

Tiny Design Specification

|  |  |  |  |
| --- | --- | --- | --- |
| Project/Feature Stakeholders | | | |
| Stakeholder Name | Representing | Stakeholder Name | Representing |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

This document is meant to facilitate the design review by all stakeholders. Once the design is finalized, this document will guide documentation and test breakout. The document is a “live” document, meaning that additions are being made as the implementation progresses; the spec owner is responsible for keeping the document up-to-date. To reflect the modifications being made, make sure you are using the Change Tracking feature. Do not accept or reject comments—simply make sure they get highlighted from version to version.

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Date [and Time] | Reason For Changes | Version | Status |
|  |  |  |  |  |
|  |  |  |  |  |

General Template Notes:

1. Use the File:Properties:Subject field for the name of the project.

2. Use the File:Properties:Author field for the author.

3. Use the File:Properties:Keywords field for the revision of this document. Make sure that you update this for each revision you make. (Automating this is not useful.)

4. Make sure that you update all fields before saving this document, and be particularly careful about updating fields before you print the document. MS Word is quite inconsistent about when and how fields are updated. Do it by hand, and don’t forget to do the headers. There are three different header sections in this template. The second and third ones use the File:Properties:Subject field.

5. Please remove all template comments (gray font inside boxes) before you print or save your document.

Table of Contents

1. Motivation 4

2. Requirements 4

2.1 Functional Requirements 4

2.2 Scalability/Performance Requirements 4

2.3 Security Requirements 4

2.4 Rejected requirements 4

3. Component Overview 4

3.1 Nomenclature 5

3.2 Architectural Overview 5

*4.* Component Design 7

4.1 Test Suite and Test Case Controller 7

4.2 Test Case Generator 7

4.3 Test Case Executor 7

4.4 Test Case Analyst 7

4.5 Data Tool 7

5. Use cases 7

6. Additional considerations 7

7. Bibliography 7

# Motivation

# Requirements

The test framework should be able to automatically generate the test cases based on the PL function’s configuration and test data spec, and provide a way for the end user to query and analyze the test result.

## Functional Requirements

Requirements that describe the functionality of the design element.

| **ID** | **Requirement synopsis** | **Rationale/comments** |
| --- | --- | --- |
|  |  |  |

## Scalability/Performance Requirements

Requirements that describe desired/necessary scalability and performance.

| **ID** | **Requirement synopsis** | **Rationale/comments** |
| --- | --- | --- |
|  |  |  |

## Security Requirements

Requirements that describe desired/necessary scalability and performance.

| **ID** | **Requirement synopsis** | **Rationale/comments** |
| --- | --- | --- |
|  |  |  |
|  |  |  |

## Rejected requirements

Requirements that were considered but rejected for the reasons listed.

| **ID** | **Requirement synopsis** | **Rationale/comments** |
| --- | --- | --- |
|  |  |  |

# Component Overview

Synopsis of the context in which the feature/component is to be implemented. Abstract description of the component, e.g. class diagrams, workflow and interaction diagrams, etc. This includes design choices and discussion of alternatives as necessary.

## Nomenclature

List all terms that need clarification in order to understand the design.

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Usage in context of feature** |
|  |  |  |

## Architectural Overview

Figure 1 is the architecture of the MADlib benchmark framework, which could be called MADMARK for short.

First, the test case generator will generate the test cases according to the test case spec XML and analytic tool configuration file, and store the test cases that will run later. The users can skip this generation step to run the test cases that are already generated directly.

Secondly, Test Suite and Test Case Controller will drive data tool to load data set. Each kinds of executor need different format of data set, such that database table is for MADlib, OS file is for R, and HDFS file is for Mahout. The users can skip this data set loading step if the data set is already available.

Thirdly, Test Suite and Test Case Controller will drive Test Case Executor to run the test cases, and store the running result to database.

Finally, the users can trigger Test Case Analyst to analyze the running result, such as the performance results, the comparisons among each kinds of executor’s running results, and so on.

