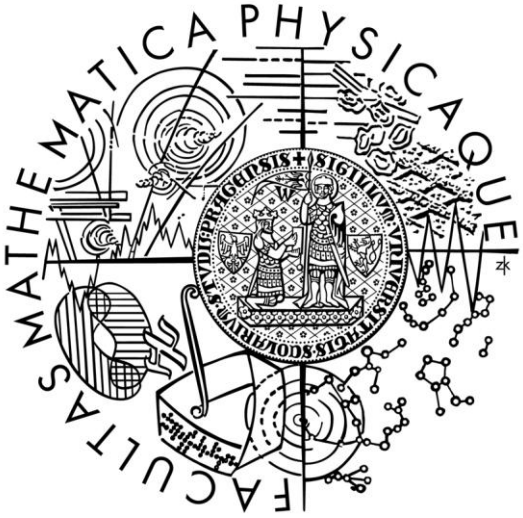
Charles University in Prague,
Faculty of Mathematics and Physics

BACHELOR THESIS



David Babka

Regression Benchmarking Web

Department of Software Engineering

Supervisor: Doc. Ing. Petr Tůma, Dr.

Study program: Computer science, Programming

2008

I would like to thank my supervisor Doc. Ing. Petr Tůma, Dr., for numerous pieces of advice, corrections, his patience and the time he spent with me. Furthermore, I would like to thank my former supervisor RNDr. Tomáš Kalibera, PhD., for guiding me through the specification and Mgr. Vlastimil Babka for his review.

I hereby declare that I wrote the thesis myself using only the referenced sources. I agree with lending the thesis.

Prague, August 6, 2008 David Babka

Contents

Název práce: *Web pro prezentaci výsledků regresivních výkonnostních testů*
Autor: *David Babka*
Katedra: *Katedra softwarového inženýrství*
Vedoucí bakalářské práce: *Doc. Ing. Petr Tůma, Dr.*
e-mail vedoucího: petr.tuma@mff.cuni.cz

Abstrakt: Při vývoji software je jeho výkon často důležitou součástí požadavků zákazníka. Vzhledem k tomu, že se software neustále vyvíjí, lze pomocí porovnávání výsledků testů výkonu určit nedostatky ve zdrojovém kódu a tím celkově optimalizovat výkon software. Tato práce se zabývá implementací flexibilní webové aplikace založené na získávání těchto výsledků a jejich prezentací je v grafech a jiných formách. Pro co nejlepší přehlednost bude možné tyto grafy dynamicky generovat. Další z výhod této aplikace bude možnost zasílání e-mailů uživatelům s informacemi o nových měřeních.

Klíčová slova: webová aplikace, výkonnostní testování, regrese, zátěžové testování, prezentace výsledků, grafy

Title: *Regression Benchmarking Web*
Author: *David Babka*
Department: *Department of Software Engineering*
Supervisor: *Doc. Ing. Petr Tůma, Dr.*
Supervisor's e-mail address: petr.tuma@mff.cuni.cz

Abstract: During the software development the performance of the software is often important part of client requirements. Considering the constant development process of the software, comparison of the performance testing results could pinpoint to various defects in the source code, which could lead to optimizing software and its performance. This thesis focuses on implementation of flexible web application created for retrieving the performance results and presenting them in plots and other forms. The plots are dynamically generated to provide as much transparency as possible. This application will also provide the ability to send e-mails to users with the information about new measurement results.

Keywords: web application, performance testing, regression, benchmarking, result presentation, plots

1 Introduction

1.1 Regression benchmarking

It is generally accepted [1] that performance testing has an essential place as an integral part of the software development process - with the advent of distributed software development centered around web accessible frameworks such as development forges, the need for publishing of performance testing results on the web increases.

However, performance testing results can be affected by various random factors, which makes the results quite distorted [16]. For example software compilation does not always create identical binaries and therefore testing different compilations could lead to different results. It is vital for the results to display the most representative value. Performance tests are therefore usually repeated many times including recompilation, creating a huge amount of data. Statistical analysis then provides trustworthy results such as mean or 99% confidence interval, which can be published on the web.

Regression benchmarking allows capturing performance changes of software development. Multiple versions of the software are separately measured and the obtained results are compared to show the performance impact of code changes between them, which could pinpoint to effective or ineffective code for further improvements. One of the basic visual outputs of regression performance testing is a performance impact plot, such as the one displayed on Figure 1.1.

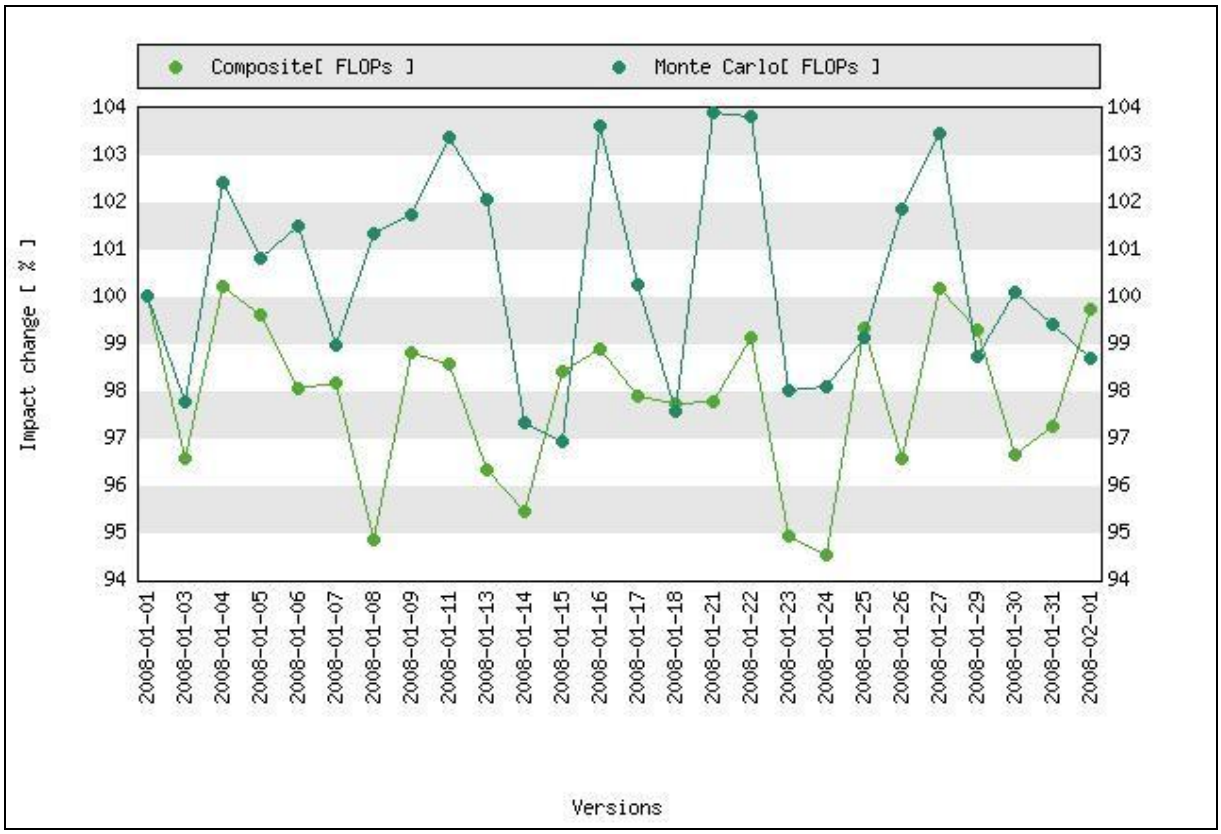


Figure 1.1 - Performance impact plot

1.2 Goal of the thesis

This thesis focuses on the part of the regression benchmarking framework that is responsible for displaying results. The goal is to implement a flexible web interface for displaying benchmarking results that can be used to track performance of software as it develops.

This implementation of the regression benchmarking web will be used as a replacement for the existing MONO Regression Benchmarking web [2], which no longer fulfills required expectations such as dynamic plot generating, and lacks flexibility.

The regression benchmarking web will be based on an open source content management platform and the required features will be provided by plug-in modules created for this platform, thus making the web more versatile, which is essential for further improvements and extensions.

After correct installation of these modules, the web application will be able to parse benchmarking results and present them in dynamically generated plots. There will be also the possibility to send e-mail notifications about newly submitted results to registered users.

1.3 Overview of the following document

The second chapter presents an analysis of detailed requirements and a discussion of possible approaches to fulfill these requirements. The third chapter describes how to use the implementation from the installation to result parsing or plot displaying. Detailed description of the web implementation follows in the fourth chapter and the fifth chapter concludes the thesis.

2 Analysis

2.1 Detailed requirements

There are four primary tasks essential for the regression benchmarking web.

Projects, benchmarks and platforms management

Unlike MONO Regression Benchmarking [2], which is specialized only for performance testing of MONO [20], the regression benchmarking web will have the ability to manage multiple software projects and with it comes the need to manage individual benchmarks[1] and platforms[2].

The management of these three types will have few aspects in common. They will all hold the name, description, publisher and the published tag. The rest of the settings will be specialized for each type. In the project management, the management of dependencies is essential; it will hold the information about benchmarks and platforms used for the project performance testing. Benchmark management will focus on the management of its metrics.

Performance testing results parsing

The benchmark results are usually stored in files in human-readable ASCII-based[3] format and for further presentation it is vital to extract the results from these files. After the extraction, the most necessary data are stored to the database for quicker and more comfortable access. Detailed discussion about parsing can be found in following section 2.2.

Performance testing results presentation

The next major improvement from MONO Regression Benchmarking [2] is dynamic plot generation, which makes the presentation more transparent. Plots will be created and parameterized by plot wizards and have automatically justified axes to fit all displayed results and to prevent blank spaces.

