处理框架被广泛应用于大数据的处理分析场景中，如Web索引建立、日志挖掘、机器学习、大规模图分析等。在这些处理场景中，大数据系统应用通常可以表示为（input data，configurations，user code）。其中，input data表示输入数据，通常作为数据块存储在分布式文件系统上；configurations表示配置参数，包括系统参数（例如，输入块大小，分区号）和应用参数（例如，K-means的集群k）等；user code表示用户代码，即用户定义的函数（如*map()*，*reduce()*和*join()*等），它们用于处理输入数据或中间结果。

然而，这些应用在处理大数据时，经常会出现内存溢出、I/O异常、任务超时等运行时错误；同时，在流处理过程中会遇到数据完整性和计算结果错误等情况。这些运行时错误以及计算结果错误等会直接造成应用执行失败，甚至更加严重的后果。大数据系统应用的可靠性已经成为学术界和工业界尤为关注的问题。

已有的关于大数据系统的可靠性研究主要有以下几点：

（1）MapReduce应用内存使用问题方面：由Xu[8]等人通过研究123个真实Hadoop和Spark应用的内存溢出错误，发现了内存溢出错误的三大原因：框架暂存的数据量过大、数据流异常以及内存使用密集的用户代码。

（2）大数据在线查询分析错误方面：Li[9]等人研究了250个SCOPE job（运行在微软的Dryad[56]框架之上）的故障错误，发现错误主要原因是未定义的列、错误的数据模式以及不正确的行格式等。其中84.5%的错误是由于数据处理中的缺陷引起的。他们也发现3个内存溢出错误，错误原因是在内存累积了大量的数据（比如一个大表的所有行被存放到内存中）。

（3）大数据系统运行错误方面：Kavulya等人[10]分析了4100个执行失败的Hadoop jobs，这些jobs运行在Yahoo!管理的M45集群。他们发现36%的故障是数组访问越界错误，还有23%的故障是I/O异常。Zhou等人[29]研究了大数据平台在微软中存在的问题，他们发现36％的问题是由系统缺陷引起的，其中2个问题（1％）是内存问题。

（4）系统在云环境中的开发和部署方面：Gunawi等人[30]研究了云上部署的系统（如Hadoop MapReduce、HDFS和HBase等）的3655个issues。他们发现87％的问题是软件故障，而13％的问题是硬件故障。他们还报告了HBase中的1个内存溢出错误（用户在大型数据集上提交查询）和Hadoop文件系统中的1个内存溢出错误（用户并行创建了数千个小文件）。

上述的研究发现表明，这些错误的原因多种多样，既包括系统缺陷（如设计缺陷和实现bugs），也包括数据异常（数据维度过高、数据倾斜等）和应用缺陷（如参数配置不当、代码缺陷等）。目前，针对这些错误的解决方案[14][15]通常是在问题出现之后，再针对某一类应用及问题进行分析诊断。通过测试来发现错误是一种常用的方法，由于运行时错误的多样性和复杂性，当前还没有一个通用的错误检测方法能够提前发现系统、应用和数据的潜在问题。

## 测试基准框架研究现状

当前，常见的针对大数据系统的测试基准有BigBench[31]、BigDataBench[32]、HiBench[16]和SparkBench[20]等；针对特定的大数据应用的测试基准包括Spark SQL benchmark[17]（SQL查询）、Graphalytics[19]（大规模图分析）、StreamBench[33]（流式应用）等。

* 大数据系统测试基准

BigBench是一种端到端的大数据测试基准，其基本业务模式是产品零售商。该测试基准涵盖了一个数据模型和一个合成数据生成器，针对数据模型的一组查询，生成具有更大数据集、更高速率、更多数据类型（如，来自关系表的结构化数据、半结构化数据以及来自网络点击和社交媒体中的非结构化数据等）的合成数据。BigBench的应用场景比较单一，对大数据系统中的应用类型覆盖不全面。

BigDataBench是针对Web搜索引擎提供大数据测试基准，其相对于BigBench涵盖了更广泛的应用场景，同时提供了多样化和有代表性的真实数据集。该测试基准提供了6个真实数据集以及19个大数据工作负载，并针对不同的数据类型和数据来源提供不同的方法来合成数据。BigDataBench提供的不同类型的工作负载（如社交网络、搜索引擎、关系查询等），分别对应于不同的大数据系统（如Spark、Hadoop、MySQL等）。

HiBench是针对Hadoop系统提出的测试基准。该基准提供了微基准、Web搜索、机器学习等8个工作负载，并从作业运行时间、吞吐量（每分钟完成的任务数）、HDFS带宽以及系统资源（如CPU、内存及I/O等）等方面来评估Hadoop性能。同时，HiBench还为Hadoop的测试提供了非结构化类型的真实数据集。

SparkBench是针对Spark的性能测试工具。BigBench，BigDataBench和HiBench等其他测试基准仅涵盖了少量的Spark支持的应用类型。SparkBench测试基准则覆盖了Spark支持的四种主流的应用类型，包括SQL查询、大规模图计算、机器学习以及流式应用。并且测试中使用到的数据大部分由自带的数据生成器生成。通过对算法或应用的横向对比以及纵向对比测试，从系统资源（如，CPU、Memory等）、任务执行时间、数据处理速度等方面考察Spark的性能。

* 大数据应用测试基准

Pavlo[34]设计了一个SQL基准测试来比较MapReduce和关系数据库之间的性能。Berkeley AMPLab开发了一个SQL基准[17]来比较Spark、Apache Hive[57]以及Apache Impala[58]等的性能。

Graphalytics是专门用于图处理平台的测试基准。该测试基准提供了有代表性的图计算应用，同时使用LDBC社交网络基准（SNB）数据生成器（Datagen）生成的Graphalytics合成数据集。目前，已经测试了比较流行的Giraph，GraphX和Neo4j等图处理平台。

StreamBench是一个用于流处理系统性能比较的基准框架。该测试基准定义了一个公共API组件和一组核心工作负载，并为三种广泛使用的开源流处理系统（Storm、Flink和Spark Streaming）提供基准测试。StreamBench支持定义新的工作负载，同时支持对新的流处理系统进行基准测试。

## 小结

* 测试基准框架存在的问题

表2-2总结了上述介绍的大数据系统及应用的测试基准的特性。通过对这些测试基准框架进行分析，发现现有的测试基准框架主要存在如下问题：

（1）仅支持性能测试

现有的测试基准仅针对大数据系统应用的性能提供了基准测试，没有考虑大数据系统应用存在的可靠性问题。

（2）仅提供常规测试

现有的测试基准通常使用常规数据（通常使用的、按照一般规则生成的数据）以及固定的配置参数来测试大数据系统，缺乏测试的多样性。

表 2-2 测试基准比较

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **测试基准** | **数据集** | **工作负载** | **测试对象** | **测试类型** |
| **BigBench[31]** | 合成数据集 | 关系查询 | DBMS、Hadoop | 性能测试 |
| **BigDataBench[32]** | 真实数据集、合成数据集 | 社交网络、搜索引擎、关系查询 | NoSQL系统、实时分析、在线分析系统 | 性能测试 |
| **HiBench[16]** | 真实数据集 | 微基准、Web搜索、机器学习 | Hadoop、Hive | 性能测试 |
| **SparkBench[20]** | 真实数据集、合成数据集 | SQL查询、图计算、机器学习以及流式应用 | Spark | 性能测试 |
| **Graphalytics[19]** | 真实数据集、合成数据集 | 大规模图计算 | GraphX、Giraph、Neo4j | 性能测试 |
| **StreamBench[33]** | 合成数据集 | 基本操作、join、迭代 | Storm、Flink、Spark Streaming | 性能测试 |

针对大数据系统及应用面临的可靠性问题，以及现有测试基准存在的不足，面向大数据系统及应用的可靠性测试基准框架有很大的研究价值和发展空间。

# 可靠性测试框架设计

为了能够提前发现大数据应用及系统存在的可靠性问题，本章设计了可靠性测试框架。首先，通过实证分析，给出了可靠性测试框架的设计需求，并根据需求给出了框架组成；然后，依次介绍了可靠性测试框架的各个组成部分，包括典型应用选取、异常数据生成、组合参数测试以及测试报告生成。

## 设计需求及框架组成

本节首先通过对论文等的实证分析，从数据、测试以及平台三方面分析可靠性测试框架的设计需求；然后，在此基础上介绍了可靠性测试框架的具体组成。

### 设计需求

表 3-1 实证分析

|  |  |  |  |
| --- | --- | --- | --- |
| **错误** | **内容** | **原因** | **来源** |
| **内存溢出** | Xu等人[8]研究了123个真实Hadoop和Spark应用的内存溢出错误。 | 框架暂存的数据量过大，数据流异常以及内存使用密集的用户代码。 | 论文[8] |
| Logistics Regression运行时迭代次数的增加。 | 配置参数迭代次数的设置过大会导致内存溢出。 | Spark user list[62] |
| **I/O异常** | Kavulya等人[10]通过分析4100个执行失败的Hadoop jobs，发现23%的故障是IO异常。 | Zhou等人[29]研究了大数据平台在微软中存在的问题，他们发现36％的问题是由系统缺陷引起的。 | 论文[10] [29] |
| **运行超时** | Li等人[9]研究了250个SCOPE job（运行在微软的Dryad框架之上）的故障。 | 发现错误主要原因是未定义的列，错误的数据模式，不正确的行格式等。 | 论文[9] |
| **计算结果**  **错误** | Spark SQL中，一个表同时参与多次join操作。 | 当一个表同时参与多次join操作但不重新命名时，Spark会将列名弄混导致结果错误。 | Spark-11576[60] |
| **GC** | Logistic Regression中aggregator序列化不必要数据。 | 由于大量的序列化不必要的数据，将导致性能问题 | Spark-16008[61] |

通过对issues的实证分析以及对相关论文的研究，表3-1中总结了现有大数据应用中存在的可靠性问题以及出现这些问题的原因。通过上述研究表明，大数据应用出现可靠性问题的原因多种多样。I/O异常、内存溢出、运行超时等运行时错误，主要是由异常数据（如，数据格式错误、数据维度过高、数据倾斜等）、不恰当的配置参数（如，迭代次数过大、reducer数目太小、决策树深度过大等）以及用户代码缺陷（如，时间或空间复杂度太高、内存泄漏等）等原因造成的。计算结果错误主要是由于系统缺陷（如，系统执行机制等）造成的。

通过对产生可靠性问题的原因进行归纳总结，主要体现为以下几点：

1. 系统缺陷：如设计缺陷和实现缺陷。
2. 数据异常：如数据格式错误、数据倾斜等极端的数据。
3. 应用缺陷：如测试过程中配置参数不当、代码缺陷等。

可靠性测试框架旨在通过对应用提供数据、配置参数来执行测试，以提前发现系统或应用可能存在的可靠性问题。本节接下来将分别从数据、测试以及平台三方面来分析可靠性测试框架的设计需求。

* **数据需求**

测试大数据应用的数据（1）首先，应该尽可能的符合真实场景；（2）其次，数据应该是随机但可预测的，即数据必须足够随机以覆盖更多测试场景，同时数据处理的输出结果必须是可以预测的以允许验证实验结果。这样的数据集可以是有代表性的真实数据集，也可以是符合真实数据特征的、随机合成的数据集。

现有的大数据基准测试的数据生成满足上述需求，但是数据仍然相对的固定、单一化。为了能够发现应用或系统存在的可靠性问题，除了数据集的大小以外，可靠性测试框架在数据生成方面，还应该具备以下特点：

1. 应用相关性：测试数据应该与应用的计算特性相关，那些可能会影响到应用的时间或空间复杂度的数据，容易使应用在计算数据过程中出现错误。如，以矩阵为输入时，矩阵过于稀疏可能会造成空间的浪费。
2. 分布多样性：测试数据还应满足某一种或某几种概率分布，以模拟具有更复杂特征的真实数据（如，数据高度倾斜等）。
3. 数据多样性：除了数据集的大小、数据的分布等数据特征外，还应考虑输入数据的维度、稀疏度以及倾斜程度等。

* **测试需求**

现有的大数据基准测试通常使用固定的参数配置来执行测试，以观察大数据系统在某种配置下的性能或者扩展性。然而，测试过程中参数配置不当会导致应用出现错误，如表3-1中提到的，使用迭代模型的应用在迭代次数过大的情况下，可能会出现内存溢出的错误。另外，很多大数据应用（如，机器学习等），测试时需要提供多个应用参数，这些参数的相互作用可能会导致应用出错。因此，可靠性测试框架在测试方面，还需要具有以下特点：

1. 参数组合：可靠性测试需要对应用的参数配置重点关注，以发现可能存在的、导致应用出错的参数或参数组合。
2. 高效性：可靠性测试不同于性能测试以及扩展性测试，因为即使经过了很多次的测试，仍不能保证可以发现错误的存在。因此，可靠性测试需要能够高效的避免过多不必要测试的浪费。

* **平台需求**

Huppler[35]提出，一个成功的测试框架需要具备的属性有相关性、可重复性、公平性、可验证性、经济性等。随着大数据系统的迅速发展，Agrawal[20]等人补充测试框架的新属性有如下几点：

（1）简单易用。测试框架需要易于理解和部署，并能通过适度的配置便可自动化地执行和分析。

（2）应用多样性。测试框架需要满足大数据系统中应用类型的多样性，通过选择不同操作类型的有代表性的应用来体现大数据系统操作的多样性。

（3）可移植性。测试框架需要适配更广泛的大数据系统，能够为更多的大数据系统提供测试。

（4）可扩展性。测试框架需要满足大型分布式系统的测试需求，能够生成数据量大且变化多样的数据集。

### 框架组成

本文在上一节中，通过对论文以及问题等的实证分析，分别从数据、测试以及平台三方面分析了可靠性测试框架的设计需求。接下来，本节将从相关的两个概念出发，进而介绍可靠性测试框架的具体组成，同时对测试的执行流程就行了说明。

经上一节的研究发现，大数据应用中的错误一般是在极端配置、极端数据下产生的。所以，可靠性测试框架需要产生一些极端的异常数据，以及提供一些不恰当的参数配置组合，来对系统进行测试，以使系统尽可能的在高压力、高负载下提前暴露问题。这就需要考虑两个问题：（1）什么是异常数据？（2）测试需要哪些配置参数进行组合？

* **异常数据**

首先，什么是异常数据呢？本文把具有以下特点的数据称为异常数据：**数据量大、数据倾斜、数据稀疏、数据维度高、数据分布异常**。

1. 数据量大：即数据规模巨大，在当前应用配置下无法正常应对；
2. 数据稀疏：无用元素过多，即增加数据信息量的元素很多（如矩阵中的0）；
3. 数据维度高：即用于测试的数据特征过多；
4. 数据分布异常：数据分布不均匀，呈现高斯分布、伽马分布、泊松分布、指数分布、Zipf分布或其混合等分布形式；
5. 数据倾斜：如单个key多次出现，或value值过大。

当数据具有上述数据特征的某一项或某几项时，我们称该数据为异常数据。通常情况下，这些特征是组合出现的。

* **参数组合**

其次，测试需要哪些配置参数进行组合呢？大数据系统在运行应用时，需要的配置参数包括**系统参数**和**应用参数**。其中，（1）系统参数指的是大数据系统运行应用时可能会影响系统数据分配或任务分配等的参数（例如，并行度、划分函数等），这些系统参数将会影响到系统运行时的CPU内核分配以及内存的使用；（2）应用参数指的是应用本身运行时需要的参数（例如，使用迭代模型进行计算的应用，一般需要“最大迭代次数”作为参数），这些应用参数将会直接影响到应用的运行，可能会对应用的结果正确性、时间或空间复杂度等造成影响。

