目录

[1 绪论 3](#_Toc478238513)

[1.1 研究背景 3](#_Toc478238514)

[1.2 研究内容 4](#_Toc478238515)

[1.3 论文组织 5](#_Toc478238516)

[2 大数据系统及应用可靠性相关工作 6](#_Toc478238517)

[2.1 大数据系统及应用 6](#_Toc478238518)

[2.1.1 大数据系统 6](#_Toc478238519)

[2.1.2 大数据应用 9](#_Toc478238520)

[2.2 大数据系统及应用的可靠性问题 10](#_Toc478238521)

[2.2.1 可靠性定义 10](#_Toc478238522)

[2.2.2 可靠性问题分析 11](#_Toc478238523)

[2.3 测试基准框架研究现状 12](#_Toc478238524)

[3 可靠性测试基准设计 15](#_Toc478238525)

[3.1 基准需求 15](#_Toc478238526)

[3.2 测试基准组成 16](#_Toc478238527)

[3.3 基准应用 17](#_Toc478238528)

[3.3.1 应用类型 18](#_Toc478238529)

[3.3.2 工作负载 18](#_Toc478238530)

[3.4 测试数据 21](#_Toc478238531)

[3.5 基准执行 23](#_Toc478238532)

[3.6 测试度量 23](#_Toc478238533)

[4 可靠性测试基准关键技术 24](#_Toc478238534)

[4.1 数据生成方法 24](#_Toc478238535)

[4.1.1 异常特征 24](#_Toc478238536)

[4.1.2 数据概率分布 25](#_Toc478238537)

[4.1.3 数据生成 27](#_Toc478238538)

[4.2 参数组合测试方法 33](#_Toc478238539)

[4.2.1 参数配置 33](#_Toc478238540)

[4.2.2 组合空间削减测试 35](#_Toc478238541)

[4.3 流式负载生成方法 41](#_Toc478238542)

[5 可靠性测试基准框架实现 42](#_Toc478238543)

[5.1 系统架构 42](#_Toc478238544)

[5.2 系统实现 43](#_Toc478238545)

[5.2.1 系统总体设计 43](#_Toc478238546)

[5.2.2 数据生成器 47](#_Toc478238547)

[5.2.3 组合参数发生器 53](#_Toc478238548)

[5.3 可靠性测试应用验证 55](#_Toc478238549)

[5.3.1 实验环境 55](#_Toc478238550)

[5.3.2 参数配置 55](#_Toc478238551)

[5.3.3 实例及分析 56](#_Toc478238552)

[6 结束语 62](#_Toc478238553)

[6.1 论文贡献 62](#_Toc478238554)

[6.2 未来工作展望 62](#_Toc478238555)

[参考文献 65](#_Toc478238556)

[发表文章 66](#_Toc478238557)

[致谢 68](#_Toc478238558)

# 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，从而更好的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。

为了应对海量数据产生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MapReduce[]、Storm[3]、Hadoop[4]、Spark[5]、Flink[6]等，这些分布式的大数据处理框架已被广泛应用到了工业界中的大数据应用的处理分析中。常见的大数据应用有SQL操作（如，Scan、Join等）、大规模图分析（如，PageRank、ConnectedComponents等）、流式数据处理（如，多个流的join操作等）以及机器学习（如，Kmeans、SVM等），这些应用针对不同的大数据场景提供了解决方案。这些系统针对大数据处理给出了自己的解决方案，可以应对大数据所呈现的易失性、突发性、无序性等特点，但是仍然存在着许多关键性的问题。应用在处理大数据时，经常出现内存溢出、IO异常、任务超时等运行时错误，以及在流处理过程中的数据完整性和计算结果错误等，这些错误会直接造成应用执行失败。

分析产生上述问题的原因主要有以下两点：（1）系统本身是否可靠，即部署好或升级后的系统是否存在缺陷；（2）系统提供的应用是否可靠，即开发好的系统应用是否存在潜在的运行时错误。其中，系统错误，即部署好或升级后的系统是否存在缺陷，包括硬件错误和软件错误。硬件错误包括CPU、内存、网络以及硬盘等错误；软件错误包括特定的逻辑错误、数据竞争等。另外，对于应用错误，已有的实证研究发现：（a）对于IO异常、内存溢出及任务超时等运行时错误，其产生原因[7][8][9]包括不恰当的配置、数据异常以及用户代码缺陷等。不恰当的配置包括输入的数据块过大，分区数目过小，分区函数不均衡等；数据异常包括异常的输入数据、中间结果或输出数据，例如倾斜数据以及高维数据等；用户代码缺陷包括内存泄漏、较高的时间或空间复杂度等。（b）对于数据和计算完整性，根据博客、论坛等的实证分析[10][11][12]，其主要错误有数据错误（如数据丢失、数据重复等）及状态错误（状态丢失、快照错误等），产生原因主要有数据流速过快、Task失效、快照恢复机制不完善等。

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。测试是一种常用的且有前景的方法，但是目前的解决方案[13][14]大都是出现问题之后，再针对某一类应用及问题进行分析诊断，当前还没有一个通用的检测方法能够提前发现系统、应用和数据的潜在问题。另外，现有的大数据平台的测试基准如HiBench[15]、BigSQL benchmark[16]、Spark-perf[17]、Graphalytics[]以及SparkBench[]等，关注的焦点主要是特定的大数据平台上的性能及扩展性的基准测试，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，它们不能直接用于检测潜在错误。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。针对上述问题以及研究现状，本文将研究如何开发一个针对大数据系统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，支持SQL操作、大规模图分析、流式数据处理以及机器学习中使用广泛的典型应用的可靠性测试；并提供可配置的数据生成以及自动化的参数组合测试。
2. 集成数据生成、数据存储、参数组合测试以及报告生成模块，构建一站式大数据系统的可靠性测试基准平台；将开发人员从算法学习、测试脚本编写等复杂繁琐的工作中抽离出来。
3. 构建基于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

本文设计并实现了大数据系统的可靠性测试基准框架。为了检测潜在的应用程序或系统错误，可靠性测试基准通过生成异常输入数据，并结合特定的系统及应用的配置来测试应用程序。与性能测试基准不同，该基准（1）根据应用特性生成异常输入数据，同时（2）基于参数特征减少参数组合空间。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. 提出了针对大数据应用的异常数据生成方法。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应用的异常参数生成方法。采用贪心算法对系统和应用参数进行组合空间削减测试。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。
3. 提出了针对流式大数据应用的高负载生成方法。该方法通过构建多个负载发生客户端，并通过并行划分数据流量、流速的方法，来生成高负载的应用数据。

