Evaluation on Real Data: Benchmark Result Framework

D6.3 v1.0

WP6 – Evaluation on Real Data: D6.3 Benchmark result framework

Dissemination Level: Confidential

Lead Editor: Miha Stajdohar

Date: 05/10/2013

Status: Final

Description from Description of Work:

T6.3 Benchmark result framework/web access: In this task we aim to provide a framework to allow each benchmark run to be searched and catalogued with all measurables in one place. Integrated diagnostics and system metrics, graphs and web pages that allow access to each benchmark run. Various diagnostics and test code supplied by partners will be hooked into this.

Contributors:

Miha Štajdohar, UL
Janez Demsar, UL

Internal Reviewer(s):

2ndQ: Martin Marques
UNIMAN: Mikel Lujan

Version History

Version	Date	Authors	Sections Affected
0.1	25/09/2013		Initial version
0.2	30/09/2013	Miha Štajdohar	Feedback
0.3	05/10/2013	Miha Štajdohar	Summary, ToC
1.0	29/10/2013	Miha Štajdohar	Incorporate feedback

Table of Contents

List of Figures

Abbreviations

RESTful API	Representational state transfer like application programming interface
MVC	Model-View-Controller

1. Summary

This document describes the benchmark result framework, its features, specifications and implementation details.

2. Description

We developed a framework that allows each benchmark result to be stored and catalogued in one place and provides search and retrieval capabilities with some basic graphical analysis. We integrated RESTful API[1] so platform can be accessed from scripts.

We designed a web-based application with JavaScript and HTML5 powered user interface and Django backend.

Data model

A benchmark measurement consists of:

- required fields (title, date, user),
- tags / keywords,
- arbitrary custom fields that can be either strings or numbers.

The data model does not explicitly distinguish between the dependent and independent variables, that is, the manipulated and the measured data.

Data import

The data can be input through an upload form, in which the user enters all data manually or through a RESTful API.

In a manual submission, the user starts with a form containing the required fields and manually adds new fields.

For submission through RESTful API, we provided examples on how to send benchmark data to the server. We expect that the majority of the data will be submitted automatically, through scripts, and not through the web interface.

[1] Representational State Transfer (REST) is an architectural style that abstracts the backend. Our implementation uses the four HTTP methods GET, POST, PUT and DELETE to execute different operations on benchmark entries.