* **框架设计**

接下来，需要考虑另一个更为关键的问题，可靠性测试框架应该对什么样的大数据应用、提供什么样的异常数据、并以哪种参数组合来执行测试，最后得到什么样的测试结果？为了回答上述问题，可靠性测试框架需要包含以下几个组成部分：**典型应用选取**、**异常数据生成**、**组合参数测试**以及**测试报告生成**，如图3-1所示。

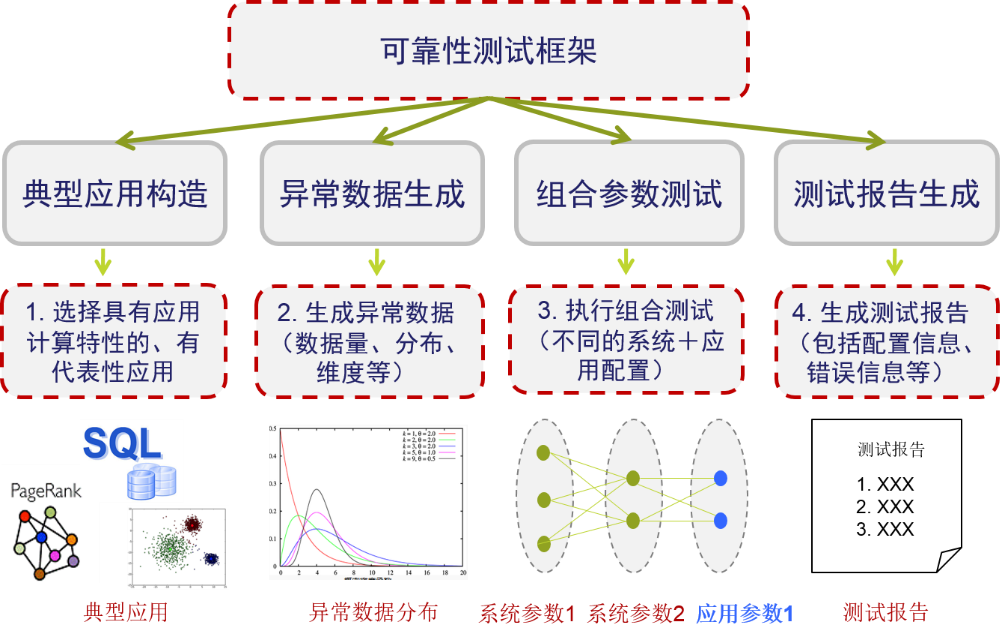
****

图 3-1 可靠性测试框架图

1. 典型应用选取：选取SQL、Graph以及Machine Learning等应用领域中，广泛使用的、有代表性的、并且具有应用计算特征的应用，作为典型应用。
2. 异常数据生成：根据应用计算特征，生成满足特定异常数据特征（数据量大、数据倾斜、数据稀疏、数据维度高、数据分布异常）的异常数据。
3. 组合参数测试：通过组合系统参数与应用参数，提供对典型应用的参数组合测试，同时还需考虑测试空间削减问题。
4. 测试报告生成：提供测试中的配置信息、运行指标以及错误信息，从而生成测试报告，并对大数据应用的可靠性进行简单的分析。

* **测试执行流程**

在上述设计的可靠性测试框架中，测试的具体执行流程如图3-2所示。



图 3-2 测试执行流程

1. 在准备好待测系统以及存储系统的环境后，用户首先配置系统信息，包括待测系统和存储系统的访问路径等信息。
2. 测试环境以及访问路径都准备好后，用户通过配置数据生成需要的参数信息，生成自定义的数据集。例如，在SQL数据生成中，可以配置数据倾斜度来生成倾斜的异常数据。
3. 数据生成结束后，用户选择工作负载，并配置所需的系统参数和应用参数信息。同时使用已经生成的数据集，通过触发脚本执行集群环境下的参数组合测试。
4. 测试完成后，用户可以查看测试报告。

本章接下来将对可靠性测试框架的组成部分进行介绍，并在下一章中就其中的关键技术进行重点介绍。

## 典型应用选取

可靠性测试框架用于对大数据应用进行测试，以发现系统或应用存在的可靠性问题。因此，首先需要选取有代表性的应用。本节将介绍根据计算特性选取的使用广泛且有代表意义的典型应用；同时，介绍了不同应用类型的工作负载。

### 应用类型

现有的大数据应用场景主要包括社交网络、搜索引擎、数据采集及数据查询等，并根据这些应用场景衍生出了一系列用于处理特定领域的典型应用。本文针对这些应用场景，并依据现有的大数据基准测试中提供的应用类型[16,36-39]，选取了SQL查询（SQL）、大规模图分析（Graph）、机器学习（Machine Learning）中的若干应用作为可靠性测试框架的典型应用。表3-2列出了可靠性测试框架提供的典型应用类型及其计算特性。

### 工作负载

本小节针对上述典型应用类型，分别在SQL、Graph、Machine Learning应用类别中，提供了下述具体的工作负载。

* SQL

表 3-2 典型应用及计算特性

|  |  |  |
| --- | --- | --- |
| **类别** | **应用** | **计算特性** |
| **SQL** | Scan | 单表过滤操作 |
| Aggregate | 单表聚合操作 |
| Join | 多表关联操作 |
| Mix | 多表混合操作 |
| **Graph** | PageRank | 迭代计算 |
| TriangleCount |
| ConnectedComponents |
| SingleSourceShortestPaths |
| **Machine Learning** | LogisticsRegression | 分类算法、迭代计算 |
| K-means | 聚类算法、迭代计算 |
| ALS | 交替最小二乘法 |
| RandomForest | 分类/回归、宽度优先树 |
| SVM | 分布式双梯度下降 |

SQL是一种使用广泛的查询语言，并有广泛的大数据生态系统支持。现有的数据库评测基准，如Big Data Benchmark，TPC-H和TPC-DS等，都覆盖了较为丰富的查询语句。这些应用于不同的大数据系统、不同应用领域的SQL查询，通常可以由基本的查询子句构造出来。因此，可靠性测试框架选取的SQL类别中的典型应用为基本查询语句，如表3-2所示。

为了支持上述基础查询语句，本文根据Pavlo[34]等人提出的测试方案设计了两张表，分别为Rankings和UserVisits。其中，Rankings表用于记录网页排名信息，UserVisits表用于记录用户访问网页信息。两张表的属性以及具体含义如表3-3所示。

表 3-3 SQL表结构

|  |  |  |  |
| --- | --- | --- | --- |
| **表名** | **属性名** | **数据类型** | **含义** |
| **Rankings** | pageURL | VARCHAR | 网页URL |
| pageRank | INT | 网页排名 |
| avgDuration | INT | 平均停留时间 |
| **UserVisits** | sourceIP | VARCHAR | 源IP地址 |
| destURL | VARCHAR | 目标URL地址 |
| visitDate | DATE | 访问日期 |
| adRevenue | FLOAT | 广告收入 |
| userAgent | VARCHAR | 用户代理 |
| countryCode | VARCHAR | 城市编码 |
| languageCode | VARCHAR | 语言编码 |
| searchWord | VARCHAR | 搜索关键字 |
| duration | INT | 停留时间 |

以下是本文选取的SQL查询中的工作负载。

Scan

针对单个二维表，提供基本过滤查询，如SELECT *\** FROM *uservisits* WHERE *adRevenue > X;*

Aggregate

针对单个二维表，提供聚合操作，如SELECT *destinationURL*, SUM(*adRevenue*) AS *total* FROM *uservisits* GROUP BY *destinationURL* ORDER BY *total* DESC;

Join

针对多个二维表，提供多表关联操作，如**SELECT** *sourceIP*, *url*, *adRevenue* **FROM** *rankings* **INNER** **JOIN** *uservisits* **ON** *url = destinationURL;*

Mix

针对多个二维表，提供过滤、聚合及关联的混合操作，如SELECT *sourceIP*, *totalRevenue*, *avgPageRank* FROM (SELECT *sourceIP*, AVG(*pageRank*) as *avgPageRank*, SUM(*adRevenue*) as *totalRevenue* FROM *rankings* AS *R*, *uservisits* AS *UV* WHERE *R.url* = *UV.destinationURL* AND *UV.* *avgDuration* BETWEEN 30 AND 70 GROUP BY *UV.sourceIP*) ORDER BY *totalRevenue* DESC LIMIT 10。

针对上述SQL中的典型应用，可靠性测试框架通过提供规模不同、倾斜程度不同的表文件来测试SQL应用的可靠性。

* Graph

社交网络中人与人之间相互通信的关系、交通网络中车辆的行驶轨迹等都可以用图来表示。现有的大数据系统中，Spark的GraphX以及Flink的GElly等都提供了图应用。本文依据现有的大数据系统应用库中提供的图算法，以及现有的大数据基准测试中提供的图应用，选取了如下几种典型应用作为可靠性测试框架的工作负载。

1. PageRank

PageRank是Google提出的在搜索结果中对比网站排名的算法，通常应用于搜索引擎中对搜索结果排序的场景下。该算法通常使用迭代模型计算，例如，使用Vertex-Centric（顶点为中心）的迭代模型进行PageRank计算，图中的每个顶点在每次迭代过程中都会从相邻顶点处接收消息，并在经过计算后，活跃的顶点会将更新后的消息传递给邻居顶点。持续地进行迭代，直到达到最大迭代次数或结果收敛。

1. TriangleCount

TriangleCount用于统计（有向/无向）图中不同三角形的数目。一般运用在社交网络分析中。社交网络中的三角形越多，说明关系网越强。该算法同样使用迭代计算模型。

1. ConnectedComponents

ConnectedComponents用于计算输入的图的连通分量。在一个图中，如果将顶点抽象为变量，顶点之间的连通性抽象为变量间的引用关系，那么构造并查集就可以解决变量的循环引用问题，避免引用环的存在，从而方便引用技术和垃圾回收。同样的，如果将顶点抽象为计算机，顶点之间的连通性抽象为网络，那么动态连接性问题可以减少不必要的布线。

1. SingleSourceShortestPaths

SingleSourceShortestPaths用于计算图中单源最短路径。该工作负载可以评估大数据系统中的图处理框架是否可以有效地处理本地或子图计算。

对Graph中的典型应用进行测试时，首先需要生成图数据到存储系统，然后以该数据为输入进行测试。Graph应用的测试通过提供规模不同或顶点度分布不同的图数据文件来测试Graph应用的可靠性。

* Machine Learning

Machine Learning在大数据处理分析中应用较为广泛，为结果预测、信息推荐等提供了模型和算法支持。目前使用较为广泛的算法模型有回归、分类、聚类等。现有的大数据系统中，Spark的MLlib以及Flink的ML都提供了相应的应用实现。本文依据现有的大数据系统中的机器学习算法模型，以及流行的大数据性能测试基准中提供的机器学习应用，构建了如下工作负载。

1. LogisticsRegression

LogisticsRegression作为机器学习的分类器，可用于预测连续或分类数据。该算法使用随机梯度下降或L-BFGS来训练分类模型。输入的数据集在每次迭代中计算、更新和广播参数向量。

1. K-means

K-means是一种常见的聚类分析算法，使用无监督学习方法，通过迭代不断的选取离中心点最近均值进行聚类分析。

1. ALS

ALS算法将稀疏评分矩阵分解为用户特征向量矩阵和产品特征向量矩阵的乘积，并使用交替最小二乘算法来计算矩阵分解。

1. RandomForest

RandomForest是基于决策树的集成模型，通过对数据随机采样来单独训练每一棵树；通过组合每棵独立树的结果进行预测；通过训练数据来构建一棵回归树，从而对未知数据进行回归预测。

1. SVM

SVM模型使用基于hinge-loss的具有高效通信能力的分布式双梯度上升的方法，通过在高维或无限维空间中构建一组超平面来进行分类训练。与线性和逻辑分类相比，SVM可以将输入隐式映射到高维特征空间，并有效地进行非线性分类。

Machine Learning的可靠性测试通过输入不同的数据集（如，数据规模、数据分布、数据维度等不同），同时针对不同的应用提供不同的参数组合，来进行参数组合测试。

## 异常数据生成

在上一节选取完典型应用后，本节将为这些应用提供测试数据。首先，本节针对SQL、Graph以及Machine Learning三种应用类型，给出了测试数据格式的设计。其次，为了检测潜在的应用错误或系统错误，测试数据除了提供常规数据（即通常使用的、按照一般规则生成的数据），还应根据应用计算特性生成符合异常数据特征的异常输入数据（即满足数据量大、数据倾斜、数据稀疏、数据维度高或数据分布异常等异常数据特征的数据）。本文将在4.1节中详细的介绍基于应用特征分析的异常数据生成方法。最后，本节将针对可靠性测试框架提供的数据的生成方式进行介绍。

### 数据格式

* SQL数据

SQL数据根据3.3.2节中给出的Rankings表和UserVisits表定义的具体模式来生成。Rankings表中的数据格式为（pageURL,pageRank,avgDuration）。UserVisits表中的数据格式为（sourceIP,destURL,visitDate,adRevenue,userAgent,countryCode,

languageCode,searchWord,duration）。其中，visitDate，adRevenue、pageRank和sourceIP等字段从特定范围内生成随机值，pageURL、destURL、countryCode以及languageCode等字段将从真实数据集进行随机选择。这些随机值将根据数据列在查询语句中的作用程度，采用不同的数据分布形式。

SQL的数据生成通过配置数据倾斜度，来生成相应的常规数据或倾斜的异常数据。每条数据之间以换行作为分隔，每个数据文件以文本文件的形式存储在HDFS的节点上。

* Graph数据

Graph数据包括Vertex（顶点）数据集以及Edge（边）数据集。Vertex定义为二元组(vertexID,value)，其中vertexID用来标识顶点的编号，value是顶点的值。Edge定义为三元组(srcID,destID,value)，其中srcID为源顶点ID，destID为目的顶点ID，value为源顶点到目的顶点的边的权重。Vertex中的vertexID采用范围内顺序生成的方式，value为满足某个范围的随机数据。Edge中的srcID和destID需要在Vertex的vertexID范围内生成，其权重value为特定范围内生成的随机值。

Graph的数据生成可以通过配置顶点之间的相遇概率，生成满足不同幂率分布的异常数据。数据文件以文本文件的形式存储在HDFS节点上。

* Machine Learning数据

Machine Learning中不同应用的数据格式有所不同。下面将分别介绍Machine Learning应用类别中的各个典型应用的数据格式。

LogisticsRegression、RandomForest以及SVM都是有监督的学习。这些应用使用带标签的数据，数据的格式为。这种数据格式称为LibSVM Format。其中label表示标签，在分类算法中为离散值（例如，二分类时，label取值为0或1），在回归算法中为连续值。index表示各个特征的序列号，value表示各个特征值。在监督学习中，各个样例不一定包含全部特征，因此数据集通常为稀疏数据。

K-means是无监督学习，使用不带标签的数据。数据的格式为。value表示各个特征值。无监督学习中，每个样例都包含全部特征，因此数据集通常为稠密数据。

ALS数据使用Netflix数据集（电影评分数据集）格式。其中userId为用户编号，itemID为电影编号，rating为电影评分，timestamp为时间戳。userId和itemId为某一范围内的随机值，rating为1~5范围内的随机数，timestamp为日期类型的数据。

Machine Learning的数据生成同样需要配置数据特征参数（如实例数、维度、分布方式等），并根据配置信息生成满足不同维度、不同分布、不同规模的异常数据。这些数据同样存储在HDFS的节点上。