最后，论文介绍了大数据系统的可靠性测试基准的框架的实现，并将该可靠性测试基准框架应用在Spark系统上。通过对Spark上的典型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），并给出了测试结果及部分错误分析。

## 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。首先，介绍了大数据系统和大数据应用的研究现状；接着，介绍了大数据系统及应用面临的可靠性问题，包括可靠性定义以及可靠性问题的分析；最后，介绍了测试基准框架的研究现状。

第三章介绍可靠性测试基准的设计。分别介绍了基准需求，基准应用，测试数据，基准的执行以及测试度量。

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。最后介绍了大数据系统应用流式负载生成方法。

第五章介绍大数据应用可靠性测试基准框架实现。首先，介绍了系统架构；然后，介绍了系统实现，包括系统总体设计、数据生成器以及组合参数发生器；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，主要包括论文的工作、论文贡献以及对未来工作的展望。

# 大数据系统及应用可靠性相关工作

本章介绍了大数据系统及应用的发展现状，可靠性问题分析以及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分析用户需求和进行风险管理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，并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的发展起来。

### 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[]。因此，现有的大数据处理系统主要分为了批处理系统、流处理系统以及批处理和流处理的混合系统。

* + 批处理系统

当前比较流行的批处理系统有Apache Hadoop等。Apache Hadoop是一种以MapReduce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。通过共享大规模系统集群，MapReduce拥有良好的可伸缩性。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键/值对；（2）reduce阶段，通过reduce()函数，将中间结果键值相同的键/值对合并。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，通过键/值对代替关系表[]，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[][]等。Apache 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。一个Storm作业只需实现一个Topology及其所包含的Spout与Bolt，并通过指定连接方式便可满足流式作业的需求。Storm使开发人员能够以任何编程语言虚拟地开发他们的逻辑，这样可以支持stdin/stdout中的基于JSON的协议通信。Storm的主要特点是（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、可靠的处理消息：Storm提供“至少一次”的语义保障，任务失败时，从数据源恢复数据；（3）扩展性好：Storm提供多个线程和进程，并行的进行流式计算，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[]、Apache Flink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。

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[]，这使得Spark在机器学习等应用中要比Hadoop快10倍~100倍。同时Spark Streaming可以亚秒级增量的对流进行缓冲，然后将这些缓冲作为小规模的固定数据集进行批处理[]。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？是Spark的生态系统。



图表 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），提供支持流处理和批处理两种类型应用的功能，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额内存管理机制，可以灵活的避免频繁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？是Flink的生态系统。



图表 2-2 Flink生态系统

图？和图？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，在迭代中可以显著的减少计算。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。



图表 2-3 Flink计算迭代流程



图表 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？。

表格 2-1 大数据系统对比表格

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

在上述大数据系统中，Spark和Flink不仅提供了丰富的处理类型，同时有着更加丰富的生态系统，并针对机器学习、交互式查询、大规模图计算等任务提供了支持。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，并且也在飞速的发展着。本文后续将Spark和Flink作为主要研究对象。

### 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典型应用。

* + 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。社交网络中的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。

由此可见，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的图数据分析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都提供了对图算法的支持。

* + 机器学习

机器学习[]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，学习对象需要不断的与环境进行交互，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算法（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信息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[SparkBench]：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很高的要求。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是真正的流式处理框架，其提供的DataStream API为流式应用提供了接口支持。

## 大数据系统及应用的可靠性问题

### 可靠性定义

IEEE 610.12-1990将可靠性定义为“系统或组件在规定的时间内以及在规定的条件下，执行其所需功能的能力”。在大数据系统运行应用中，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，大数据系统框架执行该应用的能力”。

对于大多数的项目和软件开发者来说，可靠性等于正确性，即它们正在测试并发现和修复 “bug”的数量[]。而在大数据系统测试环境下，系统是否可靠可以看做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。这些错误不单纯是代码的逻辑错误引起的“bug”，可能是应用本身（如，不恰当的配置、数据异常以及用户代码缺陷等）造成的，也可能是系统框架（如，Spark、Flink等）执行机制引起的。

本文将在下一节对大数据系统面临的可靠性问题展开进一步的分析。

### 可靠性问题分析

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，如Web索引建立、日志挖掘、机器学习、大规模图分析等。然而这些应用在处理大数据时，经常会出现内存溢出、IO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。大数据应用及系统的可靠性已经成为学术界和工业界的关键问题。

常见的大数据应用，在大数据系统中，通常可以表示为（输入数据，参数，用户代码）。输入数据通常作为数据块存储在分布式文件系统上；参数包括系统特定配置（例如，输入块大小，分区号）和应用特定配置（例如，K-means的集群k）；用户代码指用户定义的函数，如map()，reduce()和join()，它们处理输入数据或中间结果。

已有的研究工作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），也包括数据异常和应用缺陷（如参数配置不当、代码缺陷等）。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。

已有的关于大数据系统的可靠性研究主要有以下几点：

1）在MapReduce应用内存使用问题方面：由Xu[7]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，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[8]等人研究了250个SCOPE job（运行在微软的Dryad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，不正确的行格式等等。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）在大数据系统运行错误方面：Kavulya等人[9]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IO异常。Zhou等人[]研究了大数据平台在微软中存在的问题，他们发现36％的问题是由系统侧缺陷引起的，其中2个问题（1％）是内存问题。

4）在云系统中的开发和部署方面：Gunawi等人[10]研究了云系统上部署的系统（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（OOM）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（OOM）错误（用户并行创建了数千个小文件）。这些研究帮助我们设计异常数据发生器和配置发生器。

通过上述研究发现，大数据框架在运行时产生错误的原因主要有以下三点：

1. 数据异常：如数据维度过高、数据倾斜等。
2. 不恰当的参数配置：如系统分配内存的大小、reducer数目太小、决策树深度过大等。
3. 用户代码缺陷：如时间或是空间复杂度太高、有内存泄漏的风险等。