[1] **Benchmark** is a computer program or sets of programs designed to test the software performance
[2] **Platform** describes hardware architecture or software framework used for the benchmarking
[3] American Standard Code for Information Interchange (**ASCII**) – character encoding

The regression benchmarking web will provide following performance testing presentations:

Performance results plot

The performance results plot (Figure 2.1) is composed of multiple vertical segments, each segment belonging to results from one version of the tested software (versions are denoted by dates on which they were committed to the source control repository). Within each segment, the X axis shows the sequential number of the collected sample in a benchmark run that collects a sequence of samples, the Y axis the value of the sample (a single grey dot is displayed for each sample). Segments also contain mean (red horizontal stripes) and 99% confidence interval (blue horizontal stripes) in the current vertical position related to Y axis.

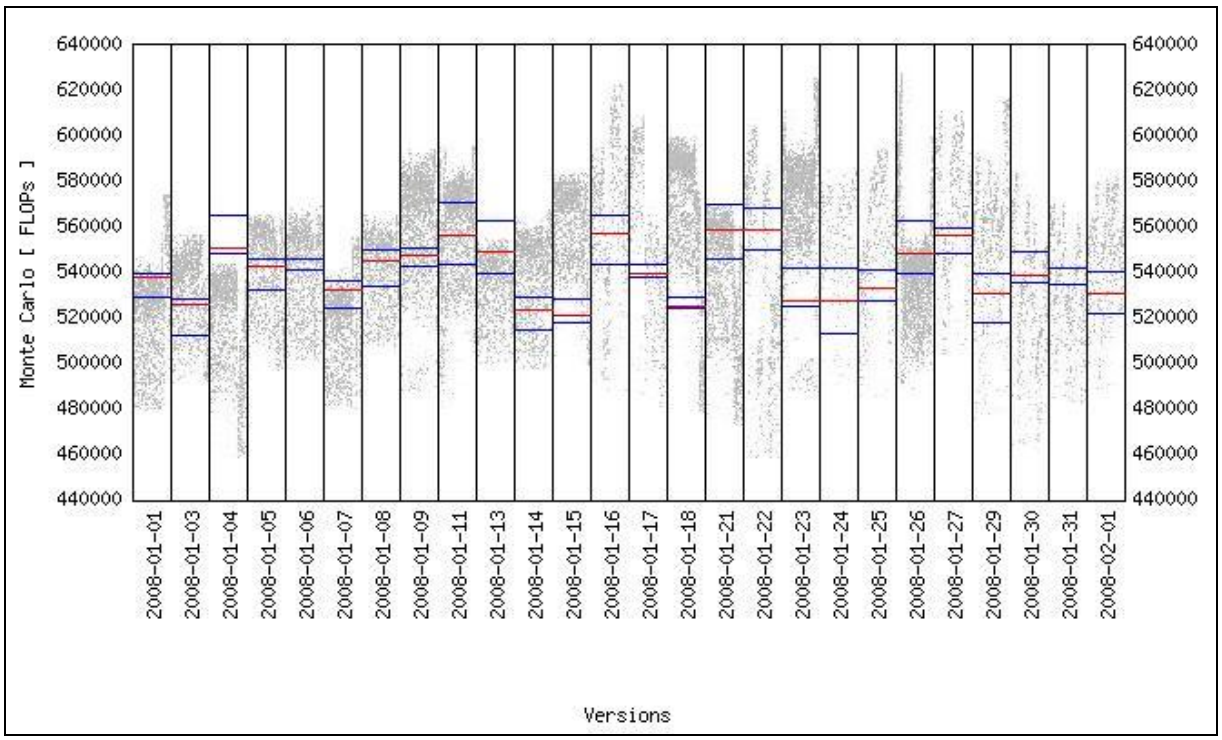


Figure 2.1 - Performance results plot

99% Confidence plot

This plot displays both mean and confidence interval (Figure 2.2). On the X axis there are displayed versions of the tested software by the date of commit to the source control repository. The Y axis shows the value of the mean (red circles) and 99% confidence interval boundaries (blue circles connected by blue line) of all the samples collected from a single version. The mean circles are connected to emphasize the regression of the software versions.

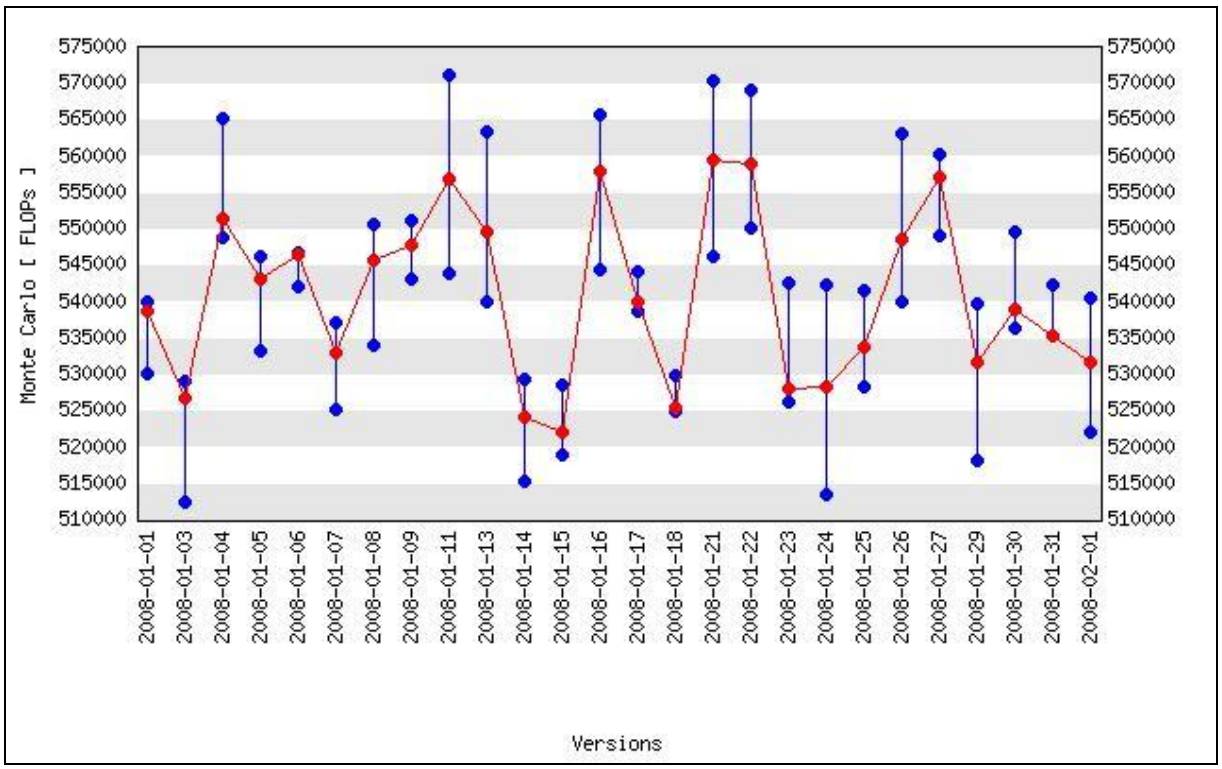


Figure 2.2 – 99% Confidence plot

Impact plot

The impact plot (Figure 1.1) shows performance changes over multiple software versions relatively to the first displayed version. The performance changes may consist of multiple metrics described in the plot legend. The X axis represents software versions by the date of commit to source control repository. The Y axis shows different metrics, which are displayed in percentage of a mean.

Error plot

Error plot (Figure 2.3) is divided into two large vertical segments with common Y axis showing the software versions by the date of commit to source control repository.

The first segment displays percentage of both failed compilations (green bar) and failed runs (red bar). The X axis of this segment shows values of the mentioned data starting from zero in order from right to left.

The second segment contains the number of systematic (yellow bar) and random (blue bar) errors occurred during the testing. The value of this data is presented in the X axis starting from zero in order from left to right.