### 生成方式

可靠性测试框架提供的数据生成，除了满足上述数据格式定义以及异常数据特征等数据需求之外，还需要考虑数据的生成方式。

首先，**可扩展性**在测试基准的数据生成中起着至关重要的作用[20]。例如，使用较为广泛的数据库评测基准TPC-H[63]和TPC-DS[64]，其中TPC-H有内置的可扩展的数据生成器，可以高效的生成测试数据；TPC-DS在其基础之上做了进一步的改进，增加了数据的不均匀分布以及数据相关性等。因此，可靠性测试框架提供的数据生成应该具有可扩展性，即使用面向接口式的开发方式，支持具有不同数据格式需求的新应用类型的数据生成。另外，**并发和高效性**是大数据测试中数据生成时同样应该具备的特性。通过分布式集群并发的执行数据生成，能够快速的生成用于大规模集群测试的有代表性的测试数据，并分布到集群中不同的节点上。通过分布式的数据生成，可以提高数据的生成效率。

综合上述分析，大数据应用的可靠性测试框架提供的数据生成使用下述方式，实现高效的数据生成：

（1）提供面向统一接口的数据生成脚本。

（2）使用分布式集群并发的方式执行数据生成脚本。

## 组合参数测试

在具备了典型应用和测试数据后，接下来，本节将首先介绍需要进行组合测试的配置参数，然后再介绍针对典型应用的测试方式。

### 配置参数

大数据系统运行应用的配置参数包括系统参数和应用参数。下面将分别对这两种类型的参数进行介绍。

* 系统参数

系统参数指的是大数据系统运行应用时可能会影响系统数据分配或任务分配等的参数，例如并行度、划分函数等。这些系统参数将会影响到系统运行时的CPU内核分配以及内存的使用。

不同的大数据系统由于运行机制的差异，对系统参数的定义略有不同。例如表3-4所示，Spark使用Input split number（Mapper端分片个数）来确定数据并行度，使用Partition number（Reducer端分片个数）来确定任务并行度。又如表3-5所示，Flink使用NumberofTaskSlots（单个TaskManager可以运行的并行算子数目）以及Parallelism（总并行度）来确定任务并行度。

表 3-4 Spark系统参数

|  |  |
| --- | --- |
| 参数 | 含义 |
| Input split number | 数据并行度 |
| Partition number | 任务并行度 |

表 3-5 Flink系统参数

|  |  |
| --- | --- |
| 参数 | 含义 |
| NumberOfTaskSlots | 单个TaskManager的并行度 |
| Parallelism | 总的任务并行度 |

* 应用参数

应用参数指的是应用本身运行时需要的参数。例如Graph中的大多数应用需要进行迭代计算，因此需要使用“迭代次数”作为参数，这些应用参数将会直接影响到应用的运行，可能会对结果正确性、运行复杂度等造成影响。

由于SQL中的查询操作没有应用参数，因此，表3-6中主要针对Graph以及Machine Learning应用类别中的典型应用，给出了具体的参数列表。

表 3-6 应用参数列表

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **应用** | **参数** | **含义** |
| **Graph** | PageRank | maxIteration | 最大迭代次数 |
| convergenceAccuracy | 收敛精度 |
| ConnectedComponents | maxIteration | 最大迭代次数 |
| SingleSourceShortestPaths | srcVertexId | 源顶点Id |
| maxIteration | 最大迭代次数 |
| **Machine Learning** | LogisticsRegression | numIteration | 最大迭代次数 |
| numRegparam | 正则参数 |
| convergencoTol | 误差容忍度 |
| numCorrection | 在LBFGS更新中使用的校正的数量 |
| numClasses | 分类数 |
| K-means | itermin | 最小迭代次数 |
| itermax | 最大迭代次数 |
| initializedMode | 选择初始聚类中心的方式 |
| RandomForest | numTrees | 森林中树的个数 |
| maxDepth | 最大树深 |
| maxBins | 最大分箱数 |
| numClasses | 分类数 |
| ALS | dataFeature | 特征数量 |
| sampling | 是否为放回采样 |
| sampleFraction | 采样率 |
| numIteration | 最大迭代次数 |
| SVM | features | 特征数量 |
| sampleFraction | 采样率 |
| seed | 采样种子 |
| numIteration | 最大迭代次数 |

### 组合测试

本小节接下来将介绍可靠性测试框架使用的测试方式以及在测试中遇到的问题，而具体的解决方案将在4.2节中详细介绍。

极端的配置参数可能会导致错误的发生，而在有多个参数（包括系统参数和应用参数）的情况下，极端的参数组合起来进行测试会加剧错误的发生。因此，针对3.4.1节中介绍的配置参数，需要对这些系统参数和应用参数进行组合测试，来覆盖更大范围的测试用例，尽可能多地发现其中的可靠性问题。

组合测试[43]使用由某些抽样机制生成的覆盖数组来检测待测系统（SUT）在参数组合作用下触发的错误。通过组合测试，测试人员可以在确保错误检出率的前提下，使用较少的测试用例来测试系统[44]。通常，组合测试用例集可以通过矩阵来表示。其中矩阵的一行代表一个测试用例；矩阵中的一列代表一个参数；矩阵中的一项代表参数的取值。据此，Cehen等人[45]给出了覆盖数组的概念，用来描述组合测试用例集。下面给出了**覆盖数组**的定义。

如果一个的数组具有以下属性：

（1）每个列仅包含来自集合的元素，其中；

（2）每个子数组的行覆盖来自个列的所有元组至少一次；

则称其为强度为的覆盖数组（或混合强度覆盖数组（MCA）），记为。覆盖数是满足参数，n和所需的最小N。当时，覆盖数组被记为，其覆盖数为。

传统上的组合测试是使用约束求解或最优化的方法，通过搜索覆盖数组来实现的。然而，在使用覆盖数组的组合测试中，任意的个参数所有的组合都需要被覆盖住，因此测试用例数仍然很多。

* **组合空间爆炸问题**

对于一个具有*n*个配置参数的待测系统（SUT），假设这些参数分别有*m1*,*m2*,…*mn*个可选值，那么完成所有组合来测试这个系统需要的参数组合空间为：

在上述对所有参数进行组合测试的方式中，当参数种类过多或单个参数的取值范围过大时，这个组合数将是一个非常庞大的数字，容易产生组合空间爆炸的问题。因此，**如何削减测试的参数组合空间，是可靠性测试中需要解决的一个关键问题，同时也是降低测试开销的重要手段**。

可靠性测试框架应该如何解决组合测试过程中遇到的参数组合空间爆炸的问题呢？本文将在4.2节中详细的介绍基于贪心算法的参数组合空间削减方法。该方法首先分析了参数的相关性，并在两个假设的基础上，提出了组合空间削减算法，同时针对不符合假设的条件提出了探测性参数验证方法。

## 测试报告生成

在选定待测应用类型，并给定测试数据以及参数配置后，可靠性测试框架执行参数组合测试，并在执行结束之后，提供测试报告。

测试报告的主要内容包括：

1. 配置信息：如应用名称、测试数据、系统参数以及系统参数等。
2. 运行指标：应用运行过程中使用的最大内存、CPU等资源占用情况。
3. 错误信息：应用运行过程中是否出现异常。

通过上述报告内容，可靠性测试框架给出测试结果以及简单的测试分析。测试人员可以从测试报告中分析得出：

1. 如果出现了异常，可以得到该应用出现错误的数据集以及具体的参数配置信息。
2. 如果没有发现异常，可以得到能够使该应用具有最差资源使用情况下的数据集和参数配置信息。

测试人员可以根据测试报告找到出现问题的配置参数组合，以及具有异常特征的数据集。那么在系统上线后的应用执行过程中，可以通过加强输入数据格式检测、避免极端配置参数组合等方式，尽可能的避免错误的出现。

## 小结

本章通过实证分析，从数据、测试以及平台三个角度分析了可靠性测试框架的设计需求。在此需求基础上，本文介绍了可靠性测试框架的四个组成部分：（1）典型应用选取：选取SQL、Graph以及Machine Learning中使用广泛并且具有一定计算特性的应用作为典型应用。（2）异常数据生成：定义了不同应用类型的数据格式，并通过分布式集群并发执行数据生成。异常数据生成的具体方法将在下一章中具体介绍。（3）组合参数测试：对测试中使用到的系统参数和应用参数进行了汇总，并给出了组合测试方法。组合测试中遇到的组合空间爆炸问题将在下一章中具体介绍。（4）测试报告生成：对测试过程中的信息收集汇总，在测试结束后，展示配置信息、运行指标、错误信息等内容，同时可以简单的分析出应用出现错误或资源使用最差情况下的数据集和参数组合情况。

# 可靠性测试关键技术

本章就可靠性测试框架中的关键技术及方法进行研究。首先，研究了基于应用特征分析的异常数据生成方法，该方法通过分析典型应用的计算特性，并结合第三章中提出的异常数据特征，给出了不同应用的异常数据生成规则；其次，研究了基于贪心算法的参数组合空间削减方法，该方法首先对应用参数进行分析，在给定的两个假设前提下，提出了一种类似于贪心算法的空间削减策略和一种探测性参数验证方法。

## 基于应用特征分析的异常数据生成方法

本节首先对典型应用进行应用特征分析，并定义了相应的异常数据生成规则；然后，介绍了数据生成中使用到的数据分布方式；最后，根据异常数据规则生成满足不同异常特征的异常数据。

### 应用特征分析

可靠性测试的目的是发现应用或系统中潜在的问题。因此，测试数据应该尽可能的与应用的计算特性相关，那些分布异常或影响应用计算复杂度的数据，容易使应用在计算过程中出现错误。

为了能够使生成的数据更加贴近应用本身的计算特性，本文针对可靠性测试框架提供的典型应用，首先抽取其数据操作特征（如操作顺序、依赖关系等）。

* **数据操作特征抽取**

SQL查询中的基本操作语句在处理key/value对时，其计算复杂度与key的分布相关。当key分布不均匀（如，一个key出现的次数过多）时，会影响SQL查询的空间或时间复杂度。因此，SQL的数据生成需要考虑数据倾斜带来的key值分布不均匀的问题。

Graph中的应用通常需要迭代计算。例如，PageRank需要由顶点向相邻顶点发送其贡献值，当出现数据倾斜时（即某一顶点的相邻顶点格外多），对单个顶点的压力过大，容易出现内存溢出等可靠性问题。又如，TriangleCount在有重复的边及指向自己的环的情况下，会导致计算结果不正确。因此，Graph的数据生成需要考虑数据稀疏性带来的顶点度分布异常的问题，同时还需要考虑数据重复性可能造成的计算结果错误。

Machine Learning中的应用中，有些需要进行迭代计算，有些需要生成宽度优先树。例如，LogisticsRegression、K-means等应用的特征是迭代，且以矩阵作为输入数据，因此需要考虑矩阵特征对应用带来的影响；同时，矩阵特征会在迭代中表现的更为明显。又如，RandomForest等应用在计算过程中，需要保存宽度优先树，当数据维度过高时，每个节点存储的信息也会相应增多，容易出现内存溢出等可靠性问题。因此，Machine Learning的数据生成需要考虑数据维度、数据稀疏性等带来的内存占用问题。

* **异常规则选取**

通过对上述对各类应用的数据操作特征抽取，接下来，需要根据这些特征进行异常规则选取。本文总结应用的计算特性及其对应的异常规则，如表4-1所示。

表 4-1 应用特征分析

|  |  |  |
| --- | --- | --- |
| **应用类型** | **计算特性** | **异常规则** |
| **SQL** | SQL基础查询语句中Scan、Aggregate、Join等应用处理key/value对时，其计算复杂度与key的分布相关。 | 数据量大、数据倾斜 |
| **Graph** | Graph中的应用通常需要迭代计算，在以顶点为中心的迭代模型中，如果顶点收集消息阶段有很重的操作时，单个顶点的计算压力会增大。 | 数据量大、数据稀疏、数据分布异常 |
| **Machine Learning** | Machine Learning中的应用，如LogisticRegression和K-means等采用矩阵征作为输入数据，因此其计算与矩阵特征（1）矩阵总大小，（2）矩阵维度，（3）每个矩阵列的分布，（4）矩阵稀疏性，等有关系。  其他的应用，如RandomForest等，需要在内存中保存宽度优先树，并使用随机采样来训练树。当数据维度过高时，资源利用率也高。 | 数据量大、数据稀疏、数据维度高、数据分布异常 |

根据上述分析以及总结的异常数据规则，发现数据分布是影响应用执行的一个关键的因素。由于数据稀疏或数据倾斜造成的数据分布异常能够影响应用的计算复杂度；除了上述两种情况外，数据分布异常还包含数据混合分布等更为复杂的分布方式。因此，本文将在接下来的小节中介绍数据分布方式。

### 数据分布方式

针对数据分布这一数据特征，本文将就几种典型的数据分布形式进行了如下概述和应用场景分析[40]。

1. 均匀分布

如果随机变量X的概率密度为

则称X服从区间[a,b]上的均匀分布，记为。即服从U(a,b)上的均匀分布的随机变量X在区间[a,b]中取值的概率只与其区间长度有关，而与在区间[a,b]中所处的位置无关。

均匀分布可以用于以下数据生成场景中：与应用相关性比较小的属性列；常规的数据生成中。

1. 高斯分布

若随机变量X的概率密度为

其中μ和σ均为常数，且，则称X服从参数为μ，σ2的正态分布，简记为。

根据中心极限定理，*n*个随机过程的均值服从高斯分布，因此高斯分布可以满足大自然中多数自然现象，如人的身高、体重，医学检验指标中红细胞、血小板的数目等。同时，多个随机变量的和也可以用高斯分布来近似。所以，绝大多数的数据处理场景都可以使用高斯分布。

1. 伽马分布

若随机变量X的概率密度为

其中，则称X服从参数为的伽马分布。

伽马分布经常作为共轭分布，出现在很多机器学习算法中。同时，伽马分布可以用来研究数据的倾斜分布，通常用于排队分析中。

1. 泊松分布

若随机变量X的概率密度为

其中，则称X服从参数为λ的泊松分布，简记为。

泊松分布[41]是离散型的概率分布，适合于描述单位时间内随机事件发生的次数的概率分布场景中：1）某一服务设施在一定时间内受到的服务请求的次数；2）机器出现的故障数；3）自然灾害发生的次数；4）DNA序列的变异数；5）放射性原子核的衰变数等。

1. 指数分布

若随机变量X的概率密度为

其中为参数，则称X服从参数为λ的指数分布，记为。

指数分布通常可以用于以下应用场景：1）可以表示独立随机事件发生的时间间隔（如，顾客进入店铺购买商品的时间间隔等）；2）可以用于描述电子元器件的使用寿命；3）可以用于“无记忆性”的现象（如，元件对于它已经被使用过*s*小时没有记忆）。

1. 几何分布

若随机变量X的概率密度为

则称X服从参数为p的几何分布，其中。

几何分布可以应用于“无记忆性”的场景中。

1. Zipf分布

Zipf分布[59]是美国学者G.K.齐普夫提出的一个统计型的经验规律，即只有少数英文单词经常被使用，大部分的单词很少被使用，类似于人们常说的80/20原则。定量来说，内容访问近似符合Zipf定律。Zipf定律（Zipf's Law）指的是一个词在一个有相当长度的语篇中的等级序号（该词在按出现次数排列的词表中的位置，称之为rank，简称r）与该词的出现次数（称之为 frequency，简称f）的乘积几乎是一个常数（constant，简称C）。用公式表示，就是。

Zipf分布可用于模拟倾斜数据的生成，且通常用于以下数据场景中：1）网页排名；2）网站访问流量；3）URL访问量等符合Zipf定律的场景。

### 异常数据生成

上述内容介绍了应用特征分析以及数据分布方式，接下来，本节将就具体的数据生成方式进行详细的说明。首先，总结了SQL、Graph以及Machine Learning等不同应用类别的数据生成（包括常规数据和异常数据）；然后，对这几种应用类别的异常数据生成方法进行详细的介绍。