## 测试基准框架研究现状

随着大数据系统及应用的广泛使用，针对大数据系统的测试基准也发展起来。常见的针对大数据系统的测试基准有BigBench[]、BigDataBench[]、HiBench[]和SparkBench[]等；针对特定的大数据应用的测试基准包括Spark SQL benchmark[]（SQL查询）、Graphalytics[]（大规模图分析）、StreamBench[]（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大数据。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，并从速度（即作业运行时间）、吞吐量（即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）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 [29]，BigDataBench [26]和HiBench [23]等其他测试基准都仅涵盖了少量的Spark支持的工作负载，没有全面覆盖Spark支持的全套应用类型。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 [15]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[12]来比较Spark、Hive以及Impala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，Apache Storm、Apache 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。

表？总结了上述介绍的大数据系统及应用的测试基准的特性。

表格 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** | 合成数据集 | 关系查询 | DBMS、Hadoop | 性能测试 |
| **BigDataBench** | 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** | 真实数据集 | 微基准、Web搜索、机器学习 | Hadoop、Hive | 性能测试 |
| **SparkBench** | 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** | 真实数据集、合成数据集 | 大规模图计算应用 | GraphX、Giraph、Neo4j | 性能测试 |
| **StreamBench** | 合成数据集 | 基本操作、join、迭代 | Storm、Flink、Spark Streaming | 性能测试 |

* + 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及应用的性能提供了基准测试，没有考虑大数据系统及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正常数据以及固定的配置来测试大数据系统的性能，缺乏测试的多样性。

## 测试基准需求

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。Huppler[]提出，一个成功的基准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[]等人补充测试基准的新属性有如下几点：

（1）简单易用和自动化。测试基准需要易于理解和部署，并能通过适度的配置便可自动化的执行和分析。基准的自动化特性对于大数据系统（如，Spark生态系统）的快速演变是至关重要的。

（2）综合性。测试基准需要满足大数据系统中应用类型的多样性。大数据系统中的不同操作在某些数据以及参数配置下，可能会有不同的可靠性表现。因此测试基准应该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）可伸缩性。测试基准需要应对大数据系统的快速发展。因此，测试基准应该允许用户添加新功能或扩展现有功能，并且能够扩展到相应的社区可能开发的新功能。

（4）可移植性。测试基准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，且能方便的移植到其他操作系统。

（5）可扩展性。测试基准需要满足大型分布式系统的测试需求，能够生成数据量足够大且变化多样的数据集。

与上述性能测试基准的属性特点相近，可靠性测试基准除了满足上述属性之外，还应该满足以下几点：

（1）测试数据集更加丰富。为了发现大数据系统可能存在的问题，测试基准需要提供更加丰富多样、能够满足应用出错特征的数据集。

（2）测试多样性。为了更加方便快速的发现潜在的可靠性问题，测试基准需要提供满足不同配置的、更加多样的测试。同时提供自动化的测试，降低测试基准的使用难度，并能够快速的定位到容易出错的配置。

（3）负载突发性。对于流式应用，可靠性测试基准需要提供流速可变的流式负载，并通过突发性的负载发现大数据系统可能存在的系统或应用错误。

# 可靠性测试基准设计

为了能够提前发现大数据系统应用存在的可靠性问题，本章设计了大数据系统应用的可靠性测试基准。首先，介绍了基准提供的典型应用类型及工作负载；其次，介绍了不同应用的测试数据；接着，介绍了基准执行过程；最后，介绍了可靠性测试度量指标。

## 基准应用

本节介绍了可靠性测试基准应用以及不同应用类型中工作负载的构造。基准应用选取了大数据系统中使用广泛的典型应用。

### 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系统测试基准中提供的应用类型[]，选取了SQL查询（SQL）、大规模图分析（Graph）、机器学习（Machine Learning）以及流式应用（Streaming）中的若干应用作为可靠性测试基准的基准应用。表？列出了可靠性测试基准选取的基准应用类型及其计算属性。

表格 3-1 可靠性测试基准的基准应用

|  |  |  |
| --- | --- | --- |
| **类别** | **应用** | **计算属性** |
| **SQL** | Scan | 过滤 |
| Aggregate | 聚合 |
| Join | 关联 |
| Mix | 过滤、聚合、关联 |
| **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 |
| **Streaming** | WindowJoin | 流式聚合 |

### 工作负载

本文针对上述基准应用类型，提供了下述具体的工作负载，即待测的测试用例。下面将分别针对SQL、Graph、Machine Learning以及Streaming进行介绍。

* + SQL

SQL在当前的大数据处理中无处不在，是一种使用广泛的查询语言。同时，有更加广泛的大数据生态系统支持SQL查询。现有的数据库评测基准，如Big Data Benchmark，TPC-H[]和TPC-DS[]等，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，甚至不同应用领域的SQL查询，通常可以由基本的查询子句构造出来。

为此，本文根据Pavlo [19]等人对测试基准的研究，参照其测试方案设计了两个表，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记录信息。两张表的属性以及具体含义如表？所示。同时，本文选取了SQL查询中的基本子句来构建工作负载。

表格 3-2 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 |

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** *sourceipaddr*,*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 *sourceIPAddr*, *totalRevenue*, *avgPageRank* FROM (SELECT *sourceIPAddr*, AVG(*pageRank*) as *avgPageRank*, SUM(*adRevenue*) as *totalRevenue* FROM Rankings AS *R*, Uservisits AS *UV* WHERE *R.url* = *UV.destinationURL* AND *UV.avgTimeOnSite* BETWEEN 30 AND 70 GROUP BY *UV.sourceIPAddr*) ORDER BY totalRevenue DESC LIMIT 10。

针对上述SQL中的基准应用，可靠性测试基准将针对每一个应用执行单独的测试，并通过指定测试次数实现SQL应用的可靠性测试。

* + Graph

图是使用比较广泛的数据模型。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系统测试基准中提供的图应用，选取了如下几种应用作为可靠性测试基准的工作负载。

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。该算法在图计算中，通常使用迭代模型计算。例如，使用顶点为中心的迭代模型进行PageRank计算过程中，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。持续的进行迭代，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。该算法在图计算中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中的基准应用进行测试时，首先需要生成图数据到存储系统，然后Graph中的基准应用会以该数据为输入进行测试。Graph应用测试可通过提供不同大小或顶点分布的图数据文件来测试Graph应用的可靠性。