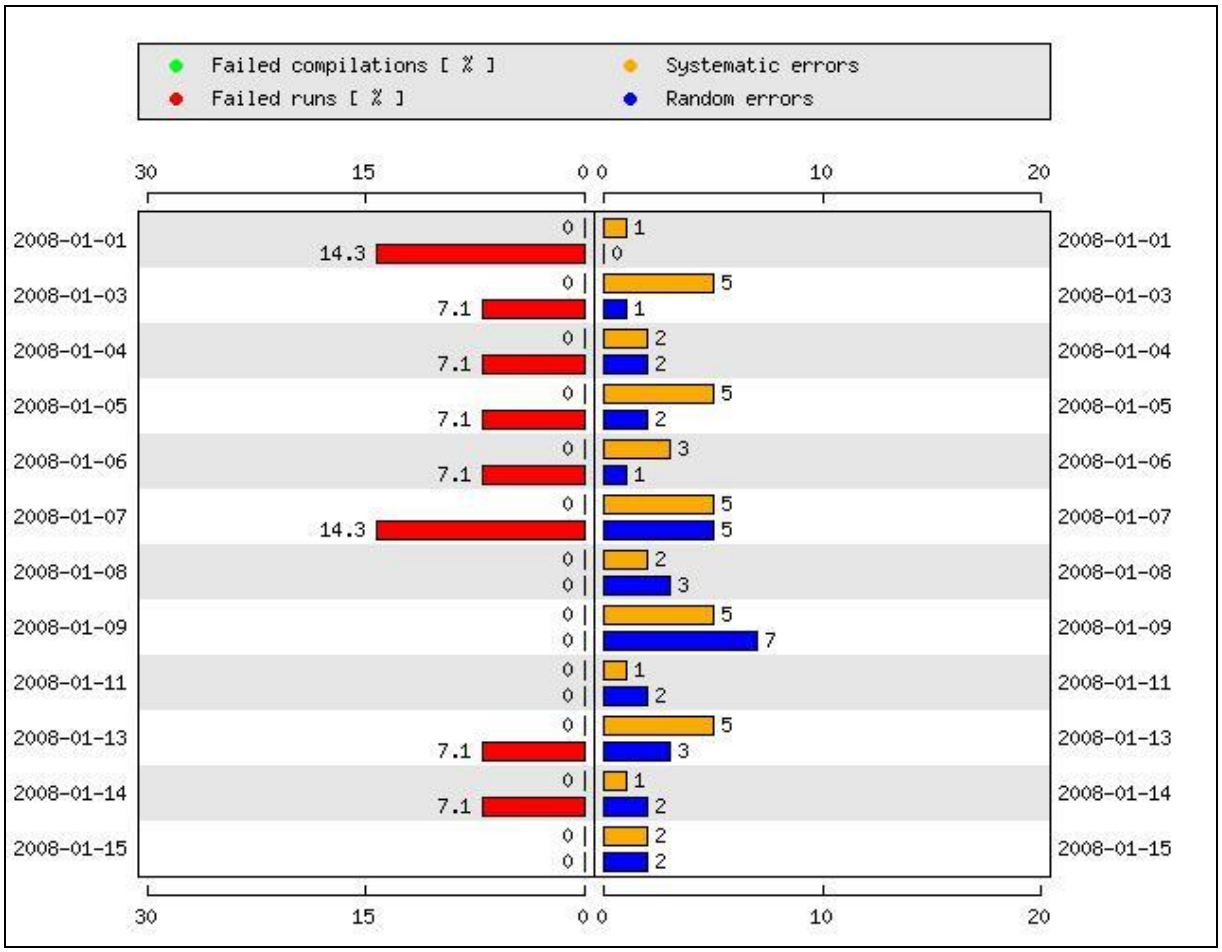


Figure 2.3 – Error plot

Project comparison plot

The project comparison plot (Figure 2.4) shows the mean of multiple project-platform couples viewed in the plot legend. In the X axis are displayed software versions by the date of commit to source control repository. The Y axis shows the mean value (colored circles) of each couple.

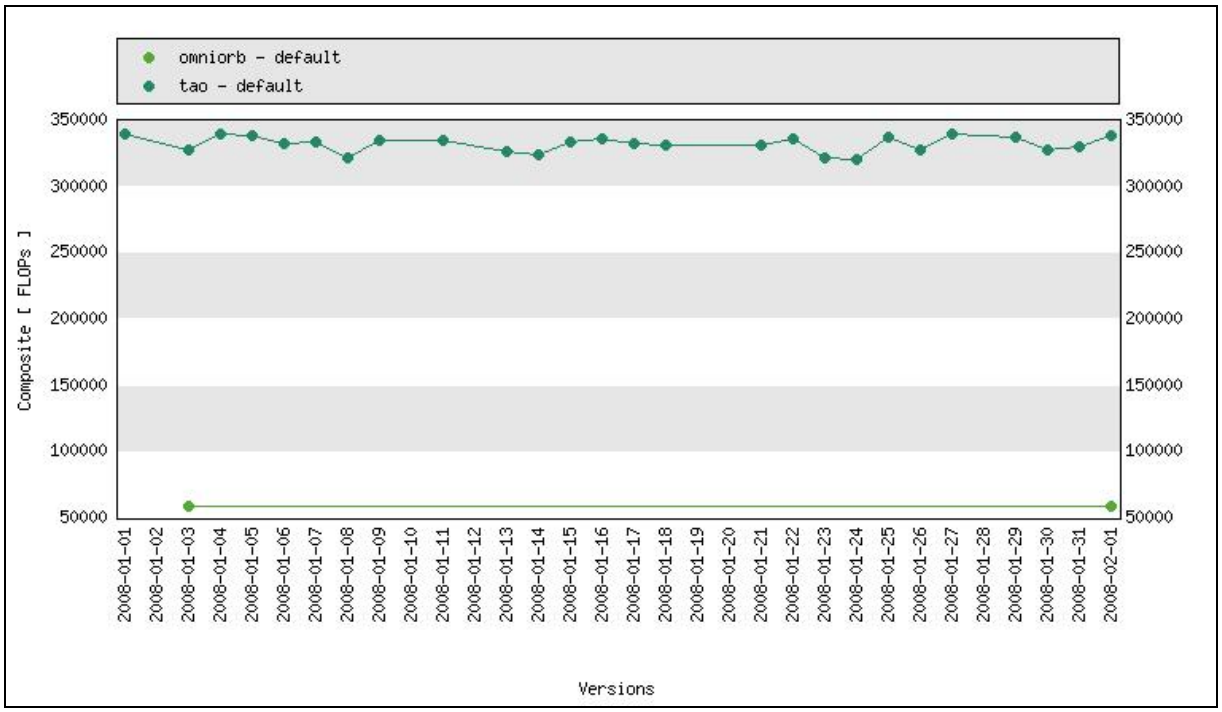


Figure 2.4 – Comparison plot

Text results

The text results (Figure 2.5) are presented as a table, which is divided into pairs of rows representing the software versions. The first of these rows contains the date of commit to source control repository, mean, performance impact, coefficient of variations, percentage of failed compilations, percentage of failed runs, number of random errors, number of systematic errors, change detection value and link to page displaying commit log data from current software version to the following version. The second row displays a table of compilation results. Each row of the compilation results table contains number, mean, coefficient of variations, percentage of failed runs, number of random errors, number of systematic errors and change detection value of one compilation.

Project version	Mean	Impact [%]	COV	Failed compilations [%]	Failed runs [%]	Random errors	Systematic errors	Change detected	Diff
2008-01-03	575262	=	17882	0	50	14	11	1	link
Compilation results									

#	Mean	COV	Failed runs [%]	Random errors	Systematic errors	Change detected
1	18767	17882	2100	14	11	15
2	0	0	0	0	0	0

Project version	Mean	Impact [%]	COV	Failed compilations [%]	Failed runs [%]	Random errors	Systematic errors	Change detected	Diff
2008-01-04	524933	-8.7	17882	0	50	14	11	1	link
Compilation results									

#	Mean	COV	Failed runs [%]	Random errors	Systematic errors	Change detected
1	18767	17882	2100	14	11	15
2	0	0	0	0	0	0

Figure 2.5 – Text results

Performance tables

Performance tables (Figure 2.6) consist of two tables. The first table assigns a color to each platform used in the second table. The second table is more complex. Columns of the table represent daily software versions and have multiple sub-columns, which are divided by the colors of the platforms. The rows of the table are divided into two types: benchmark rows and metric rows. The benchmark rows display only the colors of the platforms for each software version. Underneath each benchmark row, there are metric rows related to the benchmark. Metric rows contain the performance change from the previous version. For irrelevant performance change will be displayed "=" as a value and for missing software version results will be displayed "NA".

Color	Platform
	default

	2008-01-03	2008-01-04	2008-01-05	2008-01-06	2008-01-07	2008-01-08	2008-01-09	2008-01-10
xampler-instance								
m2[u2]	=	1	0.9	-0.8	-1.7	2.2	-2	NA
m3[u3]	=	-1.8	1.5	-0.1	1.5	-2.8	2.7	NA
m4[u4]	=	1	3.1	-4.4	1.4	-1.5	4.6	NA
m5[u5]	=	-5.2	3.9	0.6	-4.9	7.9	-5.9	NA
m6[u6]	=	-3.2	5	0.2	-5	3.6	3.7	NA
m7[u7]	=	0	0.2	-1.6	0.8	0.3	-0.8	NA
time[ms]	=	-8.7	-7.4	8.7	1.2	5	-0.4	NA
xampler-invocation								
Composite[FLOPs]	=	3.8	-0.6	-1.6	0.1	-3.4	4.1	NA
Monte Carlo[FLOPs]	=	4.7	-1.6	0.7	-2.6	2.4	0.4	NA
xampler-sequence								
m1[u1]	=	=	=	=	=	=	=	NA

Figure 2.6 – Performance tables

Notifications

This will be a feature for registered users, who would like to receive e-mail notifications about new performance testing results. The users will only have to select which project or projects to watch.

2.2 Results

As discussed in section 1.1, performance tests are usually performed more than once to get enough data to determine reliable conclusions. These tests are also often performed on several different software recompilations. Each of these performance tests will create a file with one to hundred or more samples for each metric. In worst case scenario the final size of one benchmark result file could outreach 500KB and just reading this file without any action will take approximately 0.5 seconds. Due to the quantity of the performance result files and time consumption of the process, it is quite necessary to cut down the result parsing to a minimum.

A fast way to process the results for further generation of possibly multiple plots is to parse the plain-text files only once, retrieve the data and store them in a form that is faster and comfortable to use, such as a relational database. This way, there is no need for further parsing and all the required data is prepared for next usage. However it would be quite ineffective to store all the gathered samples as it is in the database, because the database would soon be over-flooded and retrieving wanted data would take more or less the same time as parsing the performance results files. The database will therefore contain only the data that is most often viewed on regression benchmark web, such as mean, 99% confidence interval, coefficient of variations, errors occurred etc.

To access the parser, the user would open a special URL[4], which will activate the script. This URL will contain information about what sort of data is parsing and for what software version. There is possibility, that some version of the software will be retested, which could change the benchmarking results. The special URL can thus be used more than once, which will delete data stored in the database for current version and it will parse the results again.

2.3 Performance results plots

Since the performance results plot is the only type of plot that displays the individual samples from performance testing and the database does not store them (as discussed in the previous section), there is a problem getting data needed for this plot and generating it before exceeding the execution time limit (usually 30 seconds).

The simplest solution would be to parse all the data needed for the plot, but as discussed earlier, parsing is very slow and ineffective.