* **数据生成**

表4-2给出了不同应用类型的数据生成方法，包括常规数据生成以及异常数据生成。**常规数据**指的是通常使用的、按照一般规则生成的数据，同时这些数据满足数据格式的需求。**异常数据**指的是满足数据量大、数据倾斜、数据稀疏、数据维度高或数据分布异常等异常特征的数据，这些数据可能会对应用运行时的计算复杂度产生影响。

表 4-2 数据生成方法

|  |  |  |
| --- | --- | --- |
| **应用类型** | **常规数据** | **异常数据** |
| **SQL** | 使用数据库评测基准（如，TPC-DS，TPC-H等）提供数据生成。 | 生成满足异常分布（如，Zipf分布、泊松分布、泊松分布等混合）的随机数据；生成倾斜数据（如，单个key多次出现）；数据大小异常（一行特别长，value值过大）等。 |
| **Graph** | 使用真实数据，如Amazon、WikiTalk、Friendster等数据集[42]；或使用LDBC社交网络基准（SNB）数据生成器（Datagen）生成的合成数据集。 | 使用泊松分布生成顶点离散的图；使用Zipf分布生成顶点度异常分布的稀疏图。 |
| **Machine Learning** | 使用真实数据集，如SUSY、KDD2010以及ALS中使用的MovieLens MovieLens等。 | 随机合成不同维度、稀疏度、异常分布（如，高斯分布、伽马分布、泊松分布、指数分布、Zipf分布及其混合）等的数据。 |

常规数据生成使用的是真实数据集（如Machine Learning中使用的KDD2010、SUSY等），或现有的大数据基准测试（如SQL中常用TPC-H、TPC-DH等）提供数据生成。异常数据生成是根据应用的异常数据规则进行的，这些异常数据符合应用的操作特性，且能更大程度的发现系统及应用可能存在的可靠性问题。

接下来将针对不同的应用类型，给出具体的异常数据生成方式。

* **SQL异常数据生成**

SQL的基础查询语句（如Scan、Aggregate、Join等）主要用于处理key/value对，其计算复杂度主要与key的大小及分布相关。因此，可靠性测试基准在对SQL进行异常数据生成时，主要考虑数据量大、数据倾斜以及数据分布异常等异常数据特征。

通常情况下，进行测试使用的数据集都是常规数据，但是在某些特殊生产环境下，很可能会接收到非常规的数据，这其中最常见的就是倾斜数据。在SQL中，当表中某列数据的值分布不均匀，或某个key占到了总数的很大比例时，可以称其为倾斜数据。如果数据过于倾斜，在map/reduce时，会导致某一key的记录数明显高于其它key值，这个key所在的结点就会比其他结点处理更多的数据，导致结点任务分布不均。这种由于数据倾斜导致的结点任务分布不均的情况，会使倾斜数据所在结点任务加剧，而其他结点空闲，容易出现内存溢出或运行超时等可靠性问题。

本文对SQL给出了以下几种分布形式：

1）均匀分布：数值在数据段内分布均匀，对一些影响较小的列使用。

2）高斯分布：对于网页内容大小、包含的外链数目等采用高斯分布。

3）Zipf分布：可以用于网页访问量、URL访问量等符合Zipf定律的属性列。

在本文设计的两张表Rankings和UserVisits中（如表3-3所示），如果把外链数目作为pageRank确定原则，根据经验表明网页中大多数外链都指向极少数的网页，所以pageRank服从Zipf分布；同样，极少数网站承载了网络上大多数访问流量，URL的访问也遵循Zipf分布。pageURL和destURL作为两表合并的关联列，若destURL是倾斜的，那么在联接的shuffle操作时，会产生数据倾斜。本文用的浮点数表示倾斜程度，将倾斜程度为0的视为常规数据。不同倾斜程度的分布情况如图4-1所示。

下面给出了SQL异常数据的生成方法，如图4-2所示。首先，设计数据表；其次，对表中的各个列进行属性分析；然后，针对列的属性选择需要的数据分布形式；最后，对应用产生符合某种分布的异常数据。



图 4-1 单参数Zipf分布



图 4-2 SQL异常数据生成

* **Graph**

Graph中的应用大多需要迭代计算。目前使用广泛的迭代模型是Vertex-centric（顶点为中心）。使用该模型，图中的活跃顶点在每轮迭代中都会从邻居顶点接收消息，计算后更新自身的值，然后将更新后的消息传递给邻居顶点。这种以顶点为中心的迭代模型中，当出现数据倾斜时（即某一顶点的相邻顶点格外多），会对单个顶点的计算造成巨大的压力，容易产生内存溢出等可靠性问题。因此，图数据的生成需要考虑以下几点因素：1）数据集的大小；2）节点度分布；3）结构性质，如聚类系数或分类程度等。

本文针对Graph中的顶点度的异常分布，给出了以下几种分布形式：

1. 几何分布：可以混合Zipf分布使用，生成顶点度分布倾斜的图数据。
2. 泊松分布：可以生成顶点离散分布的图数据。
3. Zipf分布：可以模拟顶点度分布倾斜的情况，从而生成满足幂率分布的图数据。

在遵循实际数据特征（如数据集的大小，图的结构性质等）的前提下，根据上述提供的不同的数据分布方式特点，本文提供了对顶点度的多种分布，用于生成满足不同异常分布的图数据。图4-3为Graph的异常数据生成方法。



图 4-3 Graph异常数据生成

* **Machine Learning**

本文针对Machine Learning提供的异常数据生成主要有两种途径：（1）基于原始数据（即真实数据）扩展生成异常数据；（2）根据异常数据的特征，随机合成的数据。

基于原始数据的生成方式中，首先需要进行预处理，使用已有的真实的数据，将其转换成所需的格式（如，剔除其中格式不正确或是修正其中格式错误的数据）；其次提取需要的字段，即选取其中需要的字段组成新的数据集；最后需要进行格式归一化，即将各个维度的数据都归一化到同一区间。

对于随机合成的数据，根据3.3节中介绍的数据格式，随机生成满足不同维度、实例数、稀疏度以及异常分布的数据。其中，稀疏度分为整体的稀疏度和特征的稀疏度；异常分布包括高斯分布、伽马分布、泊松分布、指数分布、均匀分布、Zipf分布及混合分布等。每个样例的同一个属性符合某种分布。通过设定不同的数据特征信息，生成接近于真实数据的随机合成数据。图4-4是为Machine Learning生成异常数据的方式。



图 4-4 Machine Learning异常数据生成

* **异常数据生成流程**

对于可靠性测试框架提出的典型应用，其异常数据的生成将按照图4-5所示的流程进行。下面以Graph中的PageRank为例，其异常数据生成流程如下：

1. 分析应用特征。

PageRank需要由顶点向相邻顶点发送消息，当出现数据倾斜时（即某一顶点的相邻顶点格外多），对单个顶点的压力过大。

1. 选取异常规则。

根据应用特征，当某些顶点的压力过大，可能会出现内存溢出等问题。因此选择特征为数据规模大、数据稀疏以及顶点度分布异常。

1. 生成异常数据。

根据异常规则，生成数据量大（如，超过50G）、数据稀疏（如，稀疏度为0.1）、分布异常（如，满足Zipf和伽马等混合分布）的异常数据。



图 4-5 异常数据生成流程

## 基于贪心算法的参数组合空间削减方法

本节为了解决组合参数测试中遇到的参数组合空间爆炸的问题，提出了一种基于贪心算法的参数组合空间削减方法。本节首先对应用参数进行了分析，然后介绍了组合空间削减方法，最后提出了一种用于确定参数取值的探测性参数验证方法。

### 应用参数分析

已有的研究[46]发现，70%以上的程序错误是由系统参数中的某2个的相互作用导致的，而超过90%的程序错误则是由3个及以内的参数的相互影响造成的。上述发现可以说明，大多数的程序错误是由参数配置中的少数几个的相互作用造成的[44]。

因此，在解决组合空间过大的问题之前，本文首先对3.4.1节中给出的Graph以及Machine learning类型的应用参数进行可靠性的理论分析，来研究参数与资源占用（如，CPU、Memory等）是否相关，以及参数之间是否会相互作用而对资源占用产生影响。

* **Graph参数分析**

由于图算法通常采用迭代模型进行计算，因此图算法中的参数通常与最大迭代次数相关。下面针对迭代模型进行分析。

分布式图计算通常使用分布式批同步（BSP）模型，而其中比较流行的分布式编程框架有Pregel。如图4-6，Pregel采用迭代的计算模型：在每一次迭代过程中，每个顶点处理上一轮收到的消息，并向其他顶点发出消息，同时更新自身状态和拓扑结构（出、入边）等。如果某个顶点在一轮迭代中没有进行自身信息的更新，那么这个顶点便成为了不活跃的顶点，下一轮迭代中将不再进行消息的传递。

值得关注的是，该模型迭代终止的条件是工作集为空，或者达到了最大迭代次数。若迭代次数达到了最大值，然而此时工作集不为空，则无法得到正确的结果。例如，在使用ConnectedComponents应用进行联通分量计算时，如果没有迭代完全，则会得到错误的联通分量划分结果。因此参数maxIteration（最大迭代次数）在某些看重计算结果正确性（如，ConnectedComponents、TriangleCount等）的应用中，便是一个影响可靠性的危险参数。

由于图算法中的参数相对较少，通常只与maxIteration相关。而对于有些应用，如PageRank、SingleSourceShortestPaths等，还与其他参数（如，PageRank中的beta，SingleSourceShortestPaths中的srcVertexId）相关。然而这些参数只是起到了计算公式因子或者开始顶点选定的作用，与图算法执行中的时间或空间复杂度都不相关，同时与计算结果正确性也不相关。因此，在Graph需要使用的参数中，maxIteration（最大迭代次数）可能会影响应用的计算复杂度，同时可能会造成计算结果出错。然而，Graph中应用并不存在参数之间相互作用导致可靠性问题的情况，每个参数可以看做是相互独立的。



图 4-6 图算法中迭代计算模型

* **Machine Learning参数分析**

Machine Learning中的应用通常具有较多的配置参数，并且每个参数的取值区间较为分散。下面针对每个应用进行单独的参数分析，目的是发现其中可能会对应用运行的时间、空间复杂度或计算结果产生影响的参数或参数组合。

RandomForest中有三个与应用计算复杂度相关的参数，分别是numTrees（树的数目），maxDepth（最大树深）和maxBins（最大分箱树）。其中，maxBins指每一个特征最多有多少种分法。例如年龄特征，可以使用不同的年龄作为划分标准，即可以在10岁、20岁或30岁这样的地方进行划分。RandomForest的建树过程是每次取一个group的结点放到队列中，其中队列大小是有限制的，而结点大小是由决定。同时，group按层选取结点，每层的结点数是由maxDepth和numTrees决定。接下来，假设numTrees=10，featuresNum=15，maxBins=20，binSize=0.01M，同时假设队列大小为1000M。在上述假设条件下，RandomForest构建的树中，第一层有10个结点，需要占用内存的大小为；按照这种方式依次计算，到第8层的时候，森林中共有个结点，需要空间为=3840M，大于容量为1000M的队列。因此无法一次性的处理这些结点，需要多轮处理。如果单纯考虑队列内存的使用，由于队列内存是一个固定的值，所以这几个参数的乘积会对其产生一定的影响。另外，每个参数的值的增大都会使得运行时间增加，总的内存占用增加。因此，在RandomForest应用中，参数numTrees、maxDepth和maxBins的共同作用，会对应用运行时的空间以及时间复杂度的产生影响。

LogisticsRegression应用中的参数numIteration（最大迭代次数）和参数convergencoTol（误差容忍度）的相互作用，会决定应用何时运行结束。在每次迭代中，应用都会计算一下误差，误差随着迭代是递减的。应用运行结束有以下两种情况：（1）当达到设定的最大迭代次数，但还没有达到收敛精度（误差容忍度）时，运行结束；（2）在设定的最大迭代次数内，当达到了收敛精度时，运行结束。因此，这两个参数共同决定了应用运行的结束，但存在以下问题：（1）如果迭代次数过小而没有达到给定的误差容忍度，虽然没有产生错误，但如果与设定的误差容忍度相差过大，同样也应该看做是错误的；（2）如果迭代次数过大，虽然能够达到给定的误差容忍度，但是如果运行时间超过了预期，也应该看做是错误的。

### 参数组合空间削减

通过对应用参数进行分析，发现大多数的参数是相互独立的，有少数的参数或者参数组合会影响到应用运行的计算复杂度。因此，本文针对应用运行时的配置参数，做了以下两个假设：

**假设一**：*n*个参数互相独立；

**假设二**：第*i*个参数的*mi*个可选值与性能/资源利用率正相关或负相关。

在满足这两个假设的基础上，我们可以得出以下结论：**参数在取得临界值时应用性能最差或资源消耗最高（可能会触发运行时错误）**。

在满足上述假设基础上，我们提出了一个基于贪心算法的参数组合空间削减方法。该方法的具体流程如图4-7。

1. 给定*n*个参数中每个参数的具体取值范围；
2. 选择每个参数的某一临界值组合，进行测试，并记录资源占用情况；
3. 改变一个参数的取值为另一临界值，进行测试，并记录资源占用情况；
4. 比较最近两次参数组合测试中的资源使用情况，保留性能较差的那个参数作为接下来测试中的固定配置。
5. 返回第2步重复执行，直到出现异常或组合测试结束。如果出现了异常，则找到了出现异常时的参数配置情况；如果没有出现异常，则得到资源使用最差或性能最差的配置。