* + Machine Learning

机器学习在大数据处理分析中应用较为广泛，为结果预测、信息推荐等提供了算法支持。目前使用较为广泛的算法有回归、分类、聚类等。现有的大数据系统中，Spark的MLlib以及Flink的ML都提供了相应的算法实现。本文依据现有的大数据系统中的机器学习算法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（逻辑回归）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法算法来计算矩阵分解。

1. RandomForest

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（支持向量机）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），同时针对不同的基准应用提供不同的参数组合，来进行参数组合测试。

* + Streaming

随着智慧城市、智能装备、智能制造、极端交易等大数据处理场景的涌现，越来越多的数据需要实时地、流式地处理，以满足数据的时效性。

本文主要选择了流式应用中使用比较普遍的WindowJoin作为流式工作负载。

WindowJoin提供了两个使用窗口模式的数据流的join操作。与批处理join操作不同的是，流式处理的join操作会受到流速以及流量的影响。

针对Streaming的基准应用，通过提供不同流速及流量的并发流式负载，来测试流式应用的可靠性。

## 测试数据

本节将针对上一节提出的工作负载，从SQL、Graph以及Machine Learning三种应用类型给出测试数据的设计。

* + 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和sourceIP字段从特定范围内生成随机值，而其他字段（如countryCode、languageCode等）采用真实世界数据集进行挑选。

SQL的数据生成可以通过配置相应的参数生成常规数据或倾斜的异常数据。数据通过集群分布式的生成，且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。其中Vertex定义为二元组，即 (vertexID,value)，其中vertexID是标识顶点的编号，value是顶点的值。Edge定义为三元组，即 (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中不同的应用有不同的数据集。

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（label index0:value0 index1:value1）。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。各个样例不一定包含全部特征，为稀疏数据。

K-means是无监督学习，使用不带label的数据。数据的格式为（value0 value1 value2…）。value表示各个特征值。每个样例都包含全部特征，为稠密数据。

ALS数据使用Netflix数据集（电影评分数据集）格式（userId itemId rating timestamp）。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参数（如实例数、维度、分布方式等），根据配置信息可以生成满足不同维度、不同分布、不同规模的数据。这些数据同样存储在HDFS的节点上。

## 基准执行

本节将介绍可靠性测试基准的执行流程，如图？所示。

（1）在准备好待测系统以及存储系统的环境后，用户首先在可靠性测试基准中配置系统信息，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，可以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。



图表 3-1 基准执行阶段

## 测试度量

在性能测试基准中，经常以吞吐量、响应延迟作为度量指标。在软件运行的可靠性测试中，已有的方法通常使用故障平均时间来衡量系统的可靠性[]。由于大数据系统运行在多节点的集群环境下，系统的可靠性无法使用常规的故障平均时间来衡量。本文将以下异常现象称为系统不可靠的评测标准：

1. 性能出现异常。如，在用户给定的时间内，出现无响应、假死等现象。
2. 资源使用异常。如，出现IO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。

由上述系统不可靠的评测标准，本文给出的大数据系统可靠性测试基准的度量指标为：在用户给定的系统配置下，如果出现上述异常现象，则称该系统在给定的系统配置下是不可靠的。

# 可靠性测试基准关键技术

本章就可靠性测试基准中的关键技术及方法进行研究。首先，研究了数据生成方法，定义了异常数据特征，并通过分析基准应用的计算特性与异常数据特征的对应关系，给出了不同应用的异常数据生成规则；其次，研究了参数组合测试方法，提出了组合空间削减策略；最后，研究了流式负载生成方法。

## 数据生成方法

本节主要介绍了异常数据的定义，基准应用的特征分析，以及根据应用特征实现的数据生成方法。

### 异常特征

经研究发现，大数据系统中的错误一般是在极端配置、极端数据下产生的。所以我们需要产生一些极端的异常数据，让系统在高压力、高负载下提前暴露问题。然而，异常数据通常需要符合应用场景的要求，因此需要分析应用特征，进而有针对性的产生特定的异常数据。

* + 异常数据定义

首先，需要定义异常数据特征。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（如矩阵中的0），即增加数据信息量的元素很多；

（3）数据维度高，即用于测试的数据维度过高；

（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些项时，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。

* + 应用特征分析

其次，针对基准应用，抽取其应用特征（数据操作特征，如操作顺序、依赖关系等）。通过对各类应用操作特征的分析，如表？所示，本文给出了各类应用的计算特性以及归纳出的数据异常特征。

表格 4-1 应用特征分析

|  |  |  |
| --- | --- | --- |
| 应用类型 | 计算特性 | 异常特征 |
| SQL | SQL基础查询语句中Scan、Aggregate、Join等应用处理key/value对，其计算复杂度与key的分布相关。 | 数据量大、数据倾斜、数据分布异常 |
| Graph | Graph中的应用大多需要迭代计算，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。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，会产生运行超时等可靠性问题。 | 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等采用矩阵特征作为输入数据，因此计算与矩阵特征（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。此外，随机抽样方法将影响计算结果的稳定性。 | 数据量大、数据稀疏、数据维度高、数据分布异常 |
| Streaming | 通过实证分析流处理系统及应用中存在可靠性问题的原因，包括数据流速过快、Task失效、快照恢复机制不完善等。 | 数据量大，数据流速快 |

根据上表给出的应用特征及异常数据特征的对应关系，可靠性测试基准可以针对给定的新应用，分析其特征，并确定需要生成的异常数据特征，从而生成符合应用场景的极端异常数据。

### 数据概率分布

针对数据分布这一数据特征，本文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[]。

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中：1）与应用相关性比较小的属性列；2）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（无论是什么过程，期望相同且有限，方差有限）的均值服从高斯分布，因此高斯分布符合自然界的分布；在均值和标准差已知的情况下，高斯分布是最大熵分布，保留了最大的不确定性，因此用高斯分布的数据训练出来的模型更加可靠；高斯分布便于公式推导并且无限阶可导。

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正态分布来近似。所以，绝大多数数据处理场景中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[]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。

### 数据生成

通过上述应用计算特性分析以及与数据异常特征对应关系的总结，以及数据概率分布形式，表？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。

表格 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。 | 生成异常分布（如，Zipf分布、泊松分布等混合）的随机数据；生成倾斜数据（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[1]；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。 | 借鉴Graphalytics[2]的方法，生成顶点度异常分布（如Zipf、泊松分布等）、聚类系数或分类程度异常的稀疏图。 |
| 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、分布（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。 |
| Streaming | 提供正常流速的流式数据。 | 提供高流速且满足异常分布的大规模数据。 |