Caching performance results after parsing is another obvious solution for this problem. It would probably be fast enough to fit in the time limit, but generating plot will be still too slow to discourage many expectant users. Even if all required data was cached and there was no reason for parsing, the plot would still have to generate at least few million grey dots.

Third possibility is to cut off as many data as possible. Using all the data is not quite as necessary as it seems, because there is only a limited space for each software versions performance results in the plot, which means that many result dots are placed in the same pixel. The idea is thus to randomly select a smaller amount of benchmarking results during

[4] *"**Uniform Resource Locator (URL)** is a compact string of characters used to represent a resource available on the Internet"* [3]

the parsing process and storing them to the database. In the end it would be just a question of choosing the right amount of data, which would not over-flood the database and still would provide the required detail of the plot.

The ideal solution of this problem is to create an image during the parsing with all the performance results as a grey dots, store it somewhere on the hard drive and use it every time needed for plot generation. This way, the plot generator does not have to go through millions of performance results to create the dots during the generation, making the generator very fast and effective. Furthermore, the created image will have to be scaled down to fit the plot, making the dots blend together creating new dots in various grey tint as displayed in Figure 2.7, which is making the plot more pleasant to look at.

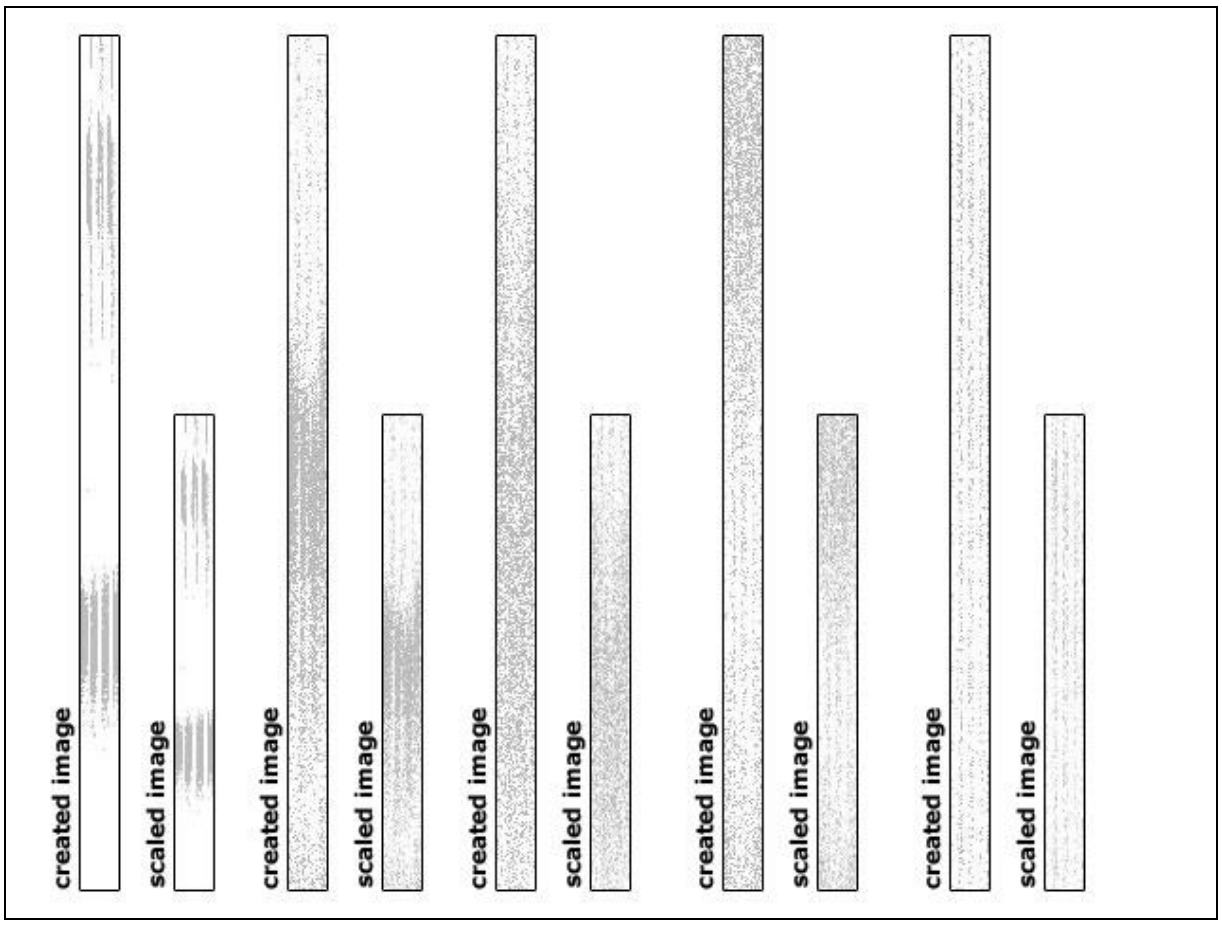


Figure 2.7 – Scaling images

2.4 Plot caching

As another way of speeding up the process of presenting regression benchmarking plots, we have considered the idea of caching the whole plot images, which would elegantly avoid repeated generating of plots. This process would require its own management, which would activate every time the cache exceeds maximum assigned hard drive space, leaving only the most used plots.

Unfortunately this idea has few major disadvantages. Because there are a lot of options for generating plots and one single plot can have from 50 to 400kB, it would require a lot of cache space just for a little probability of displaying a plot that has already been cached. To decide whether to use plot caching or not, it was necessary to test the speed of the plot generator.

	20 versions	50 versions	100 versions
Performance results plot	0.21s	0.38s	0.82s
Impact plot	0.13s	0.22s	0.34s
99% Confidence plot	0.09s	0.16s	0.29s
Error plot	0.14s	0.33s	0.62s
Project comparison plot	0.13s	0.26s	0.45s

Table 2.1 – Plot generator speed testing

Since the testing of the plot generator (Table 2.1) revealed that even for a 100 software version the results displayed in the plots are generated under one second, which is relatively fast, there is therefore no need for plot caching.

2.5 Implementation technologies

We chose Drupal [4], because it is free and plug-in able software distributed under the GPL[5] license implemented in the PHP[6] programming language, which was selected for implementing this web. Other alternatives could be for example Joomla [15], which also fulfills mentioned requirements. Drupal is able to use both MySQL[7] and PostgreSQL[8] databases, therefore this project can also use both of these databases.

[5] "*The **GNU General Public License** (**GNU GPL** or simply **GPL**) is a widely used free software license, originally written by Richard Stallman for the GNU project."* [5]

[6] "*Hypertext preprocessor (**PHP**) is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML*." [10]

[7] **MySQL** is widely-used database management system available under terms GNU GPL, as well as under a variety of proprietary agreements. [22]

[8] **PostgreSQL** is a free database management system released under a BSD-style license. [21]

3 Basic usage instructions

This section assumes that you have already installed Drupal 5 and read the getting-started guide [6].

3.1 Installation

Copy the special files ajax.php and plotgenerator.php to Drupal's root directory. This is very important otherwise the modules will not work properly.

Next, install the modules MTypes, Measurement, Parser and Subscriptions according the Drupal manual [7].

After correct installation, Drupal will write a message for each module – *Module-name* module has been successfully installed and Drupal menu will have five new sections – Benchmarks, Projects, Platforms, Measurement and Subscriptions, all described in the following section.

Fully installed and working regression benchmarking web can be seen on [19]. To access administrator account use username "admin" and password "aaa".

3.2 Sections

Aside from the five sections mentioned in the previous section, there is also one hidden parser section. Detailed description of the sections follows.

Benchmarks, Platform and Project sections

These sections are practically identical except for few differences, which will be described later. The following description will be demonstrated on Benchmarks, but it could be as easily used for Platforms and Projects.

This section takes care of the benchmark management. Clicking the Benchmarks section in Drupal menu will cause expanding the menu and show a list of all benchmarks as shown in Figure 3.1.

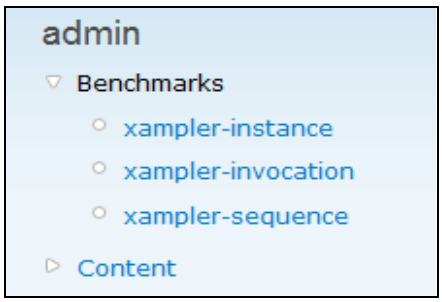


Figure 3.1 – Expanded menu

This list extended of published tag and description will be also displayed in the main frame, which will provide the possibility to add new benchmark as displayed in Figure 3.2.

Figure 3.2 – Benchmark list

Clicking the name of the benchmark will view the body of the benchmark with possible actions like View and Edit, which can be seen in Figure 3.3.

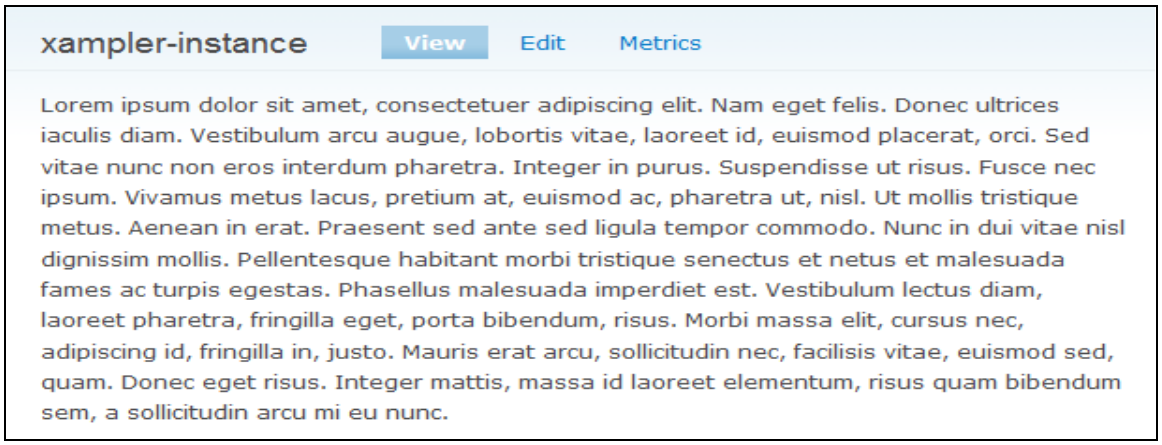


Figure 3.3 – Benchmark view

The Add new benchmark or Edit benchmark actions will display benchmark settings form, where the user can edit the name, the description, the body, the body-format, the publishing options and some additional data, which differs for each type of section.