图 4-7空间削减方法执行流程

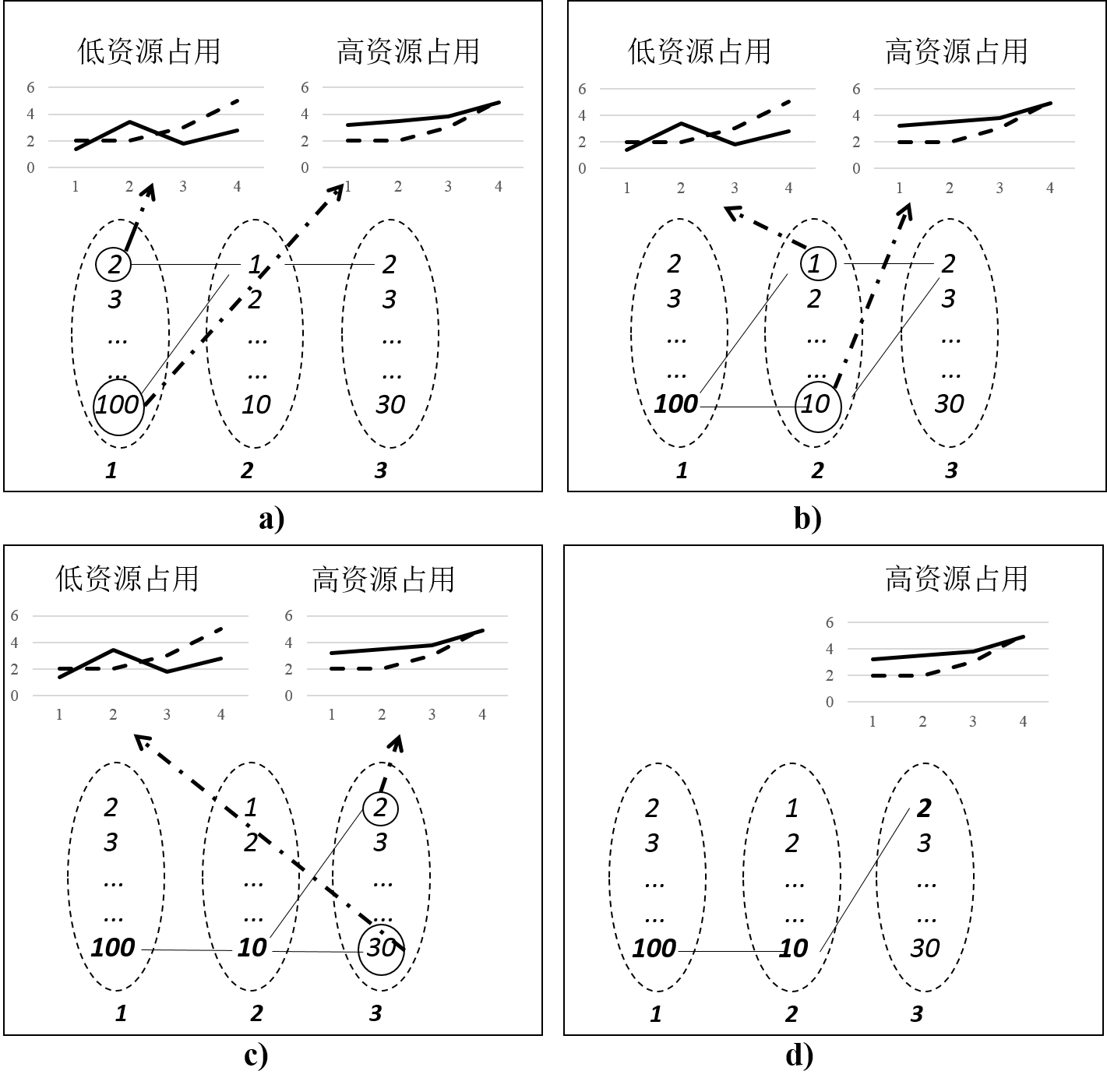


图 4-8 参数组合空间削减

图4-8给出了一个具有三个配置参数的例子。（1）首先，选择三个参数的最小边界取值组合，即*2-1-2*，进行测试并记录资源占用*R1*；（2）然后，将第一个参数的取值改为最大边界值，即*100-1-2*，进行测试并记录资源占用*R2*；（3）比较应用在参数组合*2-1-2*和组合*100-1-2*下进行组合测试的资源使用情况，即比较*R1*和*R2*。如果*R1<R2*，则固定第一个参数取值为100，即保留组合为*100-1-2*。（4）接下来改变第二个参数的临界值为10，即参数组合为*100-10-2*，进行测试，并记录资源占用*R3*。（5）比较参数组合*100-1-2*和参数组合*100-10-2*的资源占用，即比较*R2*和*R3*。如果*R2<R3*，则固定第二个参数取值为10，即保留组合为*100-10-2*。（6）接下来改变第三个参数的临界值为30，即参数组合为*100-10-30*，进行测试，并记录资源占用*R4*。（7）比较参数组合100-10-2和参数组合100-10-30的资源占用，即比较*R3*和*R4*。如果*R4<R3*，则固定第三个参数取值为30，即保留组合为*100-10-2*。此时，参数组合测试结束，并得到了使应用有最差资源使用情况的参数组合。

本文提出的参数组合空间削减方法，从空矩阵开始，通过逐列的方式扩展矩阵，直到覆盖数组中的所有测试都被覆盖。该方法是一种贪心算法，每次测试将资源占用差的参数配置作为固定参数，并进行下一轮的测试。这样每次得到的是局部最差资源占用，而非全局的。另外，该方法假设了每个参数的在其取值范围内是与资源占用成一定的相关性的，但如果不符合相关性，该如何解决呢？本文将在下一小节提出一种探测性参数验证方法，来解决参数取值与资源占用不符合相关性的问题。

### 探测性参数验证

上述参数组合空间削减方法是在两个假设的前提下提出的，但如果第*i*个参数的*mi*个取值并不满足正相关或负相关，则需要通过探测性方式找到该参数使得应用有相对较差的资源占用的取值。

首先，考虑TCP进行拥塞控制时采取的方案。在使用TCP进行网络传输过程时，为了使网络中的路由器或链路不致过载，需要进行拥塞控制。而慢启动就是TCP使用的一种阻塞控制机制，也叫指数增长期。使用慢启动进行数据传输时，拥塞窗口从1开始，然后呈指数增长，直至达到初始阈值。通过这种方式可以快速的探测到阈值。

借鉴网络传输中的这种指数增长探测阈值的策略，本文提出了解决参数组合空间削减方法中参数取值与资源占用不符合相关性的问题的探测性参数验证方法。针对某一参数的探测性参数验证的方法如下：

1. 设定参数的初始值，以及最大测试次数。
2. 参数按照函数进行取值，并进行测试，同时记录资源占用情况。
3. 当测试次数达到设定的最大测试次数，或者测试过程中出现了错误，则该参数的探测性参数验证测试结束。
4. 测试结束后，比较次测试中资源占用情况，从而确定该参数的最差取值。

在上述方法中，提供了参数的初始值，即参数取值的下限。探测过程中使用的指数函数，表示第*n*次测试时，该参数的取值为。其中，和分别是控制因子，用于调控参数值的变化情况，即控制指数的增长速度。对于那些取值范围较小的可以设置在区间内，对于那些取值范围较大的可以设置为大于1的值。在确定了和的取值后，通过最大测试次数可以控制参数取值的上限。另外，控制因子和，以及最大测试次数可由用户来决定取值。

由于在参数取值过程中，能够发现错误的上限值通常并不能通过经验来确定。同时，考虑到参数取值范围一般能够在指数增长的取值内涵盖，因此，探测性参数验证方法中，将探测次数，即最大测试次数，作为上限确定的因素。

使用探测性参数验证方法进行参数组合测试，在有*n*个参数，且探测次数为*t*的情况下，其时间复杂度在最差情况下为。

## 小结

本章提出了一种基于应用特征的异常数据生成方法，以及一种基于贪心算法的参数组合空间削减方法。下面将对这两种方法的特点进行总结。

* 基于应用特征分析的异常数据生成方法特点

可靠性测试框架通过使用该方法，可以针对给定的新应用，通过分析计算特性，确定与之对应的数据异常规则，从而生成符合应用场景的极端异常数据。同时，该方法还具有以下特点：（1）接近应用的：通过分析应用本身的计算特性，发现可能存在的问题，进而有针对性地产生相应的异常数据；（2）可实证分析的：从现存的问题出发，分析问题产生的具体原因，进而产生相应的异常数据；（3）问题放大化：即根据已有的问题，尽可能放大该问题的影响，进而产生更多的或是更严重的问题；（4）符合异常数据特征的。

* 基于贪心算法的参数组合空间削减方法特点

该方法是一种贪心算法，即通过直接搜索覆盖数组的方式来进行参数组合测试。贪心算法本身具有以下特点：（1）算法简单、有效，且处理速度快；（2）是一种近似算法，不能保证结果的最优性。本文提出的用于可靠性测试的参数组合空间削减算法，因其属于贪心算法，因此是一种局部搜索方法。该算法虽然在处理时间上有所减少，但并不能保证得到最优解。然而，考虑到可靠性测试的以下特点：（1）对最优解的需求不大；（2）对处理时间要求较高。因此，这种基于贪心算法的参数组合空间削减方法，提高了测试效率，同时不影响发现错误的存在。

# 可靠性测试框架系统实现

本章就大数据应用可靠性测试框架的系统实现进行了介绍。首先给出了系统的架构，然后对系统中的各个模块的实现进行了详细的介绍，最后将该系统应用到了具有代表性的大数据系统Spark上进行可靠性验证，并发现了其中存在的若干可靠性问题。

## 系统架构

本节主要介绍了可靠性测试框架的系统架构。本文实现的大数据应用的可靠性测试框架是一个为大数据系统及应用提供可靠性测试，从而发现潜在问题的系统。该系统提供Web访问界面，支持满足异常规则的异常数据生成，支持参数组合空间削减的组合测试，同时提供测试报告生成。具体的系统架构如图5-1。

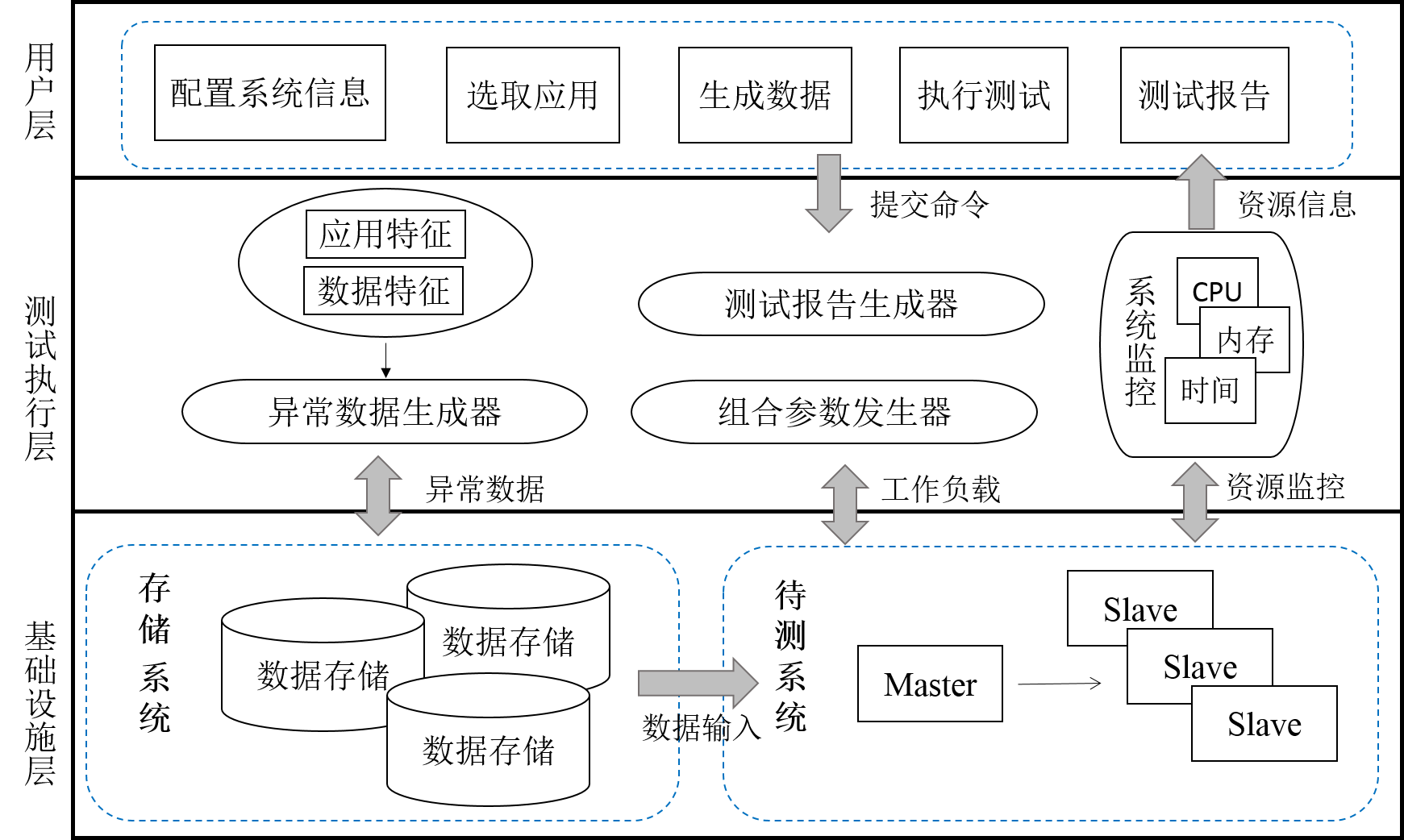


图 5-1 系统架构图

面向大数据应用的可靠性测试框架系统分为三层：用户层、测试执行层和基础设施层。

（1）用户层

用户层为用户提供Web视图界面。通过用户视图，用户可以1）配置待测系统以及存储系统等配置信息，2）选取基准应用，3）配置异常数据规则来生成自定义的异常数据，4）执行自定义参数的测试或执行自动化的参数组合空间削减的组合测试，5）最后，在测试执行结束后，用户可以查看生成的测试报告。

（2）测试执行层

测试执行层用于执行可靠性测试。测试执行层从用户层接收用户命令，从而执行相应的操作指令，主要包括：异常数据生成器、组合参数发生器以及测试报告生成器。其中，异常数据生成器可以针对不同应用类型，提供不同的数据生成脚本；组合参数发生器提供自动化的参数组合测试，通过系统监控来监测系统的资源占用情况，并使用空间削减策略进行组合测试降维；测试报告生成器通过系统监控获取测试中的运行指标，并通过汇总数据集、应用参数信息和错误信息等，生成测试报告。

（3）基础设施层

基础设施层提供数据存储以及待测系统执行平台。可靠性测试框架将异常数据生成器生成的数据集存储在数据存储系统（如HDFS等）中，并使用测试执行层中的组合参数发生器对待测系统提供可靠性测试。

接下来，本文将在该系统架构的基础上，详细的介绍系统实现。

## 系统实现

本节将介绍可靠性测试框架的系统实现。该系统是一个多层架构，包括Web模块和测试模块两部分。其中Web模块用于提供用户层界面支持，测试模块提供测试执行层的支持，用于部署到待测系统中进行可靠性测试。Web模块与测试模块之间通过远程访问建立调用关系，图5-2展示了系统模块之间的关系。下面将分别针对Web模块和测试模块的具体实现进行介绍。

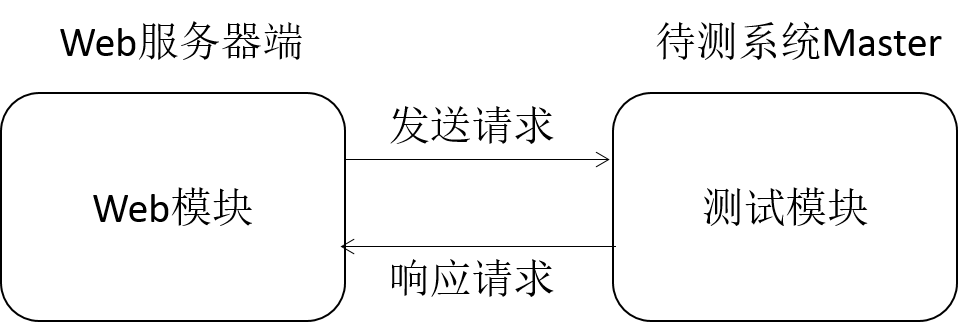


图 5-2 系统模块关系图

### Web模块

Web模块用于提供用户层的界面支持，并遵循了MVC框架，分为模型（Model）、视图（View）以及控制器（Controller）三部分，如图5-3。简单来讲，其中的Model层用于数据的存取；View层主要用于信息展示；Controller层用于对数据进行逻辑处理。在这样的三层模型中，每一部分都可以专注于自身的职责，避免了相互之间复杂的逻辑关系。



图 5-3 MVC框架图

* **包图**

图5-4给出了系统中Web模块的包图，体现了包之间的依赖关系，其中上层包依赖于下层包。

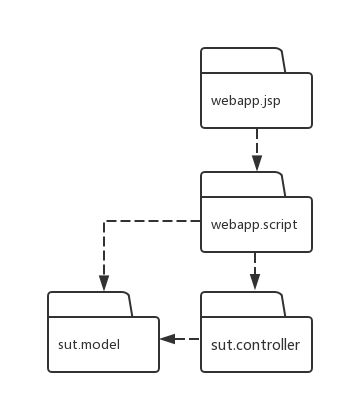


图 5-4 Web模块包图

在Web模块的包图中，（1）webapp.jsp和webapp.script为View层，负责展示用户层，即为用户进行可靠性测试提供界面支持。包括配置系统信息、选取应用、生成数据、执行测试以及最后展示测试报告。用户通过填写信息，并提交请求，完成可靠性测试流程。同时负责从Controller中接收传递回来的监控信息，形成测试报告展示给用户。（2）sut.model为Model层，负责提供用户状态信息、测试信息的存储。（3）sut.controller为Controller层，用于响应View层发送的用户请求，来修改Model层中的数据，包括向待测系统发送生成数据以及执行测试的命令，同时可以从待测系统获取监控信息，并传递给View进行展示。

