基于 Java 的 MADlib 自动化测试框架①

宋全德

(北京大学 软件与微电子学院, 北京 102600)

摘 要: 作为基于数据库管理系统 PostgreSQL 和 Greenplum 的数据分析软件, MADlib 提供了一系列数学函数及数据挖掘方法. 本文介绍了基于 Java 的 MADlib 自动化测试框架的设计与实现. 该框架可在 Linux 操作系统下自动实现数据的处理与导入, 测试用例的生成、执行, 以及测试结果分析. 从而可以避免手工测试带来的较大工作量. 同时, 基于 Java 的特点也使该框架便于开发与维护.

关键词: MADlib; Java; 数据库; 数据挖掘; 自动化测试

Java-Based MADlib Automation Test Framework

SONG Quan-De

(School of Software and Microelectronics, Peking University, Beijing 102600, China)

Abstract: As a data analysis software based on Postgre SQL and Greenplum, MADlib provides different kinds of mathematical, statistical and data mining methods. This paper introduces the design and implementation of java-based MADlib automation test framework, which can process and load data, generate and execute test cases, analyze test result automatically. As a result, large amounts of work from manual test can be avoided. What's more, this test framework is easy to develop and maintain because it is developed in Java.

Key words: MADlib; Java; database; data mining; automation test

随着社会和经济的发展, 对大数据进行运算、分析的需求日益增加. 作为一款基于数据库的数据分析软件, MADlib 提供了一系列基于 SQL 的统计分析算法与机器学习算法, 从而可以对存储在数据库中的大量数据进行计算分析与数据挖掘. 支持 MADlib 的数据库管理系统有 PostgreSQL 和 Greenplum. 其中, Greenplum 是分布式的数据库管理系统, 可以利用 MADlib 进行大数据的并行计算与分析. 另外, 鉴于 MADlib 更新较为频繁, 为保证新版本软件的功能, 质量以及与数据库管理系统的兼容性, 需要对安装在数据库中的 MADlib 进行功能测试与多平台测试. 由于 MADlib 的运行需要大规模数据的输入以及执行, 包括分布式数据库中的并行执行, 而简单的手工测试已无法满足测试的需求, 且工作繁杂, 会浪费大量时间, 因此有必要开发 MADlib 的自动化测试工具, 以提高软件测试的效率.

此外, 作为一种编程语言, Java 具有简单、成熟、稳定、平台无关等特性. 与此同时, 各种开源的工具包、集成开发环境、功能插件等也大大降低了开发与维护的复杂度. 所以可以选用 Java 来开发 MADlib 的自动化测试框架. Java 编程语言的优势可以使得对该框架的开发、调试、维护以及扩展更加方便与高效.

1 MADlib特点分析

和普通数据分析软件不同, MADlib 主要用于对数据库中存储的大量数据进行统计分析与数据挖掘. 同其他数据分析软件相比, MADlib 有以下几个特点.

第一, MADlib 提供的统计分析函数与数据挖掘函数均基于 SQL 语言, 调用这些功能函数, 尤其是数据挖掘函数时, 输入数据与输出结果均为数据库表.

第二, MADlib 主要用于分析和挖掘数据库中存储的大量数据. 因此调用 MADlib 函数时, 一般会输入大

① 收稿时间:2013-08-04;收到修改稿时间:2013-09-05

量的数据.

第三, MADlib 提供了一系列数据挖掘函数, 可以对输入的数据库中存储的数据进行机器学习与数据挖掘. 因为机器学习和数据挖掘的预测结果不可能与预期结果完全一致. 所以在需要衡量 MADlib 数据挖掘效果时, 可以计算预测结果的准确率, 以此来作为 MADlib 数据挖掘函数挖掘效果的衡量标准.

MADlib 的以上三个特点对 MADlib 自动化测试框架的设计具有重要影响. 在该测试框架中, 需要有相应的模块来实现转换数据格式、向数据库中导入数据、计算数据挖掘预测结果的准确率等功能.

2 系统原理和结构

考虑到被测软件 MADlib 的特点, 该自动化测试框架可以分为 5 个模块, 即: 数据处理与导入模块、测试用例生成模块、测试用例执行模块、测试结果分析模块、测试控制模块. 此外, 需要新建专门的配置文件以存储运行平台描述信息、测试用例描述信息、被测函数描述信息、数据集描述信息. 这些描述信息用于辅助数据的处理与导入、测试用例的生成以及测试用例的执行. 与此同时, 需要在数据库系统中新建两个数据库: 测试数据库与结果数据库. 测试数据库用于存储测试数据和执行测试用例; 结果数据库用于存储测试结果以及测试结果分析.