Benchmarks settings have folder name, outlier percentage and axis scale percentage, which are used for parsing and discussed in section 3.4. Benchmarks settings also includes the metric management, where can be added, edited, deleted and order changed of the metrics as depicted in Figure 3.4.

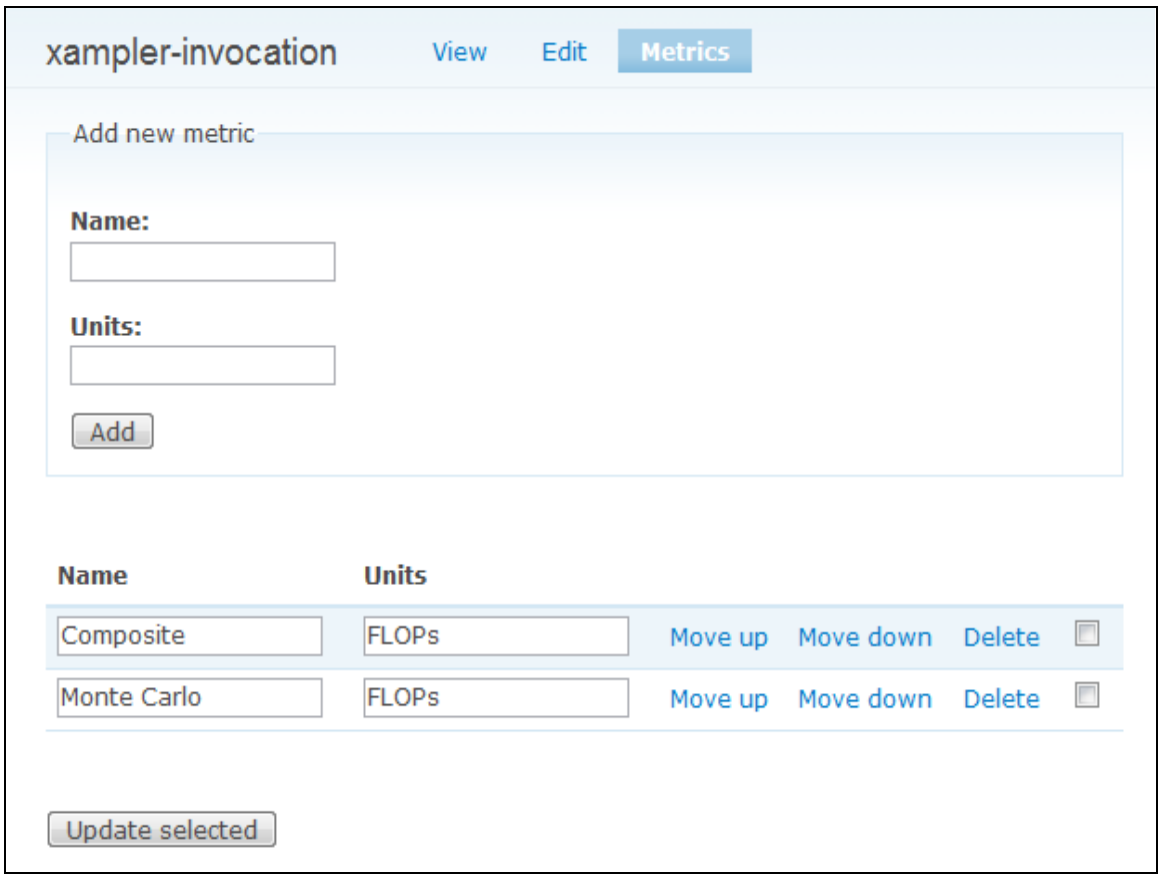


Figure 3.4 – Metric management

The additional options for project settings are Results URL used for the parsing and Diff URL, which provides the information about software changes. There is also dependence management in the project settings, which allows selecting pairs of benchmark and platform used for performance testing of current project. This dependence requires information about count of benchmark runs and count of compilations tested. For better understanding see Figure 3.5.

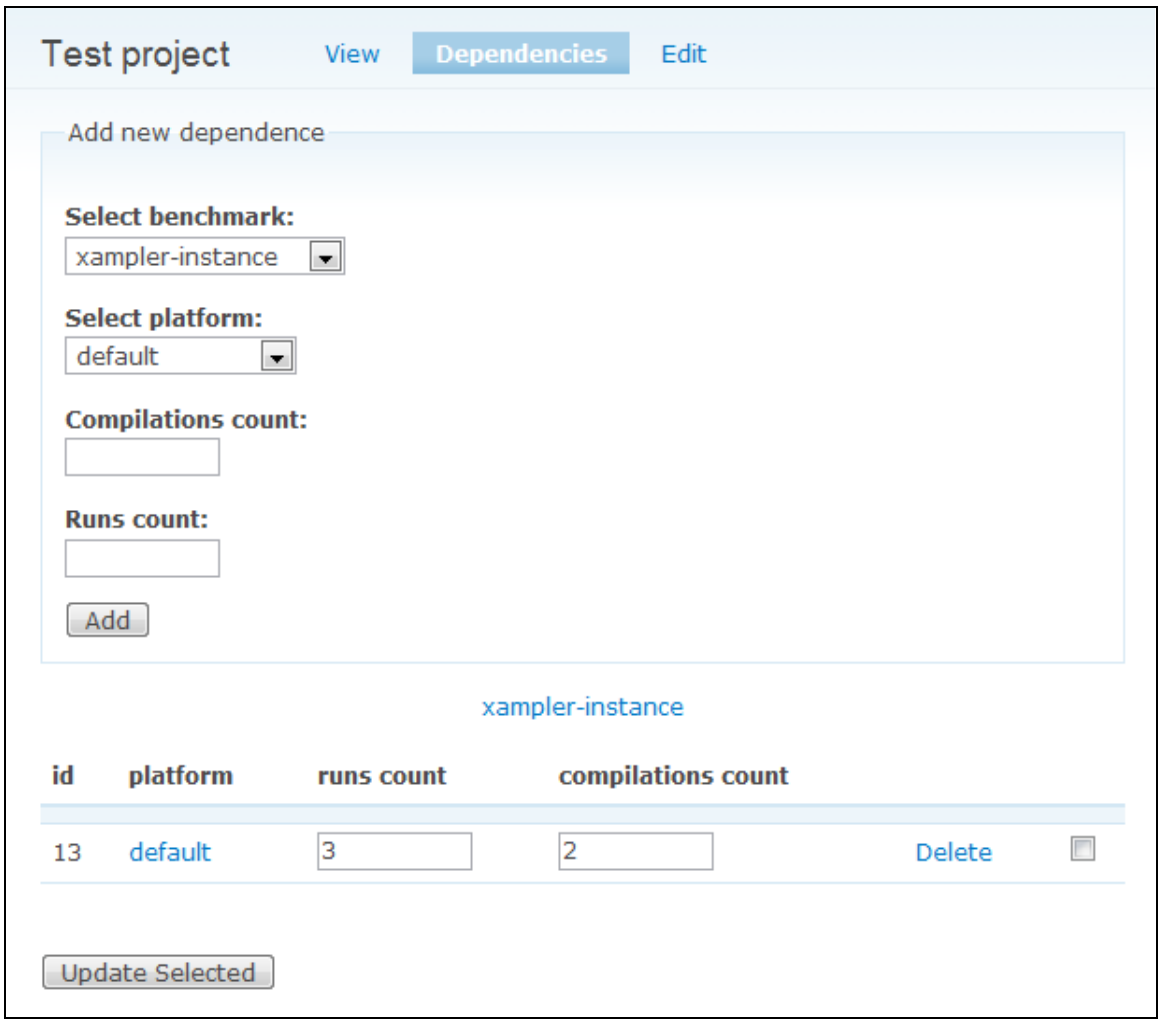


Figure 3.5 - Dependence management

The only additional option for platform settings is folder name, which is used for parsing.

Measurement section

All regression benchmarking results are presented in this section, which is divided into four subsections described in Table 3.1.

Subsection	Displays
Comparison of projects	Comparison plot – Figure 2.4.
Performance results	Performance results plot – Figure 2.1.
Performance tables	Performance table – Figure 2.5.
Regression plots	99% Confidence plot – Figure 2.2. Impact plot – Figure 1.1. Error plot – Figure 2.3. Text results – Figure 2.6

Table 3.1 – Measurement subsections

All features mentioned in Table 3.1 are described in section 2.1. Each subsection has a plot wizard, which sets up the data that will be displayed.

Subscriptions section

This section is only accessible to registered users, because it is working with registration data especially users e-mail. Users can sign up here for receiving notification e-mail about new measurement.

By clicking this section the main frame will display a form for adding new subscription and a list of all subscriptions, which were already added by the user. To add a new subscription it is necessary to choose a triplet of project, benchmark and platform, which benchmarking results needs to be watched and a change detection threshold. Every time, when a designated cron[9] job is started, there will be sent an e-mail to all users, who haves one or more subscription, if there are new benchmarking results for the given triplet with higher change detection than the threshold. The e-mail appearance can be set in the administration settings of subscriptions.

Parser section

Parser is activated by special URL composed of site URL, the "parser" string, dependence id and date string of current version. Dependence id is viewed in dependencies of project. More about parsing is written in section 3.3.