下面对Web模块中的各个包进行简要的介绍。

1. webapp.jsp包提供了前端界面需要的JSP文件，为用户展示操作步骤中需要的配置信息、应用选择等信息，并展示测试结束后的测试报告。
2. webapp.script包提供了用于连接前端展示的View层和进行逻辑处理的Controller层的JavaScript文件，可以实现交互式的通信。
3. sut.controller包提供了web.script访问后台数据或发送用户请求所需的接口。
4. sut.model包提供了web.script与sut.controller之间通信的数据定义规范。
5. service包负责接收前端发来的请求，并提供系统的后台服务（如，数据生成、组合测试、生成报告等）。

* **架构图**

Web模块主要提供用户层的界面需求，支持可靠性测试运行的整个流程。图5-5展示了Web模块的具体架构实现，其中上层为下层提供服务。下面将具体介绍每个文件的具体功能实现。



图 5-5 Web模块架构图

1. sparkConfig.jsp提供了整个可靠性测试的执行流程，提供信息配置、数据生成以及组合测试的功能选项。
2. sparkRunning.jsp提供了自动执行组合测试中的任务完成情况。
3. sparkReport.jsp提供了测试报告。
4. sparkMaster.js用于存储用户提供的待测系统信息。
5. sparkGen.js用于收集用户配置的数据生成参数，并将数据生成的命令提交到Controller端。
6. sparkRun.js用于收集运行测试时用户提供的参数信息，并将测试执行命令提交到Controller端。
7. sparkReport.js用于从Controller收集信息，生成测试报告。
8. SparkController.java用于接收script中传来的用户请求，并作出响应，向待测系统发送用户请求命令。同时将从待测系统端收集到的信息，响应给script。

在上述Web模块框架图的基础上，接下来，本文将根据测试流程进行介绍。

首先，用户需要输入用户名和密码登录到系统。登录成功后，用户需要配置待测系统的集群信息（如，访问端口、IP、用户名、密码等）和 HDFS配置信息（如HDFS路径、端口等），如图5-6所示。这些由webapp.jsp中的JSP提供支持，并通过webapp.script中的JavaScript将前端请求转发到sut.controller中进行命令执行。

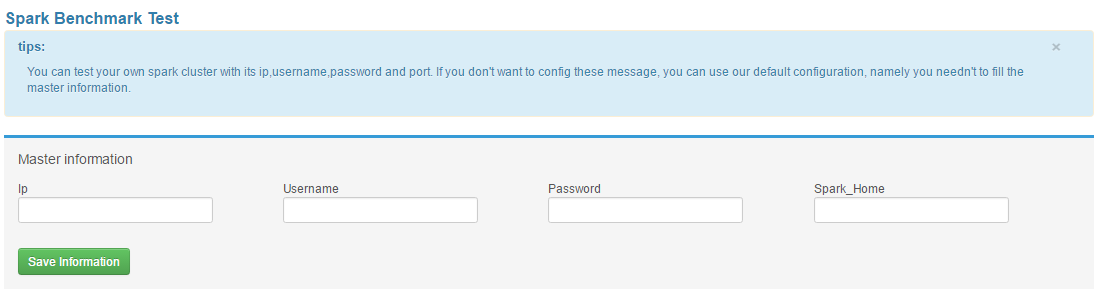


图 5-6 配置待测系统信息

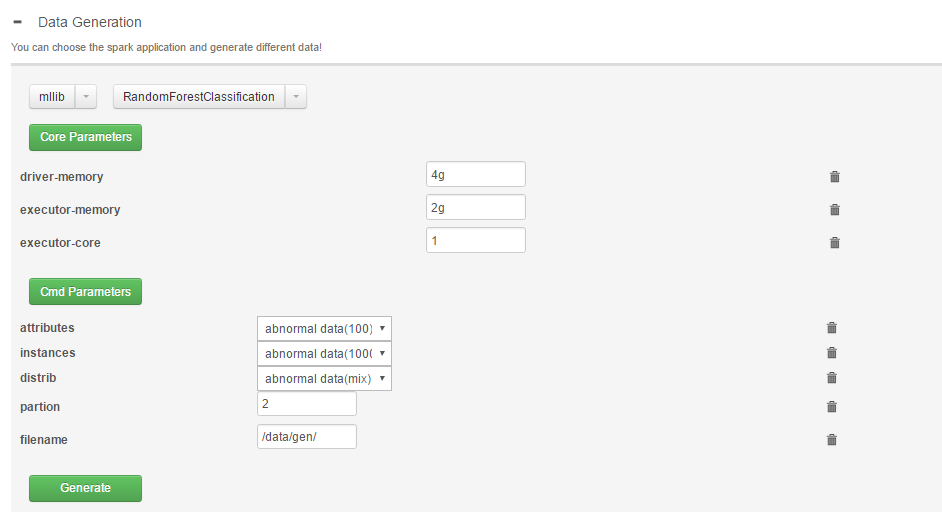


图 5-7 数据生成

然后，用户进行数据生成，如图5-7所示。用户首先选择需要生成数据的应用类型；然后给定集群运行的环境配置（即，内存和CPU的分配）；另外还需选择异常数据特征值，如RandomForest中选择特征数、实例数以及分布类型等；最后，通过点击“Generate”按钮，提交数据生成命令。该命令会通过webapp.script中的sparkGen.js传递给SparkController.java，从而向待测集群发送数据生成命令。

接下来，进入测试执行阶段，如图5-8所示。用户首先需要配置待测应用的系统参数和应用参数等信息，并给定集群运行测试所需的环境配置。如果采用自动测试的方式，在点击“Run”按钮后，平台会执参数组合空间削减的组合测试。如果采用人工测试，需要用户提供参数值来执行测试。测试执行结束，测试报告会通过JSP页面显示给用户，如图5-9所示。

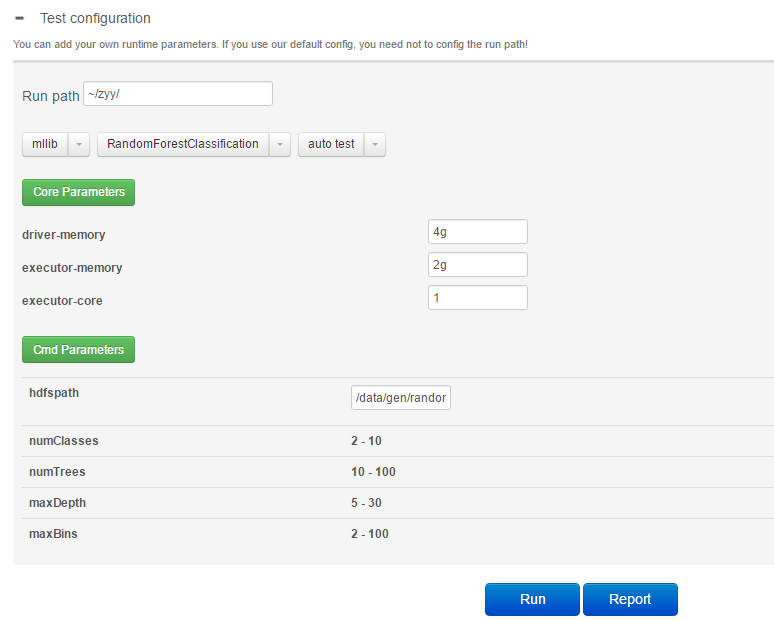


图 5-8 组合测试

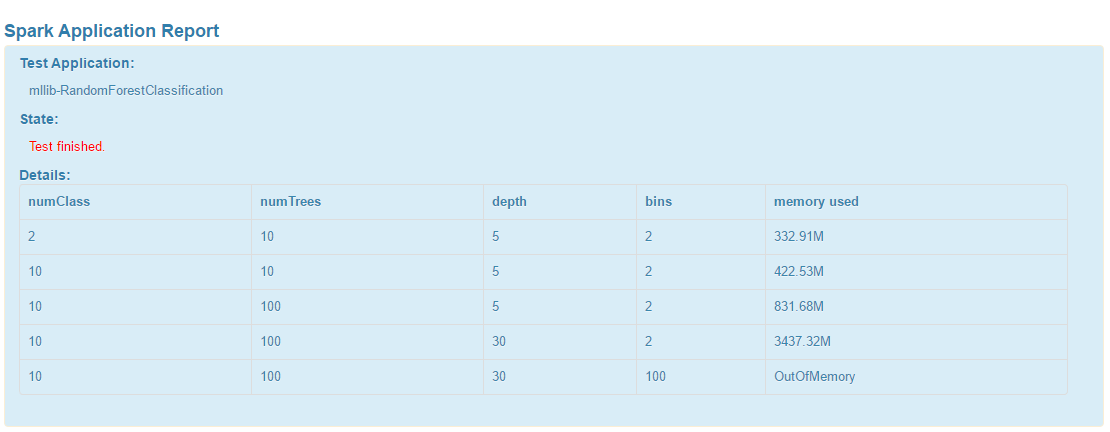


图 5-9 测试报告

图5-10展示了用户通过Web访问平台的执行流程的活动图。



图 5-10 活动图

### 测试模块

测试模块用于为待测系统提供满足不同应用类型的数据生成脚本以及组合测试脚本。本小节首先介绍了测试模块的包图，然后针对其中的核心组成部分的实现进行详细说明。

* **包图**

图5-11给出了系统中测试模块的包图，图中展示了包与包之间的包含关系，其中下层包依赖于上层包。

bench包下可提供对不同大数据系统的测试支持。目前仅提供了对Spark系统的可靠性测试支持，后续将提供Flink等系统的测试支持。conf包是bench包下的配置包，用于提供各类配置文件。由于SQL和Graph的数据生成对不同的基准应用具有通用性，因此将其数据生成脚本放到了sparkbench下的generate包中。而对于Machine Learning中的数据生成针对不同的典型应用提供了不同的脚本，将其放到了Machine Learning包下的dataGenerated包中。同时， SQL、Graph以及Machine Learning包下都包含script包，用于提供工作负载和测试脚本。util包是工具包，提供路径转换、文件读取、公式转换等常用的工具类。

测试模块中最重要的就是数据生成与参数组合测试。数据生成由数据生成器提供，参数组合测试由组合参数发生器提供，另外还需要测试报告生成器提供最终的测试报告。本文接下来将对这几个重要组成部分进行具体介绍。

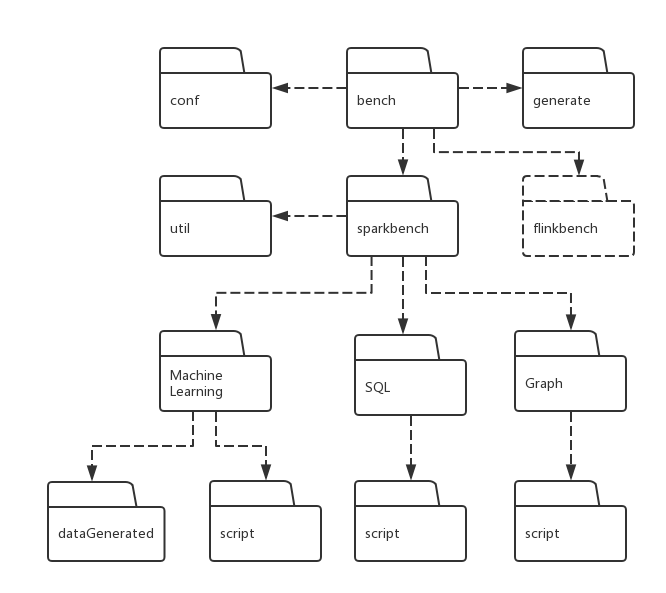


图 5-11 基准模块包图

#### 异常数据生成器

可靠性测试框架中的数据生成中使用到了多种概率分布；同时针对不同的应用需求，生成的数据模式也是不同的。接下来，首先介绍概率分布中的部分算法实现，然后介绍不同应用的数据生成实现。

* **概率分布算法实现**

针对数据生成中使用到的几种分布形式，包括均匀分布、高斯分布、伽马分布、泊松分布、指数分布、几何分布以及Zipf分布。本文就其中的高斯分布、泊松分布、指数分布以及Zipf分布，提供了如下的算法实现。

（1）高斯分布

高斯分布即正态分布，在Java中给出了满足区间的正态分布函数，即java.util.Random类中提供的nextGaussian()方法。下述算法使用该方法提供了满足任意范围内的高斯分布随机数。首先，初始化均值和方差；然后，使用nextGaussian()方法，并通过均值和方差进行数据处理，最后生成满足高斯分布的随机数据。

|  |
| --- |
| 算法1：Gaussian Algorithm: |
| 1. init: 2. . 3. generate: 5. return x. |

（2）泊松分布

下面给出了一个用来生成随机泊松分布的简单算法[47]。

|  |
| --- |
| 算法2：Poisson Algorithm: |
| 1. init: 2. Let . 3. . 4. do: 6. . 7. . 8. while . 9. return x. |

（3）指数分布

|  |
| --- |
| 算法3：Index Algorithm: |
| 1. init: 3. generate: 4. . 5. . 6. return x. |

根据指数的概率分布函数，首先将y当做满足区间的均匀分布随机变量。则其逆函数为

，

如果令，则显然z也满足区间的均匀分布。因此，生成指数分布的随机变量可以通过以下函数来计算：

。

（4）Zipf分布

本文采用蒙特卡罗方法生成zipf分布的随机数据。蒙特卡罗算法是指使用随机数（或更常见的伪随机数）来解决很多计算问题的方法，采样越多，越近似最优解。再根据Zipf定律中，一个词在文章中的出现次数排列位置r与该词的出现次数f的乘积，满足，其中C是一个常数，本文设C值为1。则生成满足Zipf分布的随机数字的算法如下：

|  |
| --- |
| 算法4：Zipf Algorithm: |
| 1. init(R,F): 2. Let. 3. computeMap(size,skew): 4. . 5. for to size: 6. . 7. done 8. for i1 to size: 10. . 11. . 12. done 13. . 14. next(): 15. . 16. . |

根据上述提供的随机数字的算法实现，可以在随机数据生成中使用统一的接口提供不同场景下的数据生成。接下来将对不同的应用的数据的具体生成方式进行详细的介绍。

* **数据生成实现**

不同的应用类型的数据生成通过使用上述概率分布算法提供数据生成脚本。

（1）SQL

SQL中主要是使用Zipf分布算法来实现倾斜数据的生成，使用高斯分布算法来实现满足正态分布的数据的生成，以及使用均匀分布算法来实现常规数据的生成。

本文使用的数据表Rankings和UserVisits表如图5-12所示。其中Rankings中的pageURL是Rankings表的主键，destURL与sourceIP是UserVisits的主键，由于关联关系，destURL相当于Rankings的外键，可以进行关联（Join）操作。



图 5-12 Rankings与UserVisits表的关联关系

对于属性URL（包括Rankings表中的pageURL以及UserVisits表中的destURL），由于极少数网站承载了网络上大多数访问流量，因此URL的访问遵循Zipf分布。如果把外链数目作为pageRank确定原则，根据经验表明网页中大多数外链都指向极少数的网页，所以pageRank服从Zipf分布。因此，对于URL与pageRank属性，本文使用Zipf分布算法来生成相应的数据。 对于每个URL包含的外链的个数以及网页内容的大小，本文使用高斯分布算法生成；对于其他的属性，则使用均匀分布算法来生成。

本文提供的SQL数据生成脚本如图5-13所示。其中SQLGen是生成SQL数据的主类，RandomUtil是辅助类，用于提供常用随机数据生成方法。下面主要对SQLGen类中的几个重要方法进行简要说明。