#### 常规数据生成

常规数据生成使用的是真实数据集（如，Machine Learning中的KDD2010、SUSY等）或现有的大数据测试基准提供的数据生成器（如，Graph中使用的LDBC提供的Datagen等）。这些数据符合现实场景中数据的使用特征，是可靠性测试的常规数据。

#### 异常数据生成

异常数据生成是根据应用的异常数据特征进行的。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。针对SQL、Graph、Machine Learning这几种应用类型，下面将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，我们称其为倾斜数据。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他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中的异常数据分布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：可以用于模拟网页访问量、URL访问量等倾斜数据。

在我们设计的表（如表？所示）中，如果把外链数目作为网页排名确定原则，根据经验表明网页中大多数外链都指向极少数的网页，所以网页排名服从齐普夫分布；同样，极少数网站承载了网络上大多数访问流量，URL的访问也遵循齐普夫分布。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？所示



图表 4-1 单参数Zipf分布

下面给出了SQL异常数据的生成方法，如图？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分布形式；最后，对应用产生符合某种分布的异常数据。



图表 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，容易产生运行超时等可靠性问题。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，用于满足幂率分布的图数据的生成。

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分布形式，本文提供了对顶点度的多种分布形式，用于生成满足不同异常分布的图数据。其中，通过Zipf分布来模拟倾斜数据的生成，同时可以混合多种分布，提供异常数据。图？为Graph的异常数据生成。



图表 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。

对于基于原始数据的生成方式，首先进行预处理，即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，根据上述介绍的两种数据格式生成不同维度、实例数、稀疏度即分布形式的数据。其中，稀疏度分为整体的稀疏度和特征的稀疏度；分布形式分为高斯分布、伽马分布、泊松分布、指数分布、均匀分布、混合分布和Zipf分布等。每个样例的同一个属性符合某种分布。混合分布是指混合了以上各种分布的属性集合。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？是为Machine Learning生成异常数据的方式。



图表 4-4 Machine Learning异常数据生成

* + 异常数据生成流程

对于可靠性测试基准提出的典型应用，其异常数据的生成将按照图？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，会产生运行超时等问题。

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，生成大规模的（超过50G）、稀疏的（稀疏度为0.1）、满足Zipf和伽马等混合分布的异常数据。



图表 4-5 异常数据生成流程

* + 异常数据生成的优势

Agrawal[21]等人认为，数据生成器应该具备以下几点特征：

（1）接近实际的：尽管数据是合成的，但应该趋向真实，即可以代表真实数据集。真实的数据集一般具有属性相关性、代表高度倾斜的非均匀分布等特点。

（2）随机但可预测：数据应该足够随机来应对未知的数据集，但同时数据处理的输出必须是可以预测的。

（3）并行和分布式的：大规模数据的生成需要多核服务器并行工作来提高效率。

（4）可扩展的：允许用户从小规模单节点服务器扩展到多节点的大规模集群。

在上述基础之上，本文设计的数据生成根据应用本身的性质，还包括以下几点特征：

（1）接近应用的：通过分析应用本身的特征，发现可能存在的问题，进而有针对性的产生相应的异常数据。

（2）可实证分析的：可从某个可以证实的前提出发，即已经发现的现存的问题，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。

## 参数组合测试方法

通过上一节的数据生成方法生成完输入数据后，接下来就要对基准应用进行测试。由于有些应用除了需要提供系统相关的参数外，应用本身还需要参数提供支持。因此，对于这些应用需要通过组合系统相关和应用相关的配置参数，来对具体的应用进行测试。

本节介绍了系统和应用的参数配置，并使用组合空间削减策略对组合测试进行降维，同时提供了一种用于确定参数取值的探测性参数验证方法。

### 参数配置

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。

* + 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的时CPU内核的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，Spark中，使用Input split（Mapper端分片个数）来确定数据并行度，使用Partition num（Reducer端分片个数）来确定任务并行度。又如Flink中，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。

* + 应用参数

应用参数指的是应用或算法本身运行时需要的参数，例如Graph中ConnectedComponents算法中的maxIteration（最大迭代次数）。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等造成影响。

下面针对3.2节中给出的Graph、Machine Learning等类别中，需要提供应用参数的应用，给出了具体的参数列表，如表？。

表格 4-3 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 |

上表中列出了测试过程中需要应用参数的应用的具体参数信息。这些参数信息在系统运行应用过程中，其中的某些取值可能会影响程序的运行，导致运行时错误。因此，接下来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，尽可能多的发现其中的可靠性问题。

### 组合测试

针对上述给出的系统参数和应用参数，本文采用了组合测试的方法来覆盖可靠性测试中的测试用例。

组合测试[]使用由某些抽样机制生成的覆盖数组来检测待测系统（SUT）在参数交互中触发的错误。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[]。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[]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。

* + 覆盖数组（CA）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。

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：

通过上述使用所有取值组合来进行组合测试的方式，当组合参数测试中参数种类太多时，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，如何削减组合测试的组合空间，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主要手段。

已有的研究[]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[]。因此，在解决组合空间过大的问题时，本文首先研究了参数配置之间的相关性。

* + 参数相关性分析

本文对3.4.1节中列出的Graph以及Machine Learning中的应用参数进行了参数相关性的理论分析，主要研究参数与资源占用（如，CPU、Memory等）是否相关，或参数之间是否会相互影响而对资源占用产生影响。

1. Graph

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大多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？，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。Spark中的GraphX以及Flink中的GElly都提供了这种编程模型或这种模型的变种。

值得关注的是，该模型迭代终止的条件是Workset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Workset不为空，则无法得到正确的结果。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，大多数只与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和空间复杂度都不相关，同时与计算结果正确性也不相关，因此这些参数可以看成是独立的因子。因此，在图应用需要使用的参数中，并不存在参数之间相互作用导致可靠性问题的出现，反而是其中的maxIteration本身会影响计算结果的正确性。



图表 4-6 图算法中迭代计算模型

1. Machine Learning

Machine Learning中的一些机器学习模型不同于Graph中的应用，其具有较多的参数信息，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及参数组合。