该自动化测试框架的结构图如图 1 所示.

各模块功能如下:

数据处理与导入模块用于处理和导入数据文件. 这些数据文件下载到计算机系统之后, 格式不一定符合要求. 此时数据处理与导入模块会对原始数据进行清洗与处理, 并将清洗和处理后的数据导入到测试数据库. 最终这些数据会以数据库表的形式存储于测试数据库中.

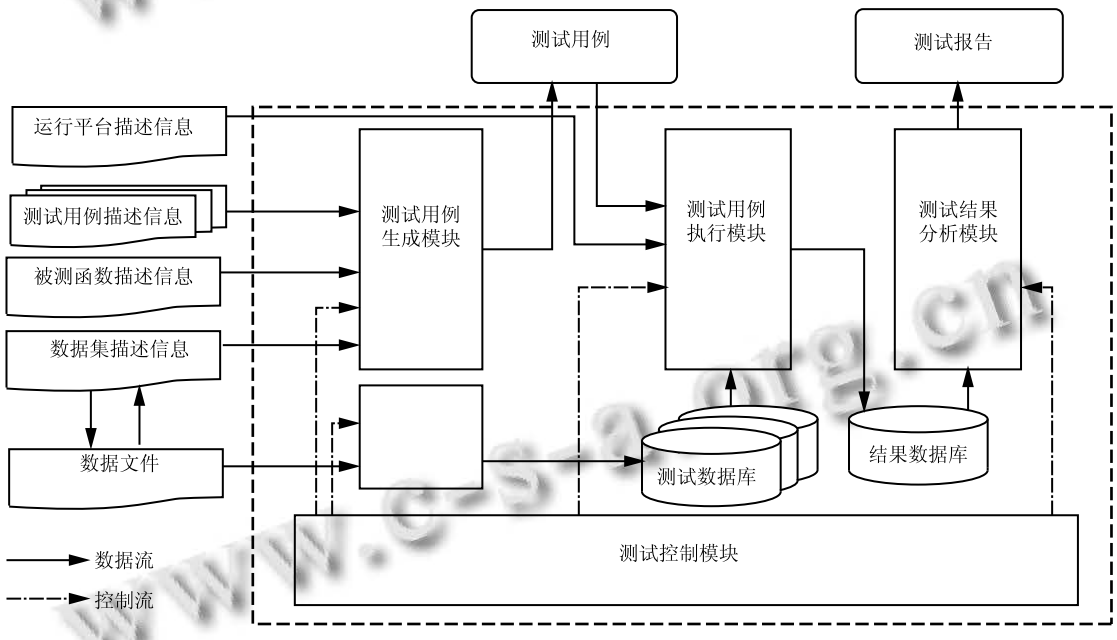

图 1 基于 Java 的 MADlib 自动化测试框架结构图

测试用例生成模块根据测试用例描述信息、被测函数描述信息、数据集描述信息生成测试用例. 每个测试用例对应一个文件, 文件中会有一组调用 MADlib 提供的统计分析函数或数据挖掘函数的语句.

测试用例执行模块根据运行平台描述信息, 在特定平台中执行测试用例. 具体过程是读取测试用例文件, 执行测试用例文件中的语句, 从而可以操作存储在测试数据库中的数据. 同时该模块会将测试结果存储在结果数据库中, 以供下一步进行测试结果分析.

测试结果分析模块会对存储在结果数据库中的测试结果数据进行分析. 该模块将计算 MADlib 提供的数据挖掘函数的预测准确率. 为了判断这些数据挖掘函数的预测准确率是否满足要求, 需要将 MADlib 提供的这些数据挖掘函数的预测准确率, 与第三方数据

挖掘工具(如 R、Weka. 本项目中采用的是 R)提供的相同函数对相同数据挖掘预测所得到的预测准确率进行对比. 该模块将根据第三方数据挖掘工具得到的预测结果来计算相应的预测准确率, 并且进一步计算和汇总 MADlib 与第三方数据挖掘工具各自提供的同名数据挖掘函数预测准确率的对比结果, 并以测试报告文件的形式输出. 此外, 该模块也会输出只记录测试结果的测试报告文件.

测试控制模块是控制整个测试流程的模块. 该模块会根据输入的参数来完成数据处理与导入、测试用例生成、测试用例执行、测试结果分析这几个功能, 并协调各功能模块之间的数据流与控制流. 测试控制模块是整个 MADlib 自动化测试框架的入口和控制中心.

以上 5 个模块, 按照图 1 所示的结构进行有效的配合, 从而可以实现对 MADlib 的自动化测试.

3 系统的设计和实现

3.1 系统的设计

3.1.1 项目结构设计

整个项目包括 Java 代码部分和辅助文件部分.