Parser URL example:
http://www.mysite.com/parser/5/20080219-235959

[9] "***Cron*** *is a time-based scheduling service in Unix-like computer operating systems."* [8]

3.3 Parsing

Because it is impossible to create a universal parser, which would be able to parse all the benchmarking results no matter the format, it requires a special file and folder structure of the parsed results for correct run.

Folder structure

For better transparency and faster access, the benchmarking results files must be stored in a folder structure consisted of four folders with specific names:

1. **Time string** of commit current software version to source control repository
2. **Benchmark folder name**
3. **Platform folder name**
4. **Software compilation number**

This structure needs to be in every results URL defined in the project settings and in the same order as viewed in the folder preview. The software compilation number folders contain the files with the results, which are named by numbers from one to benchmark runs count, and can be compressed by bzip2[10]. There also needs to be special files named "result" with the data from the statistical analysis, which are divided into two types:

1. **Complete analysis file** – this file is stored in the platform folder and contains statistical analysis data from all the benchmarking results created for current software version.
2. **Compilation analysis file** – every software compilation number folder must contain this file, which has the statistical analysis data for current compilation benchmarking results.

Numeric files structure

For every sample taken by benchmark must be a single line in the file containing the result. If the benchmark has more than one metrics, then the results of each metric must be separated by one or more of the following special characters: " ", "\t", "|" and ";". The parser also assumes that the metric results are in the same order as the metrics in the metric management. This solution for metric results is used in all the following file structures.

[10] **Bzip2** is a free lossless data compression program. [9]

Complete analysis files structure

Data from the analysis must be stored in the same way as shown in Table 3.2.

Line #	Used for all metrics	Description
1	Yes	Mean
2	Yes	Upper boundary of the 99% confidence interval
3	Yes	Lower boundary of the 99% confidence interval
4	Yes	Coefficient of variations
5	No	Number of failed compilations of the project
6	No	Number of failed runs of the benchmark
7	No	Number of random errors
8	No	Number of systematic errors
9	Yes	Change detection

Table 3.2 – Complete analysis files structure

Compilation analysis files structure

Structure of compilation analysis file displayed in Table 3.3 is similar to complete analysis files except for the absence of failed compilations line.

Line #	Used for all metrics	Description
1	Yes	Mean
2	Yes	Upper boundary of the 99% confidence interval
3	Yes	Lower boundary of the 99% confidence interval
4	Yes	Coefficient of variations
5	No	Number of failed runs of the benchmark
6	No	Number of random errors
7	No	Number of systematic errors
8	Yes	Change detection

Table 3.3 – Compilation analysis files structure

3.4 Permissions

The complete list with description of all permissions used for this project is viewed in the following Table 3.4.

Permission name	Allowed actions
Create benchmark	Add new benchmark Edit settings of own benchmarks Manage metrics of own benchmarks View own unpublished benchmarks
Create platform	Add new platform Edit settings of own platforms View own unpublished platforms
Create project	Add new project Edit settings of own projects Manage dependencies of own projects View own unpublished projects
Delete benchmark	Delete benchmark
Delete platform	Delete platform
Delete project	Delete project
Edit benchmark	Edit settings of all benchmarks Manage metrics of all benchmarks View all unpublished benchmarks
Edit platform	Edit settings of all platforms View all unpublished platforms
Edit project	Edit settings of all projects Manage dependencies of all projects View all unpublished projects
Access benchmark	View all published benchmarks
Access platforms	View all published platforms
Access projects	View all published projects and its dependencies
Access measurement	View measurement section
Access parser	May parse benchmarking results
Authenticated user	Manage own subscriptions
Anonymous user	Nothing

Table 3.4 - Permissions

4 Implementation

The regression benchmarking web is implemented in PHP 5 using AJAX[11] and MySQL 5 database. The web uses the Sarissa Library, which is distributed under the GNU GPL and offers "*various XML related goodies like Document instantiation, XML loading from URLs or strings, XSLT transformations, XPath queries etc*" as written on [12].

The project is dependent on Drupal 5 and cannot work without it. Correct functionality of modules on higher versions of Drupal would require adapting to the Drupal API[12] changes. Higher versions of Drupal have its own support of AJAX, therefore it would allow to use the built-in AJAX management functions instead of the Sarissa library.

Since Drupal supports MySQL and PostgreSQL, the implemented modules will work with both of these database systems.

4.1 Architecture

The architecture comprises the following components: Measurement module, MTypes module, Parser module, Subscriptions module, ajax.php, plogenerator.php and plot.inc. The logical model of these components is depicted in Figure 4.1 and individual components are described further in this section.

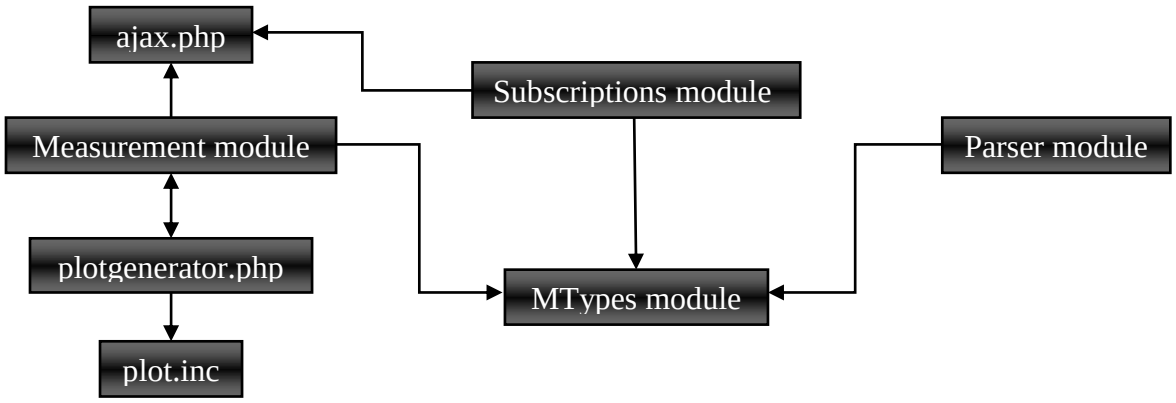


Figure 4.1 – Logical model of implementation components

MTypes module

This component provides the benchmark, platform and project management. It contains all database related functions to benchmarks, platforms, projects, metrics and dependencies, which are used by all other module components.

[11] "***Ajax*** *(asynchronous JavaScript and XML), or* ***AJAX****, is a group of interrelated web development techniques used for creating interactive web applications or rich Internet applications.*" [11]
[12] Application programming interface (**API**)

Measurement module

This component forms measurement section. It is mainly displaying plot wizards, which uses AJAX for better transparency and taking care of actions that are triggered by these wizards. The data collected by the plot wizard is given to plotgenerator.php, which creates appropriate plots.

Parser module

This module finds a path to a folder containing benchmarking results that should be parsed. The data is parsed twice, first to get axis boundaries for the image, displayed in performance results plots, according to benchmark settings (outlier percentage and axis scale percentage) and then second time to create the mentioned image by putting grey dots representing the benchmarking results. In the end it parses the statistical analysis data and stores them into the database.

Subscriptions module

This component creates the section subscriptions. It has two tasks: manage subscriptions and send required e-mails to users.

ajax.php

This component only uses given data to invoke correct hook AJAX function from the modules. Theses hook AJAX functions returns XML code, which is used for further operations.

plotgenerator.php

This component is used to generate data required for displaying plot and using the "PLOT" class from plot.inc file to generate the plot.

plot.inc

This component uses all the data given to create plot. Further discussion about plot generating is in following section 4.2.

4.2 Plot generation

As mentioned earlier plots are generated by class "PLOT" from the file plot.inc. Before plot can be generated, this file requires following data: type of plot, minimal results value, maximal results value, results in supported format and if needed legend information.

After adding the minimal and maximal results value this class computes ideal axis scale that would display all the required data with minimal space waste and provide as much transparency possible.

Before generating the plot there also needs to be computed width and height of the plot, which can change depending on the amount of displayed data. The basic plot generation consists in creating blank JPEG[13] image and filling it with text, lines, circles, squares and other objects forming required plot.

[13] "***JPEG*** *(Joint Photographic Experts Group) is a commonly used method of compression for photographic images.*" [13]

4.3 Process flow

The Plot Generator process flow (Figure 4.2) and the Parser process flow (Figure 4.3) are depicted in the following UML[14] sequence diagrams