RandomForest中有三个对内存使用会有影响的参数，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。比如年龄特征，可以分为大于20岁，小于20岁，这就是一种分法。多种分法就是指可以在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。队列大小是有限制的，假设是1000M，结点大小由features数量\*maxBins\*每个bin的大小决定，假设有10个feature，maxBins为10，每个bin为0.01M。group按照层取，每层有多少结点就由maxDepth和numTrees决定。假设有10棵树，第一层就有10个结点，需要，那么到第8层的时候，森林中有的7次方个结点，需要，大于队列大小，一次性不能一起处理这些结点，需要多加一轮来处理这些结点。所以这里单纯考虑队列内存的使用的话，因为队列内存是一个固定的值，这几个参数的乘积对其有影响。另外，每个参数的增加都会使得运行时间增加，总的使用内存增加。因此，在RandomForest中，参数numTrees、maxDepth和maxBins的相互影响，会对应用运行时的空间复杂度的增大造成影响。所以，参数组合测试时需要将这三个参数的组合看成一个独立的参数。

LogisticsRegression中，由于该应用的输入参数有numIteration（最大迭代次数）以及convergencoTol（误差容忍度），而这两个参数的相互作用会决定应用的运行结束。LogisticsRegression的每次迭代中，都会计算一下误差，误差随着迭代是递减的，当达到设定的最大迭代次数，但还没有达到收敛精度（也就是误差容忍度）时，运行结束；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，改变这两个参数会增减迭代次数，迭代次数改变了，所用的总时间和总空间也会随之增减。

### 参数组合空间削减

参数相关性的确定并非一件容易的事情，而且一些貌似毫不相关的参数可能在某些异常数据下也会导致异常的结果。因此，本节仅以参数相关性作为辅助参考，重点研究参数取值空间削减的方法。

对于应用运行时的参数，本文做了以下两个假设：

1. *n*个参数互相独立
2. 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）。

* + 组合空间削减方法

在满足上述假设基础上，我们提出了一个削减参数组合测试空间的贪心算法，其具体流程如图？：

1. 给定每个参数的取值范围；
2. 选择每个参数的某一临界值进行组合，进行测试，并记录资源使用情况；
3. 改变一个参数的取值为另一临界值，进行测试，并记录资源使用情况；
4. 比较两次参数组合下的资源使用，选择性能较差的那个参数作为固定配置。
5. 返回第2步重复执行。直到出现异常或组合测试结束。



图表 4-7 算法流程图

该方法使用贪心算法，从空矩阵开始，通过一维扩展，即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。图？给出了一个具有三个配置参数的例子。首先，选择三个参数的最小边界取值组合进行测试，即2-1-2；然后，将第一个参数的取值改为最大边界值进行测试，即100-1-2；比较应用在参数组合2-1-2和组合100-1-2下进行测试的资源使用情况，如果组合2-1-2的资源使用比100-1-2低，则固定第一个参数取值为100。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，依次进行测试比较，最后得到d)中给出的最差参数组合100-10-1。





图表 4-8 参数组合空间削减

在满足给定的两个假设条件的基础上，可以采用上述步骤进行组合测试。但这是理想状况下的假设，如果第*i*个参数的*mi*个取值并不满足正相关和负相关，需要通过探测性方式进行参数验证。

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的探测性参数验证方法。

* + 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，只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。通常情况下，的取值不会超过10。

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。

* + 算法可靠性分析

本文提出的组合空间削减算法是一种贪心算法，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，这种类似贪心算法的组合空间削减算法符合可靠性测试的需求。

## 流式负载生成方法

流式负载的生成主要是为流式应用提供高并发、高流速的异常负载。通过构建多个负载发生客户端，并通过并行划分数据流量、流速的方法，来生成高负载的应用数据。流式负载生成的部署如图？所示。

在多负载发生客户端的基础上，流式负载生成的关键就是控制流速。本文提出的流速控制的方法如下：

1. 由用户输入需要产生的流式负载的流速*v*（条/秒）。
2. 负载发生客户端将*v*条数据按照纳秒

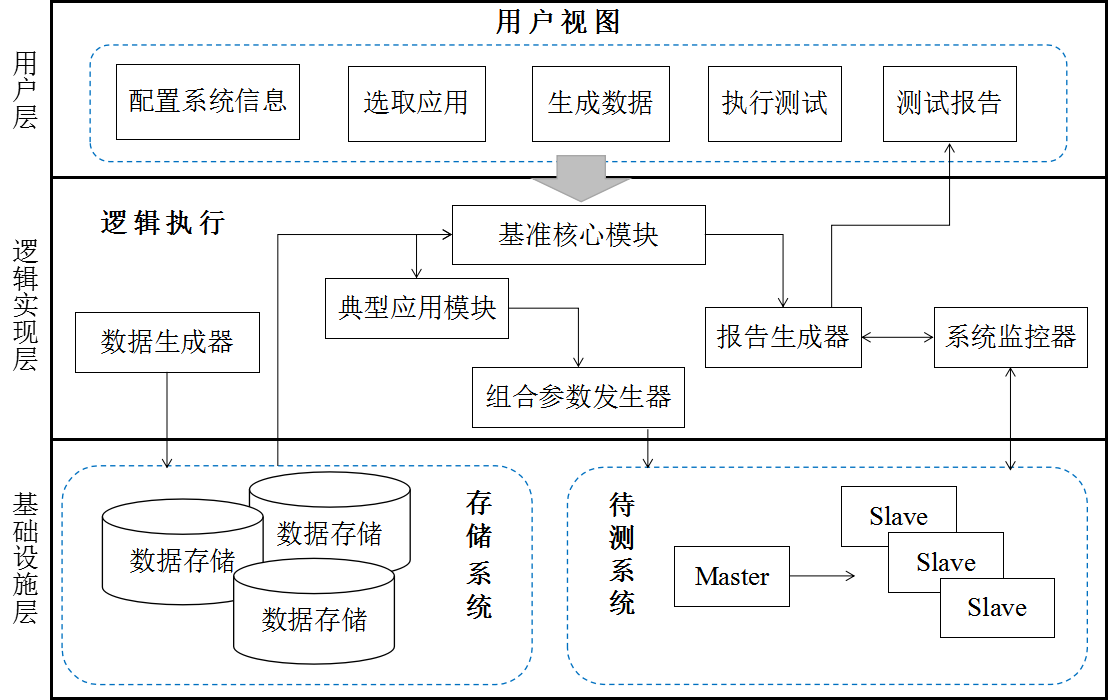
# 可靠性测试基准框架设计与实现

本章介绍大数据系统应用可靠性测试基准框架的设计与实现。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，最后将该可靠性测试基准框架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基准平台的系统架构设计。