Java 代码部分是实现 MADlib 自动化测试功能的核心, 该部分包括 generator、executor、loader、resultanalysis、main、utility 共 6 个代码包. 其中, 代码包 loader、generator、executor、resultanalysis、main 分别对应系统结构图中的数据处理与导入模块、测试用例生成模块、测试用例执行模块、测试结果分析模块、测试控制模块. 代码包 utility 用于存放工具类代码.

辅助文件部分包括 Bootstrap、Dataset、Report、ResultAnalysis、Schedule、Testcase、Testspec 共 5 个文件夹. 这些文件夹中存有配置文件和模板文件等. 其中, 配置文件主要用于记录运行平台描述信息、测试用例描述信息、被测函数描述信息、数据集描述信息以及结果数据库描述信息等. Java 代码文件会读取或操作辅助文件部分所包含的配置文件、模板文件等. 辅助文件部分对实现 MADlib 自动化测试功能具有重要意义.

3.1.2 配置文件设计

本项目采用 XML 文件和 Yaml 文件作为配置文件的格式. 记录运行平台描述信息、测试用例描述信息、被测函数描述信息、数据集描述信息以及结果数据库描述信息的配置文件为 XML 文件. 记录数据集下载链接与处理数据集命令的配置文件和该测试框架运行信息的配置文件均为 Yaml 文件.

在这些配置文件中, Algorithmspec.xml 用于记录被测函数描述信息. 这些描述信息包括数据挖掘算法名, 被测函数名, 调用该函数的 SQL 语句模板等. Analyticstool.xml 用于记录运行平台描述信息. 运行平台描述信息包括数据库系统名、主机名、端口号等一系列信息. Dataset.xml 用于记录数据集描述信息. 数据集描述信息包括数据集名、数据集的行数以及需要用该数据集来测试的 MADlib 提供的功能函数的名字. Testconfig.xml 用于记录结果数据库描述信息. 这些描述信息包括数据库用户名、数据库所在主机名等. 记录测试用例描述信息的 XML 文件有多个, 每个数学统计算法或数据挖掘算法对应一组 XML 文件. 测试用例描述信息包括测试用例名、被测函数名、被测函数的参数名以及赋给该参数的一组参数值. 以上的一系列描述信息保存在 XML 文件中, 以便于自动化测试框架的维护, 同时也可以降低该测试框架的开发复杂度.

此外, Tables.yaml 用于记录待处理数据集名、数据集下载链接与处理数据集的 Unix 命令. Run.yaml 用于记录测试框架运行信息. 使用 Yaml 文件来记录这两部分信息, 可以充分发挥 Yaml 文件读取速度快的优势, 有利于提高自动化测试系统的性能.

3.1.3 系统模型设计

根据系统结构和代码模块的划分, 可以进一步设计各模块所包含的类. 数据处理与导入模块包含类 LoadingManager 和类 Load; 测试用例生成模块包含类 Datasets、GenCases、MultiTestSuite、ParaHandler、TestCase、TestSuite; 测试用例执行模块包含类 AlgorithmTemplate、Executor、ExecutorSpec、InputParameter、MethodTemplate、ReadSkipfiles、ReportGenerator、RunCase、TestCaseExecutor、TemplateExecutor; 测试结果分析模块包含类 ReportGenerator 和 ResultAnalysis; 测试控制模块包含类 Run.

该测试框架的模型图如 2 所示.

图 2 展示了各模块包含的类之间的调用关系和依赖关系. 该系统模型是进一步实现各模块功能的基础.