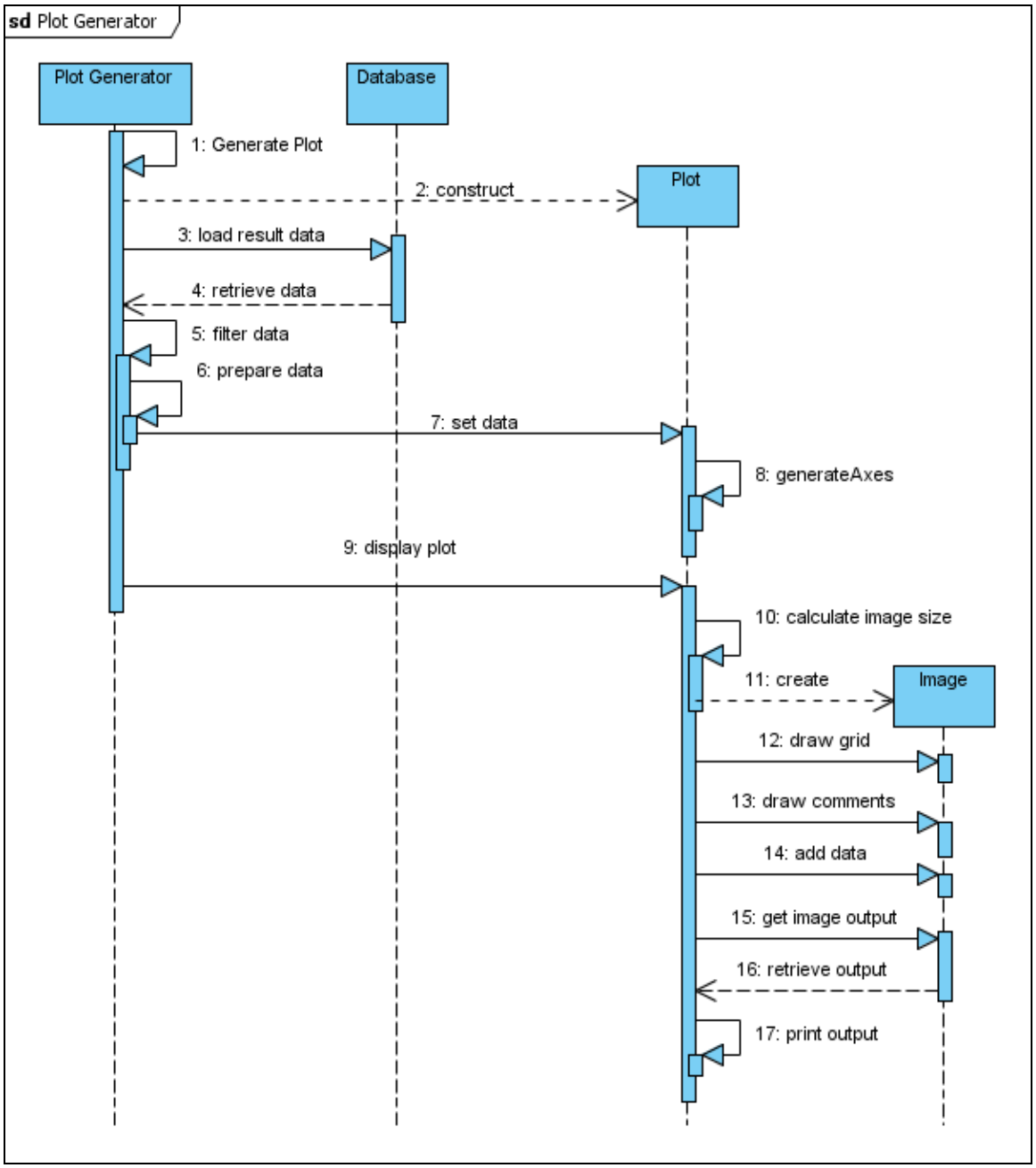


Figure 4.2 – Plot Generator process flow

[14] Unified Modeling Language (**UML**)

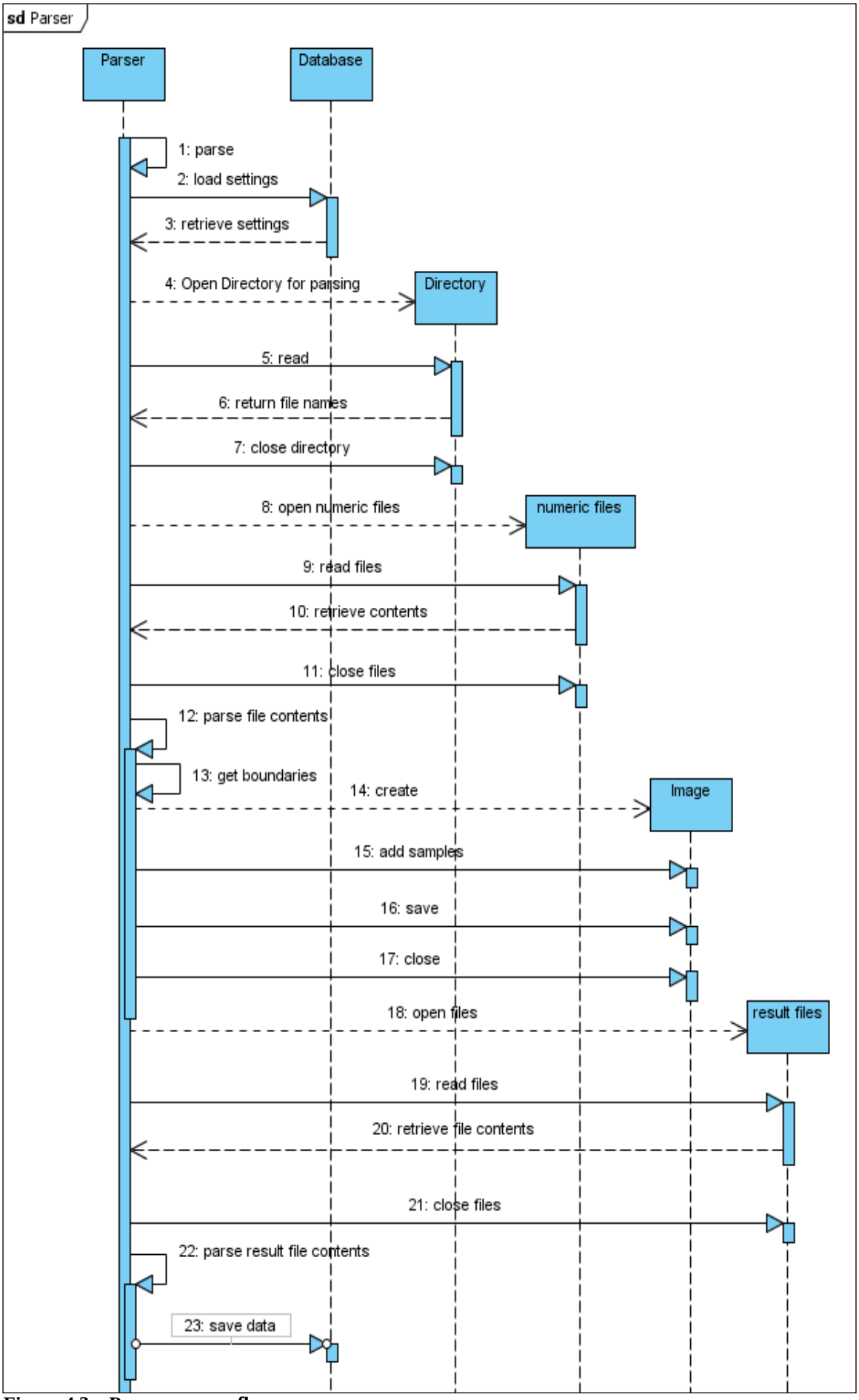


Figure 4.3 – Parser process flow

4.4 Database

The table relations are depicted in Figure 4.4 and the complete list of tables with detailed description follows.

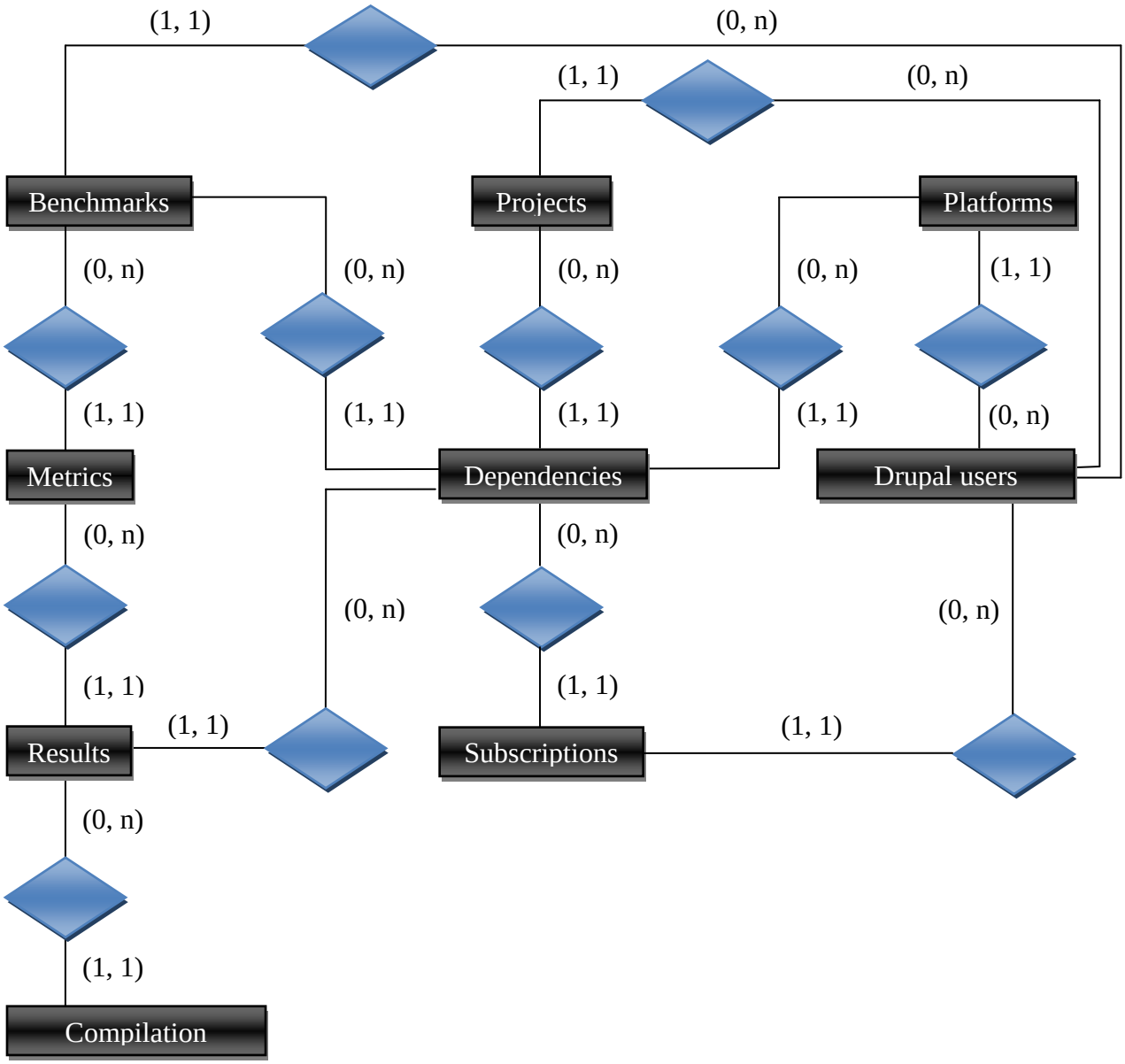


Figure 4.4 – Database ER model

Benchmarks

This table contains benchmarks settings, more displayed in Table 4.1.