大数据系统应用的可靠性测试基准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基准平台。该平台提供Web访问界面，支持用户配置的数据生成，支持自动组合参数测试，同时提供测试报告生成。具体的系统架构如图？。



图表 15 系统架构图

面向大数据系统及应用的可靠性测试基准平台分为三层：用户层、逻辑实现层和基础设施层。

* + 用户层

用户层为用户提供用户视图界面。通过用户视图，用户可以配置待测系统以及存储系统等配置信息，选取典型应用，生成自定义的常规或异常数据，以及执行测试。测试执行结束后，会为用户展示测试后生成的测试报告。

* + 逻辑实现层

逻辑实现层为可靠性测试提供了逻辑执行中使用到的关键模块，包括基准核心模块、典型应用模块、数据生成器、组合参数发生器、系统监测器以及报告生成器。这些模块共同支撑了整个逻辑执行的流程。

1. 基准核心模块：是整个逻辑实现层的核心模块，负责将用户层传递来的消息进行模块任务分发；
2. 典型应用模块：提供了SQL、Graph、Machine Learning以及Streaming等应用类型中的典型应用的算法或应用实现；
3. 数据生成器：针对不同应用类型，提供了各类应用的数据生成的脚本；
4. 组合参数发生器：提供了自动化的参数组合测试；
5. 系统监测器：在对应用进行可靠性测试时，通过获取待测系统的Metrics信息（如，内存、CPU、运行时间等）来监测待测系统的资源使用情况；
6. 报告生成器：通过用户的配置信息、应用运行得到的结果以及系统检测器收集到的资源使用情况，生成可靠性基准的测试报告，并展示给用户层。
   * 基础设施层

基础设施层提供数据存储系统以及待测系统执行平台。可靠性测试基准将逻辑实现层中数据生成器生成的数据集存储在数据存储系统（如HDFS等）中，并使用逻辑实现层中的组合参数发生器对待测系统提供可靠性测试。

* + 大数据系统及应用的可靠性测试基准平台的特色

（1） 构建自动化的大数据系统的可靠性测试基准平台。通过集成数据生成、数据存储、参数组合测试以及报告生成模块，将开发人员从算法学习、测试脚本编写以及接口学习等复杂繁琐的工作中抽离出来。

（2）该平台提供通用的逻辑方法及调度器，能够方便的提供不同待测系统的接口支持，有良好的扩展性。

（3）该平台使用带有特征分析的数据生成方法，提供了符合一定数据特征的更加丰富的数据集。

（4）该平台使用参数组合测试方法，并通过削减参数组合空间，极大的降低了测试成本。

（5）该平台提供了基于SSH的Web界面，用户只需提供待测系统的配置信息便可随时随地的对大数据系统进行可靠性测试。

（6）该平台是一种通用的可靠性测试基准框架，有良好的扩展性，可以针对不同的大数据系统提供可靠性测试支持。

## 系统实现

本节将介绍可靠性测试基准的系统实现，主要从以下几个方面展开：系统总体设计、典型应用模块、数据生成器、组合参数发生器、系统监测器、报告生成器等。

### 系统总体设计

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？，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。



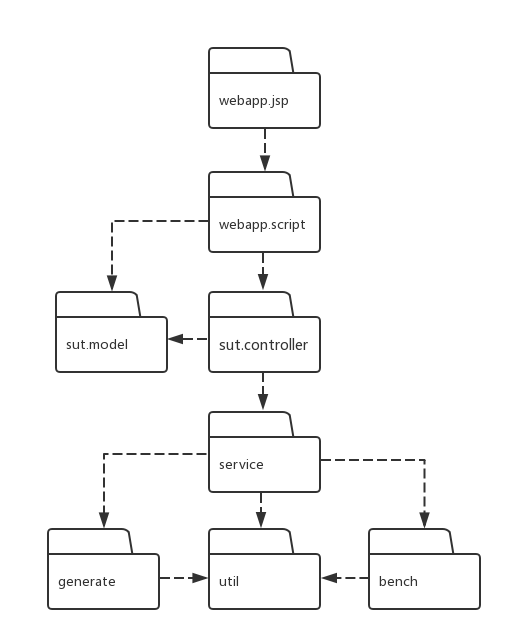
图表 16 MVC框架图

系统中的View负责展示用户层，即为用户进行可靠性测试提供界面支撑。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，并形成测试报告展示给用户。

系统中的Controller用于响应View层接收到的用户请求，来修改Model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。

* + 系统层次结构

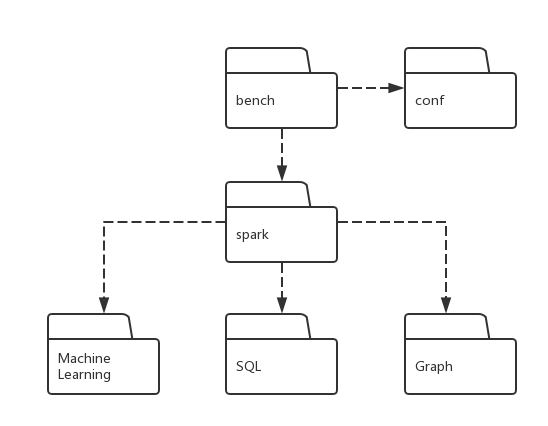
在MVC框架基础上，本系统进行分层实现。图？给出了系统实现中的包的依赖关系。其中上层包调用下层包，而下层包不依赖于上层包。下面对各个包进行了简要的介绍。



图表 17 系统包图

* webapp.jsp包提供了前端界面需要的JSP文件，用于为用户展示操作步骤中需要的配置信息、应用选择等信息，同时展示测试结束后的测试报告。
* webapp.script包提供了前端初步处理接收到的用户输入信息的JavaScript文件，用于从JSP界面中获取数据，并转换数据格式，传递给Controller。
* 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。
* sut.model包提供了web.script与sut.controller之间通信的数据定义规范。
* 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）。
* 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。
* generate包是数据生成需要的包，提供数据生成的脚本以及不同应用类型的数据生成类。
* bench包是基准测试的核心包，用于提供不同大数据系统的基准测试应用以及脚本。
  + bench包提供的服务

bench包提供了可靠性测试的测试应用及脚本，其中针对不同的大数据系统提供不同的基准实现。本系统暂时提供了对Spark的一部分应用的测试基准。图？是bench提供的包图架构。