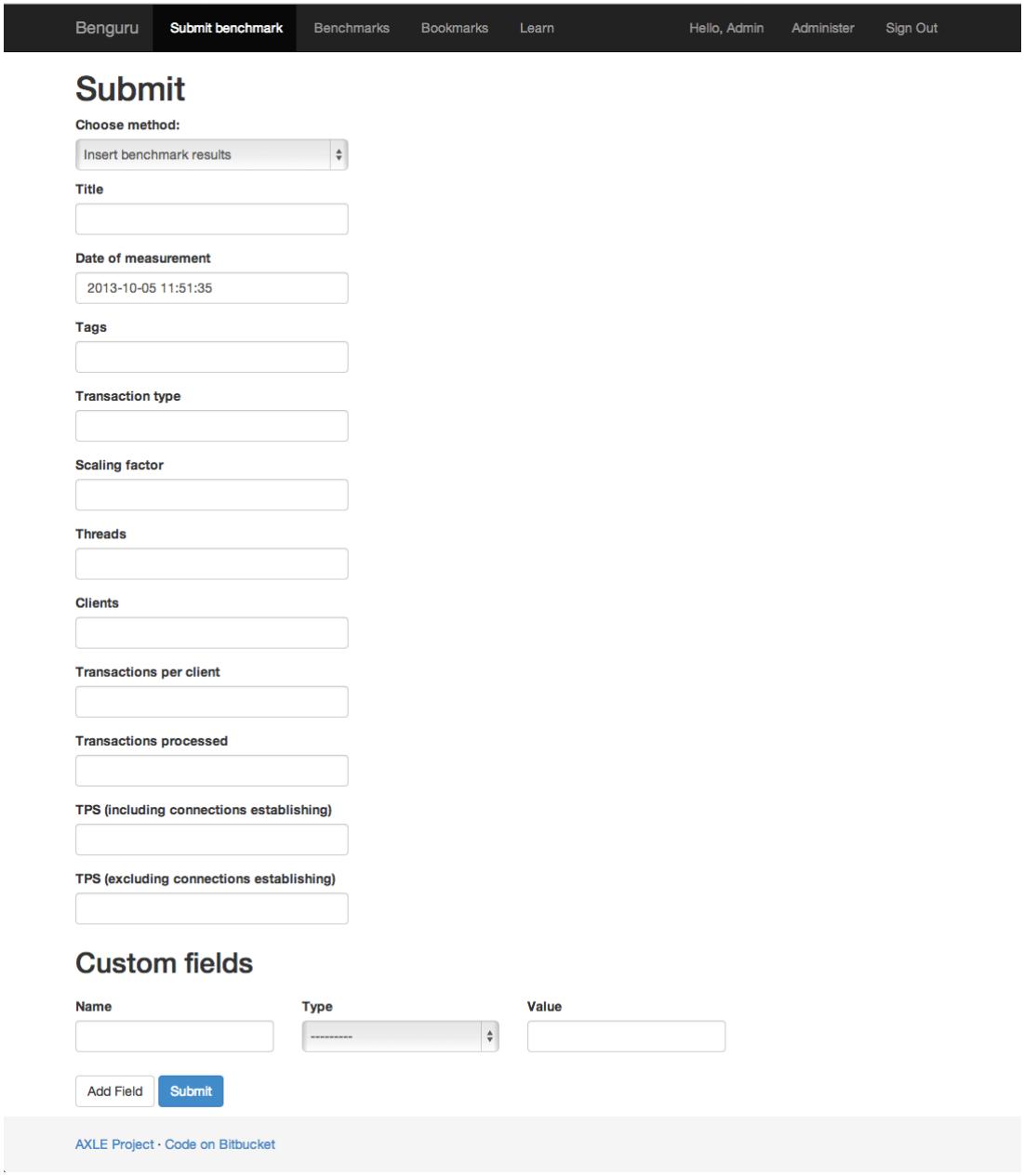

Figure 1: Submit benchmark measurement

Search and retrieval

Users search for records based on values of the fields. The system lists all measurements that match the criteria and lets the user select and then view a particular measurement.

In addition, users can specify a set of observables. For instance, they query for benchmarks tagged "Memory" that were submitted in August 2013 and use transactions per second and memory usage as observables. The result is a bar chart whose bars represent the tests with each bar separated further into transactions per second and memory usage (Figure 2).

The same capabilities are also available through the RESTful API. Depending on the needs of consortium, we will provide shell scripts that will query the data and return the results in the appropriate form.

For the case of retrieval, we expect that most users will use the web interface and not scripts (as opposed to the data import).