3.2 系统的实现

3.2.1 数据处理与导入模块的实现

该模块包含的类对应的 Java 文件均位于代码包

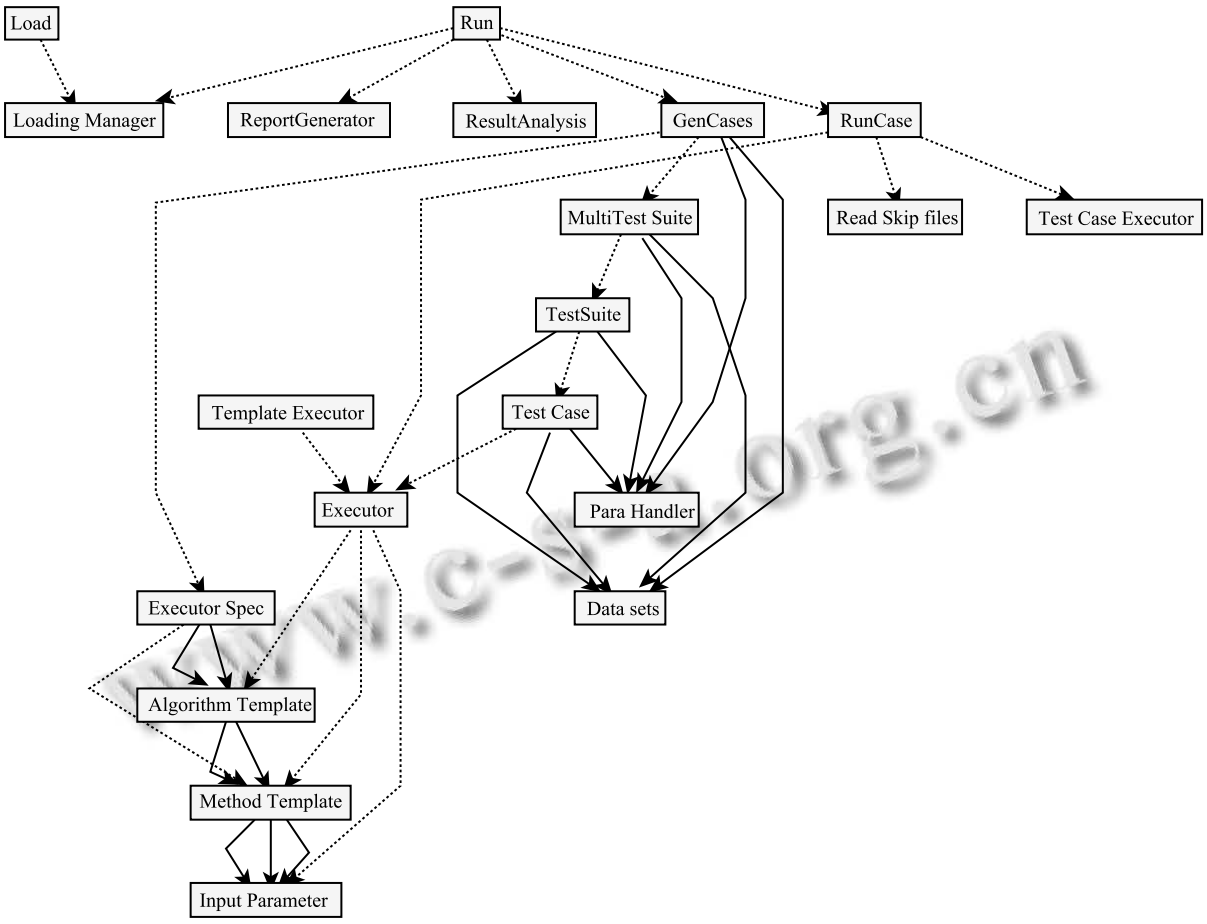

图 2 基于 Java 的 MADlib 自动化测试框架模型图

loader 中. 其中, 数据处理与导入功能主要由类 LoadingManager 实现. 在运行过程中, LoadingManager 需要读取配置文件 Tables.yaml, 并且调用工具类 AnalyticsTools 、 DbManager 、 PSQL. 工具类 AnalyticsTools 用于保存读取 Analyticstool.xml 所得到的运行平台描述信息, DbManager 用于开启、关闭、新建并初始化数据库以及提取数据库的相关信息, PSQL 用于获取数据库连接信息, 并且封装了运行 SQL 语句和获取 SQL 语句运行结果的方法.

除构造方法外, LoadingManager 还封装有以下方法:

void httpDownload(String httpUrl, String save FilePath);

void convert(ArrayList<String> list, Boolean over written);

void load(DbManager db_manager, String kind, ArrayList<String> list, boolean overload);

void doLoad(ArrayList<String> modules, Boolean overwritten,boolean overload, boolean initdb);

在这些方法中, doLoad 是该模块提供的对外接口, 供控制模块调用. httpDownload 是自动下载数据集的方法, 该方法通过调用 java 封装的库类 java.net.URL 和 java.net.URLConnection, 以及文件操作 API 来实现下载功能. convert 是转换数据格式的方法. 该方法通过调用 httpDownload 等方法下载数据集, 通过调用 org.yaml.snakeyaml.Yaml 提供的 Yaml 操作 API 来读取并解析 Yaml 文件, 得到一系列转换数据格式的 Unix 命令, 之后通过调用 java 封装的 Runtime.getRuntime(). exec 方法在命令行下执行这些转换数据格式的 Unix 命令, 将原始数据集转换为一组 SQL 文件. load 方法通过调用工具类 PSQL、Yaml 操作 API 以及 Runtime. getRuntime().exec 方法运行通过 convert 方法得到的一组 SQL 文件, 从而实现将数据导入数据库的功能. 对外接口 doLoad 调用 convert,load 等方法以实现完整的导入并处理数据功能. 类 Load 只含有 main 方法, 并且

main 方法中会调用工具类 AnalyticsTools 和同一模块内的 LoadingManager, 从而实现单独处理并导入数据的功能.