图表 18 bench包图架构

* conf包是bench包下的配置包，用于提供各类配置文件。
* spark包以及flink包分别提供针对Spark系统以及Flink系统中各类应用的测试基准，包括应用类以及测试脚本。其中，Spark系统提供了Machine Learning、SQL以及Graph应用的测试基准。
  + 系统执行流程

用户可以通过Web访问可靠性测试基准平台。平台提供了Web界面，使用户可以更加便利的进行测试。图？给出了用户通过Web访问平台的活动图。

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）。这些都有webapp.jsp中的JSP提供，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。

然后，用户可以选择是否生成数据。（1）如果需要生成数据，需要填写配置信息（如应用类型等），然后点击“生成数据”按钮来生成数据。此时，前端会将请求通过Ajax传递给后台，进行命令发送，从而执行相应的数据生成脚本，生成自定义的数据集到数据存储介质（HDFS）中。（2）如果不需要生成数据，即当HDFS中已经有了相关数据时，可以直接进行测试。

接下来，进入是否执行测试的判断阶段。（1）如果需要执行测试，则需要配置测试的应用的系统参数和应用参数等信息。然后点击“执行测试”按钮来执行测试。此时，前端将请求传递给Controller，Controller通过service执行bench中的测试脚本。（2）如果不需要进行测试，整个活动结束。

最后，测试结束后的测试报告会通过JSP页面显示给用户，整个活动结束。



图表 19 活动图

### 数据生成器

测试基准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。本节将介绍概率分布中的部分算法实现以及不同应用的数据生成。

#### 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，本文提供的算法实现如下。

* + 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。

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

* + 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[]。

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

* + 指数分布

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：

。

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

* + 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：

|  |
| --- |
| 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。

* + 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？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。



图表 20 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？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。

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



图表 21 SQL数据生成脚本类图

* + 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[]。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[]。

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？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。



图表 22 Graph数据生成类图

* + Machine Learning

Machine Learning中的数据生成与以下参数相关，如表？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。

表格 8 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？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。



图表 23 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？所示。



图表 24参数组合测试

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. . |

* + 资源监测器

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。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周期性的轮询gmond收集到的信息，并存入RDD数据库。



图表 25 系统监测器

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。

* + 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。

## 可靠性测试应用验证

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



图表 26 实验环境图

如图？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？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。

表格 9 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？所示，这些参数主要决定了系统资源的分配情况。

表格 10 Spark系统配置列表

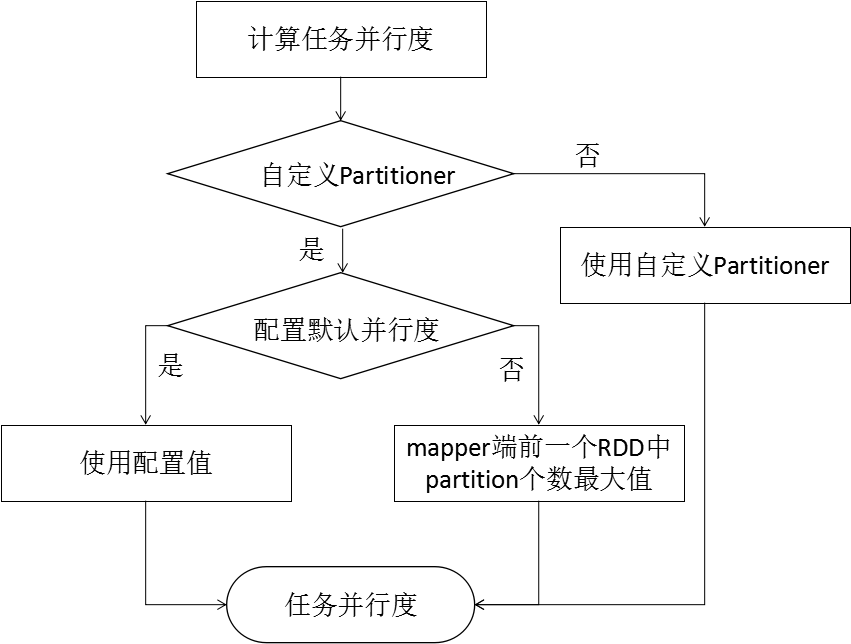
|  |  |
| --- | --- |
| 参数 | 含义 |
| Executor core | 分配给每个任务容器（JVM）的逻辑核数 |
| Total executor cores | 分配给任务容器（JVM）的总的逻辑核数 |
| Executor memory | 分配给每个任务容器（JVM）的内存大小 |

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？所示，应用参数如3.4.1节中表？所示。

表格 11 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？所示。



图表 27 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？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。

表格 12 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3.2.2节中设计的表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？给出了两种Join操作的测试结果。

表格 13 Join测试结果

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

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



图表 28 常规数据和倾斜数据对比

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？所示。



图表 29 第一次Join操作

（2）然后，继续使用表df1执行Join操作，如图？所示。



图表 30 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。

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。如下所示。

|  |
| --- |
| 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] |

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应用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3.4.1节中的表？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。

表格 14 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？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应用

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应用

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。

其主要原因是，在执行的应用构建的DAG很大，并且代码中需要进行太多shuffle的情况下，在Spark的lazy执行模式下，最后执行的动作就可能面临JVM无法承受计算压力。在这种情况下，就容易出现栈溢出的错误。

#### 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。

究其原因，主要是因为图计算需要用到迭代计算模型。以顶点为中心的迭代模型需要两个步骤：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及应用的可靠性测试基准框架。
2. 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的生成规则。同时，针对SQL、Graph以及Machine Learning等应用类型分别提出了异常数据（如分布异常、维度过高等特征）的生成方法。
3. 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。
4. 提出了针对流式大数据应用的高负载生成方法。该方法通过构建多个负载发生客户端，并通过并行划分数据流量、流速的方法，来生成高负载的应用数据。
5. 最后，论文介绍了大数据系统的可靠性测试基准的设计与实现，并将该可靠性测试基准框架应用在Spark和Flink大数据系统上。通过对Spark及Flink上的典型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），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：

* + 提供更多大数据系统中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。为了更好的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最后的测试报告，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.

发表文章

致谢