Field name	Type	Description
Id	integer(10)	Benchmark identification number
Title	varchar(128)	The name of the benchmark
Description	mediumtext	Short description viewed in the list of benchmarks
Body	longtext	Full text
Format	integer(11)	Format of the body
folder_name	varchar(255)	Name of the folder used for storing results
Outlier	double	Outlier percentage
axis_scale	double	Axis scale percentage
Uid	integer(11)	Identification number of user, who added this benchmark
Created	integer(11)	Creation time in timestamp
Changed	integer(11)	Time of last change in timestamp
Published	char(1)	"N" for not published and "Y" for published

Table 4.1 – Benchmarks table

Projects

This table contains projects settings, more displayed in Table 4.2.

Field name	Type	Description
Id	integer(10)	Project identification number
Title	varchar(128)	The name of the project
Description	mediumtext	Short description viewed in the list of projects
Body	longtext	Full text
Format	integer(11)	Format of the body
results_url	varchar(255)	Path to results
commitlog_url	varchar(255)	Path to commit log
date_format	varchar(64)	String determining the time format of the time string folders mentioned in section 3.3.
Uid	integer(11)	Identification number of user, who added this benchmark
Created	integer(11)	Creation time in timestamp
Changed	integer(11)	Time of last change in timestamp
Published	char(1)	"N" for not published and "Y" for published

Table 4.2 – Projects table

Platforms

This table contains platforms settings, more displayed in Table 4.3.

Field name	Type	Description
Id	integer(10)	Platform identification number
Title	varchar(128)	The name of the platform
Description	mediumtext	Short description viewed in the list of platforms
Body	longtext	Full text
Format	integer(11)	Format of the body
folder_name	varchar(255)	Name of the folder used for storing results
Uid	integer(11)	Identification number of user, who added this benchmark
Created	integer(11)	Creation time in timestamp
Changed	integer(11)	Time of last change in timestamp
Published	char(1)	"N" for not published and "Y" for published

Table 4.3 – Platforms table

Metrics

This table contains metrics information, more displayed in Table 4.4.

Field name	Type	Description
Id	integer(11)	Metric identification number
benchmark_id	integer(11)	Identification number of benchmark, which uses this metric
Name	varchar(64)	Name of the metric
Units	varchar(64)	Units used for measurement
Order	integer(2)	The order number of the metric in the benchmarking results files

Table 4.4 – Metrics table

Dependencies

This table contains data of dependencies between benchmarks, platforms and projects. More information about this table is displayed in Table 4.5.

Field name	Type	Description
Id	integer(11)	Identification number of the dependence
benchmark_id	integer(11)	Identification number of benchmark
platform_id	integer(11)	Identification number of platform
project_id	integer(11)	Identification number of project
comp_cnt	integer(11)	Count of compilations made for this dependence
runs_cnt	integer(11)	Count of benchmark runs done for each compilation

Table 4.5 – Dependencies table

Results

This table contains the results data, more displayed in Table 4.6.

Field name	Type	Description
Id	integer(11)	Identification number of the result
dependence_id	integer(11)	Identification number of dependence
metric_id	integer(11)	Identification number of metric
viewed_min	double	Lower boundary of image created from parsing
viewed_max	double	Upper boundary of image created from parsing
Time	integer(11)	The time of commit to of measured software
Mean	double	Mean
confidence_min	double	The lower boundary of the 99% confidence interval
confidence_max	double	The upper boundary of the 99% confidence interval
Cov	double	Coefficient of variations
Ecompilation	integer(11)	Count of failed compilations
Eruns	integer(11)	Count of failed benchmark runs
Erandom	integer(11)	Count of random errors
Esystematic	integer(11)	Count of systematic errors
Chdetected	double	Change detection value

Table 4.6 – Results table

Compilation results

This table contains the data of compilation results, more information in Table 4.7.

Field name	Type	Description
Id	integer(11)	Identification number of the compilation result
result_id	integer(11)	Identification number of full results of current software
Compnumber	integer(11)	Number of compilation of this software
Mean	double	Mean
confidence_min	double	The lower boundary of the 99% confidence interval
confidence_max	double	The upper boundary of the 99% confidence interval
Cov	double	Coefficient of variations
Eruns	integer(11)	Count of failed benchmark runs
Erandom	integer(11)	Count of random errors
Esystematic	integer(11)	Count of systematic errors
Chdetected	double	Change detection value

Table 4.7 – Compilation results table

Subscriptions

This table contains subscriptions data, more displayed in Table 4.8.

Field name	Type	Description
Id	integer(11)	Identification number of the compilation result
user_id	integer(11)	Identification number of user, who created this subscription
dependence_id	integer(11)	Identification number of dependence
Mindetection	double	Minimum change detection needed to send the e-mail

Table 4.8 – Subscription table

5 Conclusion

The regression benchmarking web implementation fulfills all expected requirements mentioned in section 2.1, creating a flexible web interface for displaying benchmarking results in various user friendly forms.

Even though the benchmarking results parser is based on the most common benchmarking results files, it can be difficult to find live benchmarking data with current format especially with the statistical analysis data file. The parser could increase its flexibility by adding the possibility to choose the correct results file structure in the benchmark settings and it could provide its own statistical analysis data, but in the end the parser would be too slow to handle large amount of results data. Since this project is made for Distributed System Research Group, Charles University, Prague [14], which pledged providing results in correct format, there was no need of further analysis of this problem.

The speed of the parser is dependent on the amount of data parsed. The feasible data amount for parsing would be up to approximately 1 MB, which is parsed in about 10 seconds. By increasing the data amount the parser becomes less effective. When the data amount increases to approximately 2 MB the parser will throw an error of exceeding the maximum execution time.

There are few other factors, which could have influence on the web application process such as too many performance testing results or too many users connected. The first option is dependant to the database system, which stores the result data, because too much data can over-flood the database. Since the amount of data for one measurement is not that large, the possibility of over-flood is low. The second factor could affect the speed of generating plots or displaying the page.

5.1 Related work

Since there are not many regression benchmarking oriented webs on the internet, the regression benchmarking web was designed mainly on the MONO Regression Benchmarking [2] with few inspirations from following webs.

CSiBe

The CSiBe is an online benchmark, which measures the compilation time and the size of code generated by GCC[15], and is designed to help GCC developers to avoid or fix code-size growth, compilation slow-down and code performance degradation [17].

Eclipse performance results

This web site displays performance results of Eclipse version 3.2.1 relative to version 3.1. It measures the time needed for performing various actions such as opening Ant editor or initializing plug-ins and compares them to old Eclipse version results. These test are performed on several platforms and displayed using impact plots [18].

[15] GNU Compiler Collection (GCC)

5.2 Future work

In the future it would be probably better to upgrade to the latest version of Drupal, because it is more mature, providing more possibilities and bug-fixes.

The next logical step would be to remake the parser to other language then PHP such as C++, which would provide more speed and comfort, because it will not be bounded by expiration time limit. This way the parser could provide its own statistical analysis and would not be dependant of files created by other scripts.

6 Bibliography

[1] Smith, C.U., Williams, L.G.: Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software, Addison-Wesley, 2001

[2] MONO Regression Benchmarking, http://dsrg.mff.cuni.cz/projects/mono/

[3] Uniform Resource Locator – Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/URL

[4] About Drupal | drupal.org, http://drupal.org/about

[5] GNU General Public License – Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/GPL

[6] Getting started | drupal.org, http://drupal.org/getting-started

[7] Installing Modules | drupal.org, http://drupal.org/getting-started/5/install-contrib/modules

[8] Cron – Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Cron

[9] Bzip2, http://www.bzip.org/

[10] PHP: Hypertext preprocessor, http://www.php.net/

[11] Ajax (programming) – Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Ajax_%28programming%29

[12] Sarissa – Sarissa Home Page, http://dev.abiss.gr/sarissa/

[13] JPEG – Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Jpeg

[14] Distributed Systems Research Group, Prague, http://dsrg.mff.cuni.cz/

[15] Joomla!, http://www.joomla.org/

[16] Kalibera, T., Tuma, P.: Precise Regression Benchmarking with Random Effects: Improving Mono Benchmark Results, in proceedings of Third European Performance Engineering Workshop (EPEW 2006), Springer-Verlag, 2006

[17] GCC Code-Size Benchmark Environment (CSiBE), http://www.inf.u-szeged.hu/csibe/

[18] Eclipse Performance Results, http://archive.eclipse.org/eclipse/downloads/drops/R-3.2.1-200609210945/performance/performance.php

[19] Regression Benchmarking Web, http://ozzy.kabel1.cz/regressionweb/

[20] MONO, http://www.mono-project.com/Main_Page

[21] PostgreSQL, http://www.postgresql.org/

[22] MySQL, http://www.mysql.com/

7 Appendices

All appendices can be found on the enclosed CD.

Appendix A

Results for parser testing
resuts.zip

Appendix B

Drupal’s 5 installation
drupal5.zip

Appendix C

Project implementation
measurement_modules.zip

Appendix D

Electronic form of the thesis
thesis.pdf

Appendix E

Generated PHPDocs
phpdocs.zip