用以上方法实现的数据处理与导入模块, 能够有效处理数据并将数据成功导入数据库.

3.2.2 测试用例生成模块的实现

该模块的功能主要由代码包 generator 中的类来实现. 这些功能类所调用的工具类除了 Analytics Tools 之外, 还包括 Configer、Parser、Tools. Configer 类用于保存读取 Testconfig.xml 所得到的结果数据库信息. Parser 类封装了操作 XML 的方法, 用于读取和解析 XML 文件. Tools 类封装了一系列字符串操作, 如以特定分隔符连接多个字符串为一个字符串等. 由于该模块包含类较多且较为复杂, 现列出该模块的类图如图 3 所示.

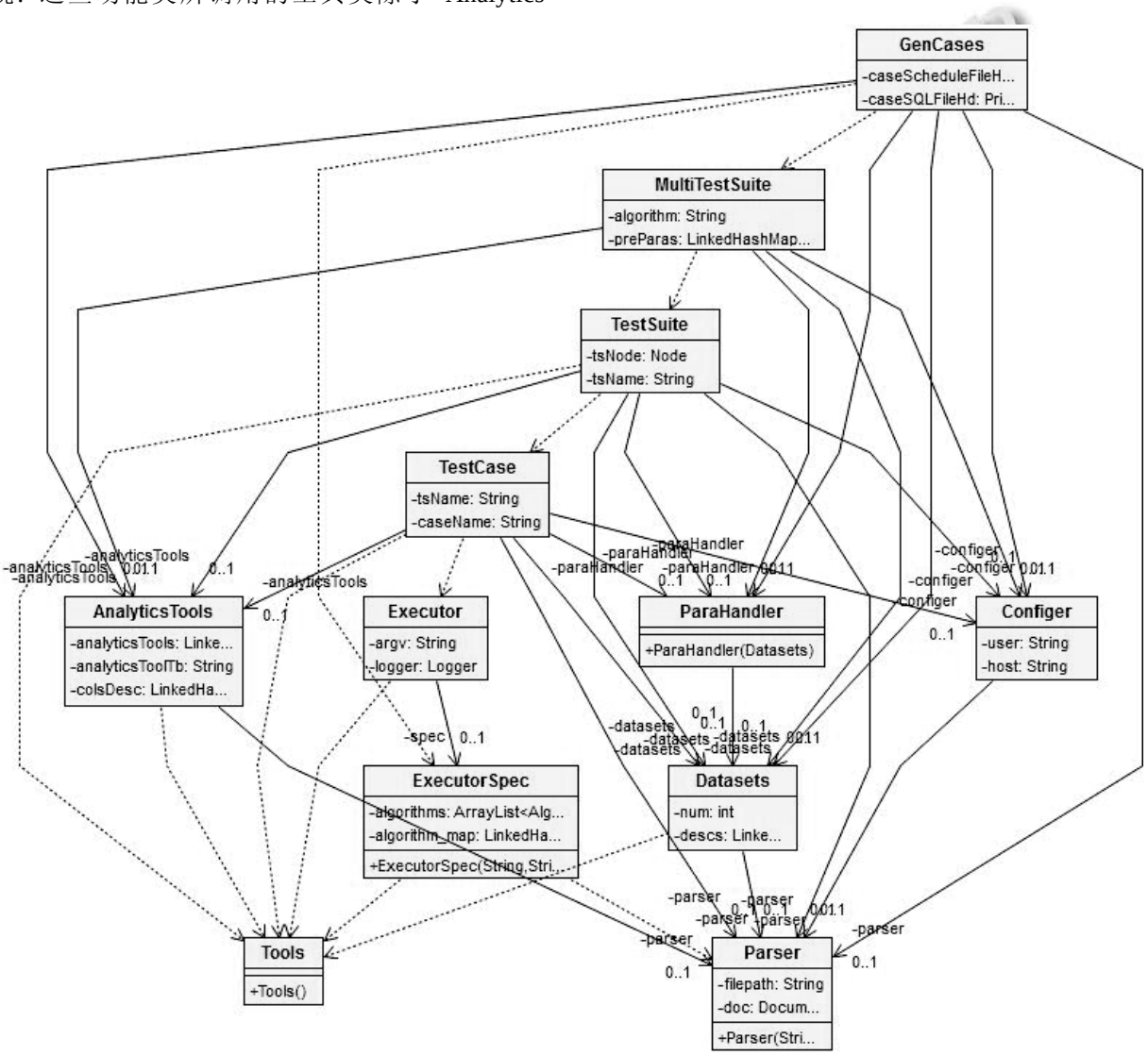

图 3　测试用例生成模块类图

代码包 generator 包含的类中, GenCases 提供了 main 函数, 从而使该模块既可以在命令行下单独被调用, 还可以在测试控制模块中被调用. Datasets 提供了解析配置文件 Dataset.xml 的 getDataSets 方法. getDataSets 方法通过调用工具类 Parser 将 Dataset.xml 中记录的数据集描述信息读到一个 LinkedHashMap 实例中并返回. ParaHandler 将 Dataset.xml 中记录的参数信息转换为命令行中的参数形式, 以便于后续的调用. 因此 ParaHandler 类会调用 Datasets 类. MultiTestSuite、TestSuite、TestCase 会根据记录测试用例描述信息的一

组配置文件, 生成一系列测试用例文件和 SQL 文件. 每个测试用例文件中写有执行该测试用例的一组 Unix 命令. 所有测试用例文件会存放在文件夹 Testcase 中. 这三个类的调用关系是 MultiTestSuite 调用 TestSuite, TestSuite 调用 TestCase. TestSuite 和 TestCase 通过调用 Java 文件操作 API 中的 PrintWriter 类来实现写文件的功能.

采用以上方法实现的测试用例生成模块既可以被单独调用, 从而达到只生成测试用例不执行测试用例的效果, 也可以被测试控制模块调用, 实现测试用例生成、执行一体化. 因而测试用例生成模块具有较高的灵活性.

3.2.3 测试用例执行模块的实现

测试用例执行模块的功能由代码包 executor 中的类实现. 这些功能类调用的工具类有 DbManager、PSQL、Parser、Tools、Logger、Path. Logger 类用于操作结果数据库. Path 类封装了一系列文件路径作为 public 属性, 以便于对相关文件的调用.

代码包 executor 中的类 RunCase 封装的 runCases 方法可作为测试用例执行模块提供的对外接口. 该方法定义如下:

String runCases(String getfile,String skipfile, Boolean isList, boolean isUnique, String platform, AnalyticsTools analyticsTool, Configer testConfiger, int run_id);