1. load\_zipf()方法用来加载随机生成的Zipf字典。
2. genURL()方法通过使用Zipf分布算法生成URL。
3. genPageContent()方法通过使用高斯分布算法来生成网页内容。
4. genIP()方法通过均匀分布算法来生成IP地址。
5. genDestinationURL()方法通过使用Zipf分布算法生成URL。
6. genRankingsFile(outputfile)方法用于生成Rankings表文件。
7. genUserVisitsFile(outputfile)方法用于生成UserVisits表文件。



图 5-13 SQL数据生成脚本类图

（2）Graph

图数据的生成主要考虑顶点度的分布。由于在随机网络中，度的分布是泊松分布等其他分布形式的情况比较罕见，大多是满足幂率分布的无标度网络[48]。而Zipf分布就是典型的幂率分布，因此，图数据生成主要考虑Zipf分布以及其他分布与Zipf分布的拟合[49]。

图数据包括Vertex（顶点）数据以及Edge（边）数据。其中Vertex定义为二元组，即；Edge定义为三元组，即。Vertex中的vertexID和value都采用范围内顺序生成的方式。由于vertexID仅是标识顶点的编号，且value为顶点初值，因此选择了最简单的数据生成方式。Edge中的srcID和destID需要在Vertex的vertexID范围内，且决定了图中各个顶点度的数目，因此对Egde数据的生成考虑使用Zipf分布。图5-14展示了图数据生成的类图，其中GraphDriver用于启动数据生成，GraphGen用于生成满足一定分布的顶点以及边数据，GraphWriter用于将生成的数据写入到对应的文件中。



图 5-14 Graph数据生成类图

（3）Machine Learning

Machine Learning中的数据生成与以下参数相关，如表5-1所示。用户可以自定义配置数据生成的参数信息，从而生成种类繁多的数据集。其中distrib用于控制数据分布方式。目前提供了均匀分布、高斯分布、指数分布、Zipf分布等概率分布形式。

表 5-1 Machine Learning数据生成参数

|  |  |
| --- | --- |
| 参数 | 意义 |
| attribute | 特征量 |
| instance | 数据实例数 |
| distrib | 分布方式 |

图5-15是Machine Learning中数据生成的类图。

* MLGen是数据生成脚本的主类，需要接受输入的参数attribute、instance以及distrib；同时load\_zipf()提供满足Zipf分布的算法实现；run()方法是MLGen类运行的主方法。
* RandomUtil类是工具类，提供了满足不同分布的随机数据生成的方法。其中randomPossion()提供了满足泊松分布的随机数据，randomIndex()提供了满足指数分布的随机数据，randomBase()提供了满足高斯分布的随机数据。
* MLWriter类通过调用write()方法来生成数据到文件中。



图 5-15 Machine Learning数据生成类图

#### 组合参数发生器

组合参数发生模块负责执行参数组合测试，并通过资源监测器收集资源占用信息，并在测试结束后通过报告生成器生成测试报告，传递到前端展示给用户。其执行流程如图5-16所示。



图 5-16参数组合测试

用户通过前端配置待测的应用信息，并点击“测试”按钮，测试命令传递到控制端，由组合参数发生器来执行测试任务。组合参数发生器从前端获取到需要测试的应用类型，首先通过*init()*初始化参数值；然后运行*run()*方法执行测试，每测试完一组实验，都会变更参数值，直到所有的参数组合都被覆盖。在测试过程中，会调用*collect()*函数从资源监测器中获取监控的CPU、Memory等信息。测试执行结束后，由报告生成器收集测试结果以及参数配置信息，形成测试报告，并传递到前端，展示给用户。

组合参数发生器中的*run()*方法的具体实现如下：

|  |
| --- |
| 1. param[] //存放第*i*个参数的初始值 2. for to *n* do 3. for to *t* do 4. run(param); 5. . |

#### 测试报告生成器

报告生成器通过收集测试的应用参数信息以及最后测试得到的最差资源占用，将信息以Json格式通过Ajax传递给前端，前端通过JavaScript解析Json数据，并将信息通过JSP展示到界面。

* 资源监测器

由于Spark监控支持GangliaSink，因此资源监测使用开源工具Ganglia。Ganglia是UC Berkeley发起的一个开源项目，主要用来监控系统资源，如CPU、Memory、硬盘利用率等。如图5-17，本平台将Ganglia部署到各个Spark节点上，集群内的节点通过运行gmond收集发布节点状态信息，然后通过gmetad周期性地轮询gmond收集到的信息，并存入RDD数据库。Monitor从Ganglia中获取收集到的信息，并进行信息过滤，得到最大的CPU使用和Mem使用，用于测试用例的测试结果对比。另外，通过Json解析Spark提供的Metrics接口中的消息，可以获得Task以及Stage的具体信息。



图 5-17 系统监测器

## 应用验证

本节针对大数据系统应用的可靠性测试基准平台进行了应用验证。首先使用Spark系统做为待测系统，使用平台提供的可视化界面进行可靠性基准测试。其次对测试过程中发现可靠性问题的测试实例进行分析。

### 实验环境



图 5-18 实验环境图

如图5-18所示，可靠性测试基准平台的Web项目部署在Web服务器上；可靠性测试平台的测试项目部署在Master以及Slave节点上。同时，将Spark集群部署在了10台机器上，一台是Master，另外九台是Slave节点。节点的配置信息如表5-2所示。同时，在测试中，每个应用程序测试5次，并使用平均值作为最终测试结果。

表 5-2 节点配置信息

|  |  |
| --- | --- |
| 配置 | 参数 |
| 处理器 | 8 Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz |
| 内存 | 16G RAM |
| 硬盘 | 2 \* 1TB SATA |
| 操作系统 | Ubuntu 11.04 |
| Spark version | Spark 2.0 |

### 参数配置

在上述实验环境下对Spark系统进行测试，其系统配置如表5-3所示，这些参数主要决定了系统资源的分配情况。

表 5-3 Spark系统配置列表

|  |  |
| --- | --- |
| 参数 | 含义 |
| Executor core | 分配给每个任务容器（JVM）的逻辑核数 |
| Total executor cores | 分配给任务容器（JVM）的总的逻辑核数 |
| Executor memory | 分配给每个任务容器（JVM）的内存大小 |

本文在固定系统配置的条件下，通过组合大数据系统运行时的配置参数，来发现可能出现的异常。配置参数包括系统参数和应用参数，Spark系统中提供的系统参数如表5-4所示，应用参数如表4-3所示。

表 5-4 Spark系统参数列表

|  |  |
| --- | --- |
| 参数 | 含义 |
| Input split number | 数据并行度 |
| Partition number | 任务并行度 |

这些系统参数在大数据系统运行应用时可能会影响系统数据分配或任务分配。其中数据并行度指的是输入数据的分片数量；任务并行度是shuffle中reduce的分片个数。Spark中的reduce端分片数目确定方式如图5-19所示。

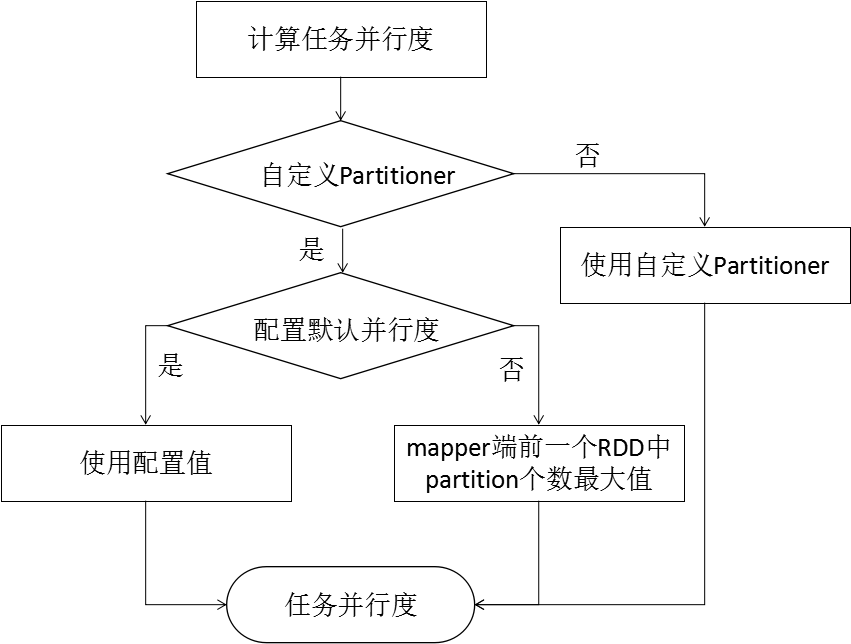


图 5-19 任务并行度确认方法

### 实例及分析

本小节采用上述测试环境，并设置Spark系统配置为4 executor（Executor core=2，Executor memory=8G），对Spark中的若干应用进行了可靠性测试。测试结果为：在6个应用中，发现了三种类型的错误，分别是内存溢出（OOM）、运行超时以及计算结果错误。表5-5给出了发现的可靠性问题的总结，下一节将针对出现的这几个可靠性问题给出实例分析。

表 5-5 Spark应用中发现的可靠性问题

|  |  |  |  |
| --- | --- | --- | --- |
| **应用名** | **输入数据** | **配置参数/操作描述** | **错误类型** |
| **Join** | 10GB,倾斜数据 | 小表Inner Join大表 | 内存溢出 |
| **Mix** | 10GB,倾斜数据 | 一表多次Join操作 | 计算结果出错 |
| **RandomForest** | 1 百万实例, 1000维度, 伽马、泊松混合分布 | numTrees=100, maxDepth=30,  dimensions=1000 | 内存溢出 |
| **LogisticsRegression** | 1.05GB 倾斜数据, 20216830维度 | split=134.13MB,  partitionNum=8 | 内存溢出 |
| split=268.25MB, partitionNum=4 | 运行超时 |
| **ALS** | 3GB数据量 | numIteration=20,  sampleFraction=0.001,  dataFeature=20 | StackOutOfMemory |
| **PageRank** | 10GB数据, 1百万顶点, 2千万边 | 收敛精度=0.001 | 内存溢出 |

#### SQL应用

针对SQL中的应用，仍使用表3-3中设计的表Rankings和UserVisit表进行测试。Join、Mix应用在10GB倾斜数据下，会出现内存溢出和计算结果出错的可靠性问题，下面将分别对这两个场景进行讨论。

* 小表内连接大表

测试提供了表连接顺序不同的两种Join语句，其中BigSmallJoin采用Uservisits(大表)内连接Rankings(小表)的顺序，SmallBigJoin采用Rankings(小表) 内连接Uservisits(大表)顺序。表5-6给出了两种Join操作的测试结果。

通过实验发现，在进行小表内连接大表的操作，并且使用倾斜度为0.8的倾斜数据时，出现了内存溢出的错误。图5-20给出了测试中的运行时间和占用内存的对比结果，其中横坐标为Task的编号，此处取的是前5个，纵坐标分别是执行时间以及占用内存大小。

表 5-6 Join测试结果

|  |  |  |
| --- | --- | --- |
| Join类型 | 数据类型 | 运行时间 |
| BigSmallJoin | 常规数据 | 51s |
| 倾斜度为0.8的倾斜数据 | 59s |
| SmallBigJoin | 常规数据 | 56s |
| 倾斜度为0.8的倾斜数据 | Failed |



图 5-20 常规数据和倾斜数据对比

通过图5-20对BigSmallJoin和SmallBigJoin的任务运行时状态分析，得出以下结论：

当给定的数据集倾斜时，某个Task上处理的记录数明显增多。其原因是key值相同的记录过多，shuffle时将这些记录推到了同一个任务结点上，导致这个任务结点运行时间远远大于其他任务结点。分析Spark内连接方法，发现当两个表进行内连接时，采用的是sortMergeJoin，将两表都进行排序。Spark将前一个表作为驱动表，后一个表作为缓冲表，即遍历驱动表中的每一条记录，在缓冲表中寻找相应匹配的记录，并将记录放入匹配表中。所以当把大表作为缓冲表时，找到的匹配记录会很多，如果此时大表存在严重数据倾斜，匹配表占用内存也会相应变多，在查询相关key时会发生内存溢出错误。

* 一个表参与多次Join操作

测试一个表的多次Join操作，具体的测试方案如下所示。

|  |
| --- |
| val df1 = Seq((1, 2), (3, 1)).toDF("col1", "col2")  val df2 = Seq((1, 2), (3, 1)).toDF("col1", "col3")  val df3 = df1.join(df2, df1("col1") === df2("col1")).select(df1("col1"), $"col3")  val result0 = df3.join(df1, df3("col3") === df1("col1"))  result0.show() |

（1）首先，执行df1与df2的内连接操作，得到一个临时表，经过select操作后，最终得到表df3，其执行过程如图5-21所示。



图 5-21 第一次Join操作

（2）然后，继续使用表df1执行Join操作，如图5-22所示。



图 5-22 第二次Join操作

在经过第二次Join操作后，理应得到的计算结果为（3,1,1,2），然而Spark执行后的结果为null。

|  |
| --- |
| Join Inner, (col3#16 = **col1#5**) **//col1#5 should be col1#49**  :- Project [**col1#5**, col3#16]  : +- Join Inner, (col1#5 = col1#15)  : :- Project [\_1#2 AS col1#5, \_2#3 AS col2#6]  : : +- LocalRelation [\_1#2, \_2#3]  : +- Project [\_1#12 AS col1#15, \_2#13 AS col3#16]  : +- LocalRelation [\_1#12, \_2#13]  +- Project [\_1#2 AS **col1#49**, \_2#3 AS col2#50]  +- LocalRelation [\_1#2, \_2#3] |

由于Spark SQL的lazy操作，这时虽然显示的用df3存储了中间结果，但最终只有在show()方法调用时，整个计划才会被执行，然而df1前后两次参与join，逻辑计划在列名绑定时出现了错误。如上所示。因此，当一个表同时参与多次Join操作但不重新命名时（子查询和外查询），Spark会将列名弄混导致结果错误。

#### Machine Learning应用

本小节对Spark MLlib中的RandomForest、LogisticsRegression以及ALS应用中发现的错误进行了测试验证。并且，在对RandomForest的测试中，验证了平台的参数空间削减方法的可用性。

* RandomForest在参数组合中出现内存溢出错误

对RandomForest应用的测试采用参数空间削减测试，其应用参数如表4-3所示。通过固定数据实例为106，维度为104，并使用高斯分布的规模为23.7G的数据，采用的系统配置中total-executor-cores大小为12，其他系统配置参数采用Spark默认配置。

表 5-7 RandomForest测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | A | B | C | D |
| numTrees | 2 | 100 | 100 | 100 |
| maxDepth | 5 | 5 | 100 | 5 |
| maxBins | 5 | 5 | 5 | 32 |
| partitionNum | 10 | 10 | 10 | 10 |
| 运行时间 | 6.4min | 41min | 内存溢出 | 内存溢出 |

通过测试得到结果如表5-7所示。A组参数配置为初始设置，B组改变参数numTrees（森林中树的个数）为100后，其运行时间明显增长。因此在C组参数配置中保留numTrees参数为100，改变maxDepth为100，出现内存溢出错误。如果继续进行如D组参数配置的测试，同样也会出现内存溢出错误。这样通过参数组合空间削减的方式，通过每次保留性能较差的参数取值，最终得到了出现异常的参数组合情况。

针对C、D两组参数组合，如果改变分布方式为均匀分布，则不会出现内存溢出的错误，说明数据的分布会影响应用的运行。同样，通过对RandomForest应用中的参数相关性的分析，发现参数numTrees、maxDepth以及maxBins的乘积会影响应用执行过程中的空间复杂度，如果乘积过大，会产生内存溢出的问题。这也是为什么C、D两组都出现内存溢出错误的原因。

* LogisticsRegression

在1.05GB的数据量，Split为134.13MB，partitionNum数目为8的环境下测试LogisticsRegression应用。运行过程中，LogisticsRegression会出现内存溢出或运行超时的错误。下面对出错原因进行简单分析。