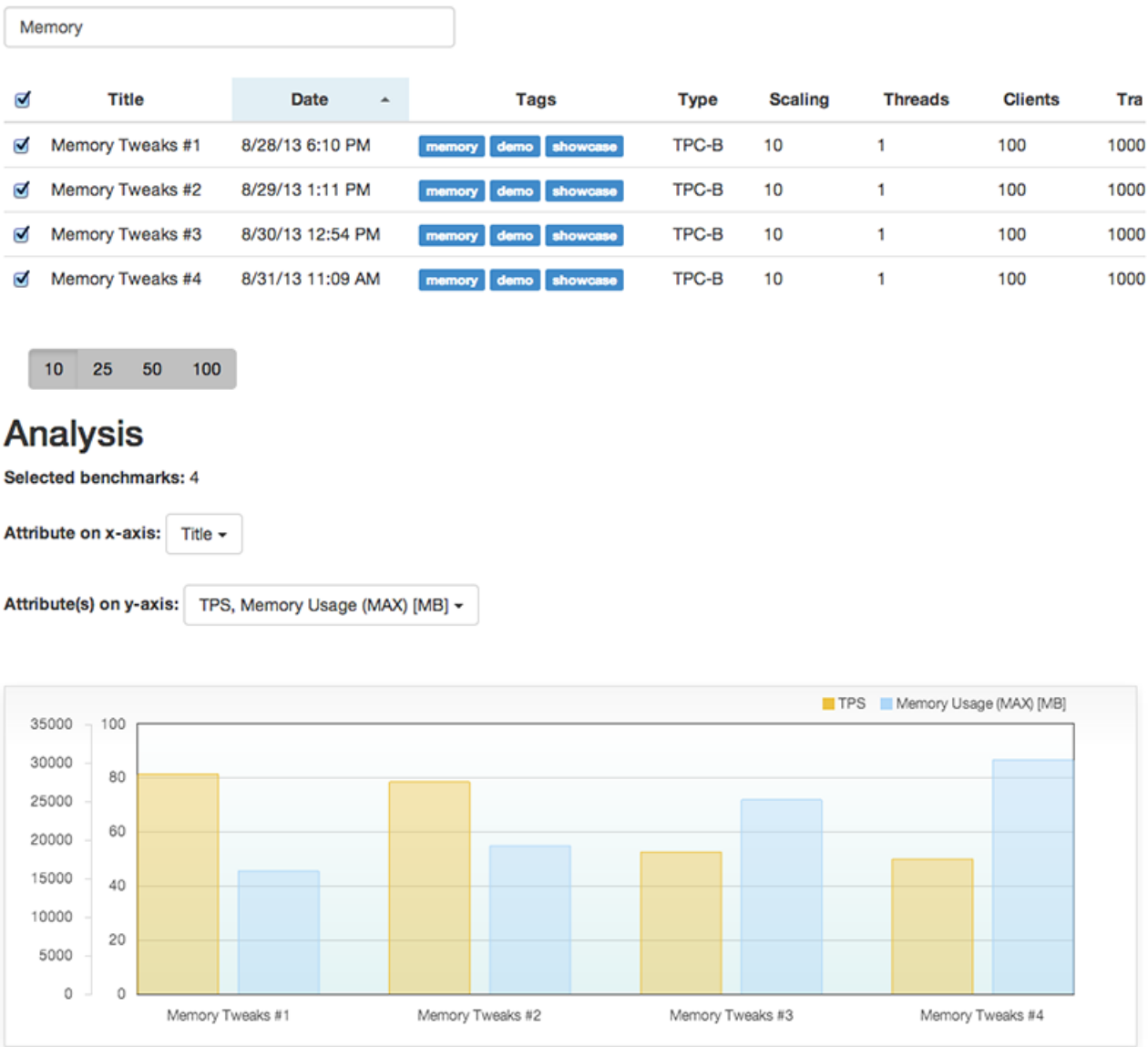

Figure 2: Filter benchmarks and select observables

3. Implementation

Architecture Design

The web application is built on Django web framework, PostgreSQL and the AngularJS JavaScript MVC[2] library.

Using Django's Object Relational Mapper (ORM) we defined our data model entirely in Python classes. We defined some of the most used metrics in the benchmark model.

Benchmark:

- id
- title
- date
- user
- tags
- transaction_type
- scaling_factor
- threads
- clients
- transactions_per_client
- transactions_processed
- tps_including
- tps_excluding

Custom Fields:

- id
- benchmark id
- name
- type
- value

This object model is accessed with auto-generated Django database API and Tastypie based RESTful API. This enables complete modularity between our web application and the database. The SQL database can not be accessed through any other interface. The provided Django admin interface is used to set up and initialize the database.

The user interface is written in a combination of HTML5, CSS and JavaScript using functionality and UI components. We used the Django templating system and Angular MVC framework to maintain strict separation between the view and the controller. Additionally, an

[2] Model-View-Controller (MVC) is a software architecture pattern which separates the representation of information from the user's interaction with it.

inheritance hierarchy of templates is maintained to minimize repetition in the markup.

We deployed the benchmark result framework—Benguru v0.1—on the server of the Faculty of Computer and Information Science, University of Ljubljana: http://benguru.fri.uni-lj.si/.

Documentation

All public functions and methods are documented using the standard Python docstrings in the rst format.

Dependencies

Required libraries: Django, Python-MimeParse, Django-Tastypie, Psycopg2 and Django-Widget-Tweaks. All requirements can be downloaded from the PyPI—the Python Package Index.

License

The code is available on Github (https://bitbucket.org/biolab/benguru). All the code is released under BSD license.

4. References

- Benguru v0.1 web application: http://benguru.fri.uni-lj.si/
- PyPI—the Python Package Index: https://pypi.python.org/pypi
- Django framework: https://www.djangoproject.com/
- AngularJS framework: http://angularjs.org/
- Tastypie RESTful API: http://tastypieapi.org/