参数 getfile 是记录需要执行的测试用例文件名的 schedule 文件的路径; skipfile 是记录需要跳过不执行的测试用例文件名的 skip 文件的路径; isList 用于标志 Run.yaml 中列出的是 schedule 文件还是 list 文件(list 文件中记录多个 schedule 文件名, schedule 文件中记录 1 个或多个测试用例文件名); isUnique 用于标志是否执行 schedule 文件中出现过多次但已经执行过的测试用例文件; platform 是执行测试用例的平台名; analyticsTool 是工具类 AnalyticsTools 的实例, 用于记录运行平台描述信息; testConfiger 是工具类 Configer 的实例, 用于记录结果数据库信息; run_id 用于标记本次测试用例执行. 函数 runCases 会返回执行测试用例的 MADlib 版本信息.

此外, AlgorithmTemplate 类用于存储被测函数描述信息, 该类通过调用工具类 Parser 读取 Algorithmspec.xml 文件来实现其功能. 与 Algorithm Template 一样, MethodTemplate 类和 InputParameter 类也通过调用工具类 Parser 实现解析和提取被测函数及参数的功能. ReadSkipfiles 类用于读取 skip 文件. ExecutorSpec 用于存储测试用例执行的相关信息. Executor 和 TestCaseExecutor 用于封装执行测试用例的底层操作, 如将测试结果存入测试数据库等. 二者通过调用工具类 DbManager、PSQL、ArgumentParser、MapFormat、Timer 等来实现这些底层功能. 在这些工具类中, ArgumentParser 用于解析命令行参数, MapForm 封装了 Map 的相关操作, Timer 封装了与时间有关的操作. 具体实现策略是调用 Java 文件操作 API、Java 封装的命令行操作方法 Runtime. getRuntime().exec 以及工具类 PSQL 以实现这些功能. TemplateExecutor 封装了 main 方法, 可以作为作为测试用例执行模块被单独调用时的入口.

RunCase 会调用本模块内的 Executor、TestCase Executor 等类及相关方法, TemplateExecutor 会调用本模块内的 Executor 类及相关方法. 二者均实现了自动执行测试用例的功能.

3.2.4 测试结果分析模块的实现

测试结果分析模块的功能由 ReportGenerator 类和 ResultAnalysis 类实现. 这些类对应的 Java 文件在代码包 resultanalysis 中. ReportGenerator 类用于读取结果数据库中的测试结果, 并将这些结果写入测试结果文件中. ResultAnalysis 类用于分析测试结果, 并将分析结果写入测试结果分析文件中. 测试报告包括测试结果文件和测试结果分析文件.