（1）使用1.05GB的倾斜数据，20,216,830维度下，会出现内存溢出的错误，主要原因是数据量过大，维度过高，而在Driver memory分配过小或迭代次数过大的情况下，容易出现内存溢出的错误。

（2）如果设置的numIteration过大而convergencoTol又过低，则应用会一直运行以达到最大迭代次数或达到误差容忍度，这种情况下便容易出现运行超时错误。另外，4个task的序列化结果(2.2G)比driver.maxResultSize(2G)大的情况下，也会导致运行超时错误。

* ALS

使用3GB的数据量，在每个executor 12GB，每个executor 4个core的配置下进行测试。发现系统参数partitionNum为20，ALS的应用参数dataFeature（特征数量）为20，sampleFraction（采样率）为0.01，numIteration（最大迭代次数）设为20时，会出现StackOverflowError错误。

#### Graph应用

对Graph中的PageRank应用，在10GB的倾斜数据、1百万顶点、2千万条边的数据，和收敛精度为0.001的参数配置下，会出现内存溢出的错误。

下面对上述问题进行简单分析。PageRank使用以顶点为中心的迭代模型进行计算，其需要两个步骤：1）从邻居顶点获取消息；2）更新值，并将更新后的值传递给邻居顶点。然而，如果此时的数据是倾斜的，即一个顶点的邻居顶点格外多，那么这个顶点就需要向更多的顶点传递消息。再加上数据量过大，收敛精度过低，需要进行的迭代次数就会增加，而每次迭代又需要传递大量的数据。在这种情况下，内存溢出错误就会很容易发生。

# 结束语

本节总结了全文工作，对论文做出的主要贡献进行了阐述，同时对未来需要进一步研究的工作进行了工作展望。

## 论文贡献

本文设计并实现了大数据系统应用的可靠性测试基准框架，其主要贡献包括以下几点：

1. 本文对大数据系统及应用的可靠性问题以及现有的测试基准进行了研究，总结了当前大数据系统及应用面临的可靠性测试方面的需求，从而提出了大数据系统应用的可靠性测试基准。
2. 本文选取了SQL查询、大规模图分析以及机器学习应用类别中使用广泛、且具备一定的计算特性的典型应用作为基准应用。
3. 本文提出了针对大数据应用的异常数据生成方法。定义了异常数据的概念和数据的多种随机分布形式。通过分析应用程序特征，提出应用计算特性与数据异常特征对应关系，并给出了特定应用的异常数据生成规则。
4. 提出了一种针对大数据应用的参数组合测试方法。该方法首先分析了参数之间的相关性，并采用贪心算法对系统和应用参数进行组合空间削减测试。参数组合空间削减需要满足参数独立性以及参数取值的正负相关性，对于不符合上述要求的，本文提出一种探测性参数验证方法，通过指数增长的慢启动方式来确定最差的资源占用的参数取值。
5. 最后，论文介绍了大数据系统的可靠性测试基准的设计，并在Spark系统上进行实现。通过对Spark上的基准应用进行可靠性测试，目前已经在6个应用中发现了三种类型的错误（如，内存溢出错误、运行超时以及计算结果错误），并给出了测试报告。并通过测试验证了本文提出的异常数据生成以及参数组合测试方法的可用性。

## 未来工作展望

本文完成了大数据系统应用的可靠性测试基准框架的设计与实现，目前支持Spark系统的可靠性测试。本文在未来工作中的研究重点包括以下几点：

* 提供对更多大数据系统应用的可靠性测试支持

目前，可靠性测试基准平台提供了对Spark中多种应用类型的可靠性测试支持。考虑到测试基准的通用性，平台还应支持其他的较为流行的大数据系统的可靠性测试，例如Flink、Hadoop等。

* 提供流式负载生成方法

目前，流式应用使用越来越广泛，并且流式应用对数据的流速和流量较为敏感。为了更好地提供对流式应用的可靠性测试支持，需要提供一种流式负载生成方法，来产生不同流速的高并发负载，以测试流式应用应对流速突变场景的可靠性。

* 提供测试在线监控

目前，可靠性测试只能在测试全部结束之后提供一个包含基本测试信息的测试报告，无法提供系统在测试过程中详细的资源占用情况。考虑到测试人员对系统运行实际情况的关注，平台还需提供在线监控界面，实时的为用户提供测试过程中的资源使用信息。

参考文献

1. 崔星灿, 禹晓辉, 刘洋,等. 分布式流处理技术综述[J]. 计算机研究与发展, 2015, 52(2):318-332.
2. 孙大为, 张广艳, 郑纬民. 大数据流式计算:关键技术及系统实例[J]. 软件学报, 2014, 25(4):839-862.
3. J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” in 6th Symposium on Operating System Design and Implementation (OSDI), 2004, pp. 137–150.
4. Apache Storm. <http://storm.apache.org/>.
5. Apache Hadoop. <http://hadoop.apache.org/>.
6. Apache Spark. <http://spark.apache.org/>
7. P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. IEEE Data Engineering Bulletin, 2015.
8. Lijie Xu, Wensheng Dou, Feng Zhu, Chushu Gao, Jie Liu, Hua Zhong, Jun Wei. A Characteristic Study on Out of Memory Errors in Distributed Data-Parallel Applications. In the 26th IEEE International Symposium on Software Reliability Engineering (ISSRE 2015).
9. S. Li, H. Zhou, H. Lin, T. Xiao, H. Lin, W. Lin, and T. Xie, “A characteristic study on failures of production distributed data-parallel programs,” in 35th International Conference on Software Engineering (ICSE), 2013, pp. 963–972.
10. S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, “Analysis of traces from a production mapreduce cluster,” in 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010, pp. 94–103.
11. <http://lqding.blog.51cto.com/9123978/1770012>.
12. <http://www.aboutyun.com/thread-19670-1-1.html>.
13. <http://www.oschina.net/question/2657298_2154166>.
14. Interlandi, Matteo, et al. "Titian: Data provenance support in spark."Proceedings of the VLDB Endowment 9.3 (2015): 216-227.
15. Gulzar, Muhammad Ali, et al. "Bigdebug: Debugging primitives for interactive big data processing in spark."Proceedings of the 38th International Conference on Software Engineering.” ACM. 2016.
16. HiBench. <https://github.com/intel-hadoop/HiBench>.
17. BigSQL benchmark. <https://amplab.cs.berleley.edu/benchmark/>.
18. Spark-perf. <https://github.com/databricks/spark-perf>.
19. Capotă, Mihai, et al. "Graphalytics: A big data benchmark for graph-processing platforms." Proceedings of the GRADES'15. ACM, 2015.
20. Agrawal, Dakshi, et al. "SparkBench–A Spark Performance Testing Suite."Technology Conference on Performance Evaluation and Benchmarking. Springer International Publishing, 2015.
21. Sun DW, Zhang GY, Zheng WM. Big data stream computing：Technologies and instances. Ruan Jian Xue Bao/Journal of Software, 2014,25(4):839-862.
22. 王彦明, 奉国和, 薛云. 近年来 Hadoop 国外研究综述[J]. 计算机系统应用, 2013 (6): 1-5.
23. Iqbal M H, Soomro T R. Big data analysis: Apache storm perspective[J]. International Journal of Computer Trends and Technology, 2015: 9-14.
24. Zaharia M, Chowdhury M, Franklin M, Shenker S, Stoica I. Spark: Cluster computing with working sets. HotCloud 2010. 2010.
25. 程学旗, et al. "大数据系统和分析技术综述." 软件学报 25.9 (2014): 1889-1908.
26. <http://www.infoq.com/cn/articles/hadoop-storm-samza-spark-flink>.
27. <https://en.wikipedia.org/wiki/Machine_learning>.
28. Rosenberg, Linda, Ted Hammer, and Jack Shaw. "Software metrics and reliability." 9th International Symposium on Software Reliability Engineering. 1998.
29. H. Zhou, J.-G. Lou, H. Zhang, H. Lin, H. Lin, and T. Qin, “An empirical study on quality issues of production big data platform,” in 37th International Conference on Software Engineering (ICSE), 2015.
30. H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria, “What bugs live in the cloud? A study of 3000+ issues in cloud systems,” in Proceedings of the ACM Symposium on Cloud Computing (SoCC), 2014, pp. 7:1– 7:14.
31. BigBench:Ghazal, Ahmad, et al. "BigBench: towards an industry standard benchmark for big data analytics." Proceedings of the 2013 ACM SIGMOD international conference on Management of data. ACM, 2013.
32. BigDataBench:L. Wang, et al, “Bigdatabench: A big data benchmark suite from internet services,” in 20th IEEE International Symposium on High Performance Computer Architecture (HPCA), 2014.
33. StreamBench：Lu, Ruirui, et al. "Stream bench: Towards benchmarking modern distributed stream computing
34. Pavlo, Andrew, et al. "A comparison of approaches to large-scale data analysis." in Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD), 2009.
35. Huppler K. The art of building a good benchmark[C]//Technology Conference on Performance Evaluation and Benchmarking. Springer Berlin Heidelberg, 2009: 18-30.
36. Streaming benchmark: <https://github.com/yahoo/streaming-benchmarks>.
37. DataArtisans: <https://github.com/dataArtisans/performance>.
38. AaltoStreamBench: <https://github.com/wangyangjun/StreamBench/tree/master/StreamBench>.
39. flink-perf: <https://github.com/project-flink/flink-perf>.
40. <http://yizhen-blog.com/11-the-uniform-distribution-exponential-distribution-and-normal-distribution/>.
41. Poisson Distribution: <https://en.wikipedia.org/wiki/Poisson_distribution>.
42. Guo, Yong, et al. "How well do graph-processing platforms perform? an empirical performance evaluation and analysis." Parallel and Distributed Processing Symposium, 2014 IEEE 28th International. IEEE, 2014.
43. Nie, Changhai, and Hareton Leung. "A survey of combinatorial testing." ACM Computing Surveys (CSUR) 43.2 (2011): 11.
44. YAN, Jun, and Jian ZHANG. "Combinatorial testing: Principles and methods." Journal of Software 6 (2009): 004.
45. Cohen, Myra B., Matthew B. Dwyer, and Jiangfan Shi. "Exploiting constraint solving history to construct interaction test suites." Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007. IEEE, 2007.
46. Kuhn DR, Reilly MJ. An investigation of the applicability of design of experiments to software testing. In: Caulfield M, ed. Proc. of the Annual NASA/IEEE Software Engineering Workshop (SEW). Los Alamitos: IEEE Press, 2002. 91−95.
47. Luc Devroye, Non-Uniform Random Variate Generation (Springer-Verlag, New York, 1986), chapter 10, page 505 <http://luc.devroye.org/rnbookindex.html>.
48. 胡海波, 王 林. "幂律分布研究简史." 物理 34.12 (2005): 0-0.
49. 罗由平, et al. "基于幂率分布的社交网络规律分析." 计算机工程 41.7 (2015): 299-304.
50. Padayachee, I., P. Kotze, and A. van Der Merwe. "ISO 9126 external systems quality characteristics, sub-characteristics and domain specific criteria for evaluating e-Learning systems." The Southern African Computer Lecturers’ Association, University of Pretoria, South Africa (2010).
51. Chua, B. B. & Dyson, L. E. 2004. Applying the ISO 9126 model to the evaluation of an e-learning system. ASCLITE Conference Proceedings.
52. Losavio, F., Chirinos, L., Matteo, A., Levy N., RamdaneCherif, A. 2004. ISO quality standards for measuring architectures. The Journal of Systems and Software. 72, 209– 223.
53. IEEE Standards Coordinating Committee. "IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). Los Alamitos."CA: IEEE Computer Society (1990).
54. Gillies, Alan. Software quality: theory and management. Lulu. com, 2011.
55. Pressman, R. S. 2010. Software Engineering A practioner’s Approach. McGraw-Hill International Edition.
56. Dryad: <https://en.wikipedia.org/wiki/Dryad>.
57. Apache Hive: <http://hive.apache.org/>.
58. Apache Impala: <http://impala.apache.org/>.
59. Zipf Distribution: <https://en.wikipedia.org/wiki/Zeta_distribution>.
60. Spark SQL issues: <https://issues.apache.org/jira/browse/SPARK-11576>.
61. Spark ML issues: <https://issues.apache.org/jira/browse/SPARK-16008>.
62. Spark user list. MLlib Logistic Regression. <http://apache-spark-user-list.1001560.n3.nabble.com/MLlib-Logistic-Regression-run-out-of-memory-td22210.html>.
63. TPC-H: <http://www.tpc.org/tpch/default.asp>.
64. TPC-DS: http://www.tpc.org/tpcds/default.asp.

发表文章

[1] Yingying Zheng, Lijie Xu, Wei Wang, Wei Zhou and Ying Ding. A Reliability Benchmark for Big Data Systems on JointCloud. The Eighth International Workshop on Joint Cloud Computing (JCC), 2017.

致谢

三年的硕士生涯是我人生中最重要的几个阶段之一。这三年中，在老师和同学们的帮助下，我得到的不只是知识上的积累，更有生活上的进步。藉此论文完成之际，我谨向那些帮助过我以及与我合作过的老师和同学们送上最真诚的谢意。

首先，我要衷心的感谢我的导师叶丹老师！在研究生期间，无论是在雁栖湖校区上课还是在实验室学习工作，叶老师都给了我无微不至的关怀和鼓励。在学习工作中，叶老师耐心的指导和帮助我；同时，叶老师也会关心我生活上遇到的问题，并给予我鼓励和帮助。在论文撰写期间，叶老师也会提醒我论文撰写中需要注意的事项，避免不必要的弯路。叶老师为人亲和友善、平易近人，并有着严谨的科研态度，这都是值得我学习的地方。

同时，我还要特别的感谢项目组的王伟老师。在项目组工作学习中，王老师会给出非常有见地的意见，帮助我解决项目中遇到的棘手的问题。同时，王老师也会在生活中给予我关心和鼓励，既是良师更是益友。另外，对于我的论文，王老师从论文选题、研究内容以及论文的写作方面都给予了我悉心的指导，这些都让我受益匪浅。王老师非常敬业，对待科研有着无限的热忱，是我学习的榜样。

另外，我还要特别感谢许利杰师兄。利杰师兄在我的论文撰写过程中给予了我极大的帮助和指导。师兄的学习方法和做事态度让我学到了很多，并让我少走了很多弯路。利杰师兄对科研充满热情，同时也在其他领域有所成就，他的敬业态度更让我深受感染。

感谢中心的各位老师为我们创造的良好的科研环境，让我们能够沉下心来专心研究。同时，还要感谢老师们对我的指导，你们的意见和建议都给了我莫大的帮助。还要感谢项目组的袁鑫晨、江勇、唐震、任仲山、支孟轩、张舒扬等师兄。

感谢数据网格组的段世凯和刘财政同学。在项目组工作和学习中，无论我遇到了什么难题，他们都会给予我帮助，并给出切实可行的意见，这让我能够很快的进步。感谢段世凯同学经常耐心的给我讲解技术上遇到的难题，感谢刘财政同学在我迷茫的时候与我一起讨论解决方案。感谢姬强、徐培兴、崔光范、倪嘉志、罗松磊、陈晓旭、段世凯、梁建华、唐弘胤、王子勇、刘财政、佘艺、邬登峰和燕东同学。研究生三年，我们一起学习交流，互相帮助，共同进步。另外还要感谢沈雯婷、刘重瑞、赵伟、王建飞、亢良一等师弟师妹们的帮助。感谢我的室友燕季薇同学和我的饭友靖二霞同学对我生活的关心。同时还要特别感谢我的父母对我的支持和帮助。

最后，向所有给予我关怀和帮助的人们献上深深的谢意。