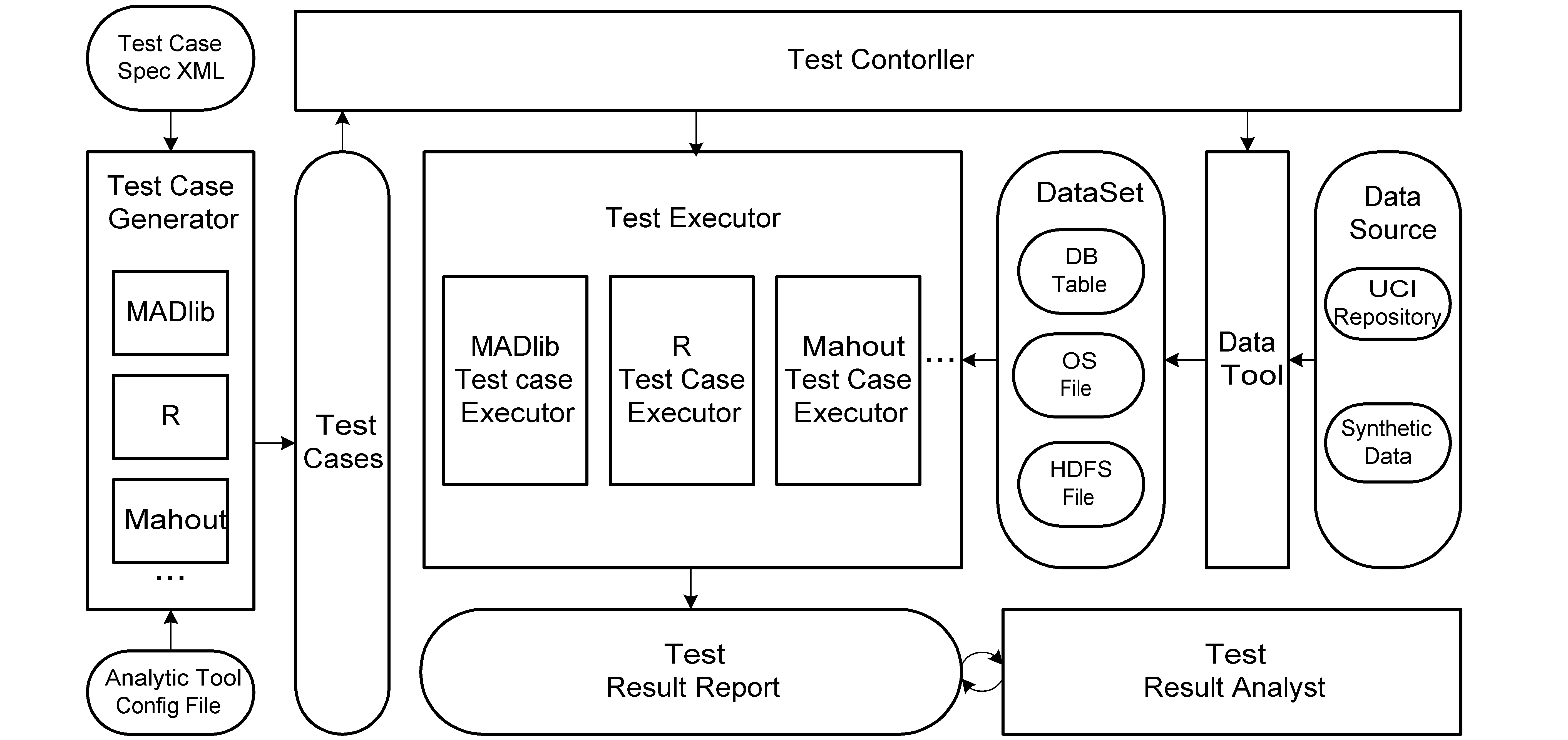


Figure 1: architecture of the MADlib benchmark framework

The MADlib benchmark test framework consists of five main components, as you can see from Figure 1:

* **Test Case Generator:**

It will generate the test cases according to the test case spec XML and the test case configuration file.

* **Test Suit and Test Case Controller:**

It will drive the Test Case Executor to run the test cases with the specified data set. And it can also drive the data tool to load data set.

* **Test Case Executor:**

It will execute the test cases, and then store the results to database. There are different kinds of Test Case Executors, such as MADlib, R, and Mahout.

* **Test Case Result Analyst**

It will analyze the test case results, such as the performance results, comparisons of different executors’ results, and then store the analytic result.

* **Data Tool**

It will load data set from different data sources, such as UCI repository or this Benchmark framework prepared synthetic data. The data set formats are different among each kinds of executor.

The Benchmark framework consists of four main storages:

* **Test Data Spec in XML**
* **Test Cases**
* **Data Set**
* **Data Source**

# Component Design

## Test Case Organization

The test case is generated from xml spec files. For each xml spec file, there will be multiple test suites with distinct test suite name in all xml spec file. In each test suite, the user can not specify any test case details, but only write the input parameters and the invoking method name. The test case generator will generate each test case automatically.

The xml test spec file is like this:

<testsuits>

<testtype>feature</testtype>

<multitestsuits>

<algorithm>linear\_regression</algorithm>

<methods>

<method>

<name>linregr</name>

<parameter>

<name>benchmark</name>

<value> TemplateExecutor </value>

</parameter>

</method>

</methods>

<testsuit>

<name>linear\_regression\_functionalities</name>

<comments>It is to test basic functionalities of linear regression.</comments>

<!-- It indicate max execution times-->

<execiterate>1</execiterate>

<method>

<name>linregr</name>

<varparameter>

<name>source\_table</name>

<value>MADLIBTESTDATA.lin\_Concrete\_oi,MADLIBTESTDATA.lin\_Concrete\_wi </value>

</varparameter>

<parameters>

<parameter>

<name>dependent\_varname</name>

<value>y</value>

</parameter>

<parameter>

<name>independent\_varname</name>

<value>x</value>

</parameter>

<parameter>

<name>analyticstool</name>

<value>GPDB\_16Segments</value>

</parameter>

</parameters>

</method>

</testsuit>

In each xml spec file, there are multiple <multitestsuite>. And In each <multitestsuite>, there are multiple <testsuit>. And in each <testsuit>, there are multiple methods. A methods is the execution unit.

For each algorithm, there could be multiple methods. According to kmeans, there are three kinds of methods, such as canopy, prepared centroids and prepared number of centroids(in this case, the init\_methd could be random or kmeans++, and user must input number of centroids for clustering). According to decision tree, there are five kinds of methods, such as c45\_train, c45\_score, c45\_display, c45\_classify and c45\_clean. For UCI data, madmark use TemplateExecutor as the executor.

Madmark classes input parameters into the predefined parameters, the vary parameters and the fixed parameters. The predefined parameters are all the same during the whole <multitestsuit>, and could not be specified in the <testsuit> tag. In the <method> tag, user can specify multiply vary parameters with different values. Suppose there are three vary parameters p1, p2 and p3, which have m1, m2 and m3 different values, madmark will generate m1 \* m2 \* m3 pairs of value of <p1, p2, p3>. In this way, madmark could generate many test cases with few xml spec files. The fixed parameters are specified in <method>’s <parameter> tag.

## Test Case Generator

## Test Case Executor

## Test Case Analyst

## Data Tool

# Use cases

# Additional considerations

Additional aspects relevant to the discussion of this design if it has not been part of the core concerns of the design yet: performance, usability, etc.

# Bibliography

List of relevant documents.