ReportGenerator 通过调用 Java 文件操作 API 和工具类 PSQL 以实现生成测试结果文件的功能. ResultAnalysis 提供了对外接口函数 generateResult AnalysisReprot 和 main 函数. main 函数使该模块可以脱离于测试控制模块而单独被调用, 对外接口函数 generateResultAnalysisReprot 可供测试控制模块调用从而实现测试用例生成、测试用例执行、测试结果分析一体化的效果. 除 main 函数和构造函数以外, ResultAnalysis 类中的其他方法有 getTestResult、getExpectedResult、getTestResultFromR、getCaseList、getCaseReport 和 generateResultAnalysisReprot.

在这些方法中, getTestResult 用于获取测试结果, getExpectedResult 用于获取预期结果, getTestResult FromR 用于获取 R 的相同方法执行相同测试数据所得

到的结果. 由于测试结果、预期结果、R 的相同方法执行相同测试数据所得到的结果都以数据库表的形式存储在结果数据库中, 因此以上三种方法通过调用工具类 PSQL 来操作数据库, 并通过调用 Java 封装的字符串操作 API 来处理得到的结果数据信息. getCaseList 用于获取所有测试用例名, 也是通过调用工具类 PSQL 操作结果数据库来得到测试用例名称信息. getCaseReport 会调用 getTestResult、getExpectedResult、getTestResultFromR, 并且根据标记本次测试用例运行的 id, 指定的测试用例名称和相应数据库表模式名称来计算该测试用例在 MADlib 上执行结果的准确率和相同测试用例数据在 R 上执行结果的准确率, 进而对所得数据进行汇总并将汇总得到的数据信息返回. 对外接口 generateResultAnalysisReprot 会调用 getCaseList 方法和 getCaseReport 方法, 并且通过调用工具类 Path 和 Java 文件操作 API 来生成测试结果分析报告. 测试结果分析报告将以文件的形式保存在辅助文件部分的 ResultAnalysis 文件夹中.

采用以上方法实现的测试结果分析模块可以被单独调用, 也可以被测试控制模块调用, 因而同测试用例生成模块一样, 具有较高的灵活性.

3.2.5 测试控制模块的实现

测试控制模块的功能由 Run 类实现. 该类对应的 Java 文件位于代码包 main 中. Run 类是测试控制模块的核心, 用于控制整个 MADlib 自动化测试框架. Run 类中包含 getFileExtensionFilter 方法和 main 方法. getFileExtensionFilter 方法用于处理文件名后缀, 并且通过调用 java.io.FilenameFilter 来实现此功能. getFileExtensionFilter 方法会在 main 方法中被调用. 该类中的 main 方法是整个自动化测试框架的基本入口. 此方法会调用数据处理与导入模块、测试用例生成模块、测试用例执行模块、测试结果分析模块这 4 个模块的对外接口, 同时会调用 ArgumentParser、PSQL、Configer 等工具类. 运行时, main 方法会读取命令行参数. 不同的参数将使 main 方法调用不同的功能模块并采取不同的操作. Run 类中的 main 方法通过调用工具类 ArgumentParser 来分析并提取命令行参数.

以上方法充分实现了测试控制模块的功能. 作为整个 MADlib 自动化测试框架的核心, 测试控制模块可以有效整合并控制其他功能模块, 从而达到数据处理与导入、测试用例生成、测试用例执行、测试结果分析 4 个功能自动化、一体化的效果.

4 实现效果

该自动化测试框架开发完成后, 需要在 Linux 或 Unix 系统中运行. 可以利用该自动化测试框架单独完成数据处理与导入操作、测试用例生成操作、测试用例执行操作、测试结果分析操作, 也可以把这些操作整合到一起从而实现整个测试过程的一体化和自动化. 通过调用测试控制模块的入口函数来完整运行自动化测试时, 会自动生成并保存测试报告文件. 以测试并分析数据挖掘方法 AR 为例, 该测试框架的运行效果部分如图 4 所示.

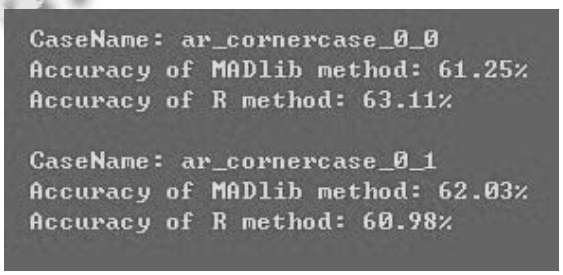

图 4 基于 Java 的 MADlib 自动化测试框架运行效果图

图 4 显示的内容会保存在测试结果分析文件 MADlibtestresult_runid_1__analysisreport 中.

作为 MADlib 自动化测试框架, 该系统可以实现自动化测试功能, 并且操作较为方便, 同时又便于进行功能测试和多平台测试, 达到了预期目的.

5 总结与展望

本文详细论述了 MADlib 自动化测试框架的设计和实现过程. 作为自动化测试工具, 该测试框架实现了从数据处理与导入到测试结果分析整个测试流程的自动化, 同时又具有较高的可操作性. MADlib 自动化测试框架不仅可以用于 MADlib 的功能测试, 还可以用于 MADlib 的多平台测试. 当 MADlib 版本更新频繁并需要进行多平台测试时, 该自动化测试框架的优势就会更加明显.

当然, 该自动化测试框架在某些方面仍然需要改进和提升. 后续的改进和功能扩展工作中主要有三个关键任务: (1)集成第三方数据分析工具 R 的 Java API, 使该测试框架可以自动运行 R, 并且进一步分析和存储 R 的分析结果; (2)集成代码覆盖率分析工具的 Java

API, 在测试结果分析模块中加入代码覆盖率分析功能; (3)在测试结果分析模块中加入对测试用例执行时间的分析和比较, 从而便于进行性能测试.

综上所述, 随着 Java API 的进一步升级和扩展, 以及后续改进工作的不断实施, 该自动化测试框架的自动化程度、可扩展性、可维护性和复用性都将得到更大的提升. 该自动化测试框架虽然针对的是 MADlib, 但是其设计思路和实现策略对其他数据分析工具的自动化测试以及自动化测试工具的设计与开发同样具有一定的意义.

参考文献

1 马瑞芳,王会燃.计算机软件测试方法的研究.小型微型计算机系统,2003,24(12):2210–2213.

2 朱菊,王志坚,杨雪.基于数据驱动的软件自动化测试框架.计算机技术与发展,2006,16(5):68–70.

3 张靖.XML 技术在软件可靠性测试中的应用.电子科技大学学报,2007,36(4):767–770.

4 Hellerstein JM, Ré C, Schoppmann F, Wang DZ, Fratkin E, Gorajek A, Ng KS, Welton C, Feng X, Li K, Kumar A. The MADlib analytics library: or MAD skills, the SQL. Proc. of the VLDB Endowment, 2012, 5(12): 1700–1711.

5 Christof JB, Kwong CW, Kapfhammer GM. Bridging the Gap between the theory and practice of software test automation. 32nd ACM/IEEE International Conference on Software Engineering, ICSE 2010. Cape Town, South Africa. 2010. Piscataway, IEEE Computer Society. 2010. 445–446.

6 Gupta P, Surve P. Model based approach to assist test case creation, execution, and maintenance for test automation. The First International Workshop on End-to-End Test Script Engineering. Toronto, Ontario, Canada. 2011. New York, ACM. 2011. 1–7.

7 Ali MM, Saha TK. A proposed framework for full automation of software testing process. 2012 International Conference on Informatics, Electronics and Vision, ICIEV 2012, Dhaka, Bangladesh, 2012. Washington, DC, IEEE Computer Society. 2012. 436–440.

(上接第 16 页)

Trans Tech Publications LTD. 2011, 50-51: 323–327.

48 刘晓佳.基于复杂网络可视化模型的专题新闻演化研究.中国管理信息化,2012,15(24):77–78.

49 Bateman A, Coin L, Durbin R, et al. The Pfam protein families database. Nucleic Acids Research, 2004, 32(suppl 1): D138–D141.

50 Kerrien S, Alam-Faruque Y, Aranda B, et al. IntAct-open source resource for molecular interaction data. Nucleic Acids Research, 2007, 35(suppl 1): D561–D565.

51 Keseler I M, Collado-Vides J, Gama-Castro S, et al. EcoCyc: a comprehensive database resource for Escherichia coli. Nucleic Acids Research, 2005, 33(suppl 1): D334–D337.

52 Baolin L, Bo H. HPRD: a high performance RDF database. IFIP International Conference on Network and Parallel Computing. Berlin, Trans Tech Publications LTD. 2007, 4672: 364–374.

53 杨利军,魏晓峰.基于知识图谱的国外社会网络分析领域可视化研究.情报科学,2011,29(7):1041–1048.

54 陈仕吉.科学研究前沿探测方法综述.现代图书情报技术,2009,9:28–33.

55 栾春娟,侯海燕,王贤文.国际科技政策研究热点与前沿的可视化分析.科学学研究,2009,127(2):240–243.