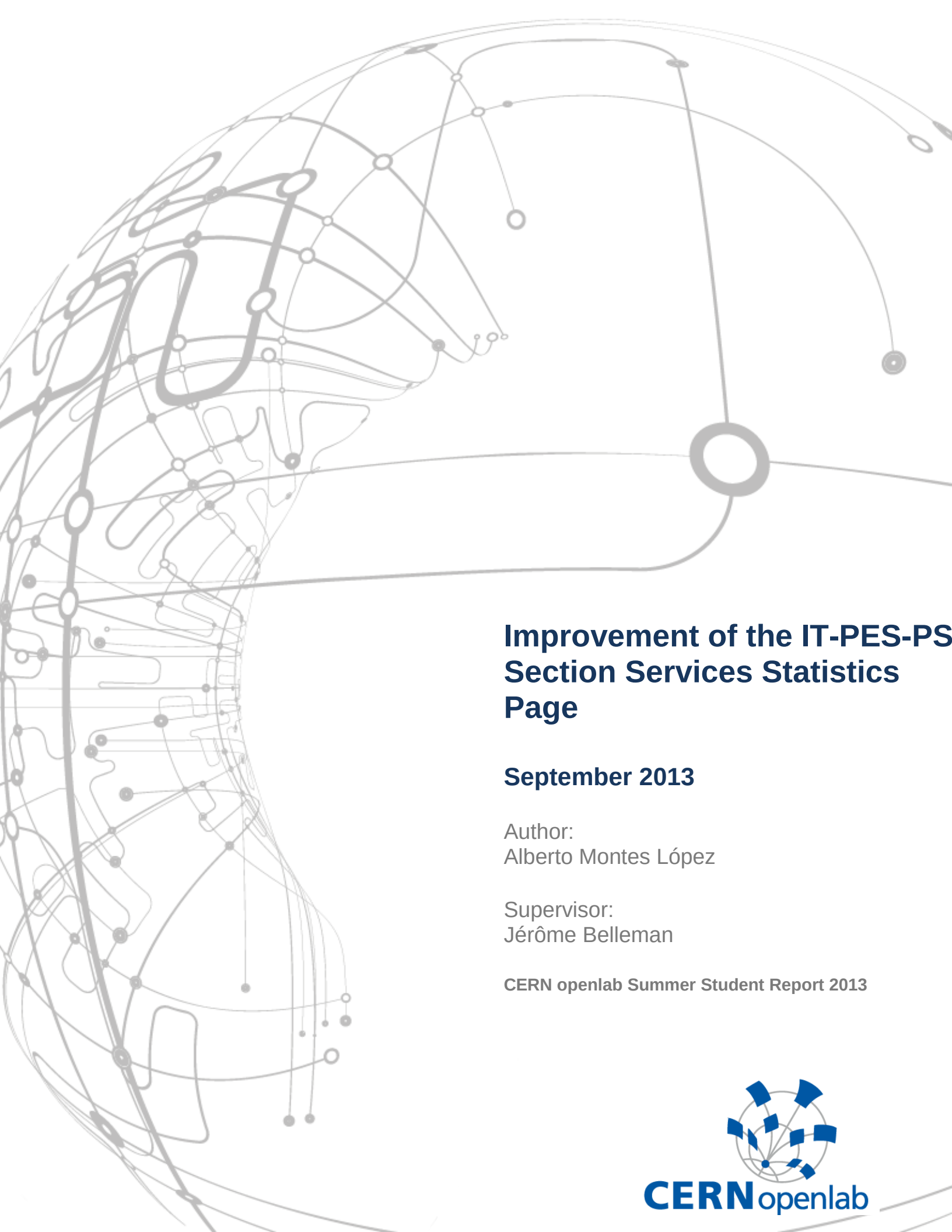


Improvement of the IT-PES-PS Section Services Statistics Page

September 2013

Author:
Alberto Montes López

Supervisor:
Jérôme Belleman

CERN openlab Summer Student Report 2013

CERN openlab

Project Specification

The IT-PES-PS service managers gather a lot of statistics for the services they run. These statistics are currently displayed by SLS (Service Level Status) or Lemon pages. They also use the Web interface provided with OpenTSDB, a DB optimised for time series. And while these various pages give very useful and technical information, they do not always emphasise the important figures.

Having different statistics pages makes it difficult to see the relevant numbers at once.

The goal of this project was to build homogeneous dashboards with interactive plots to better reflect the activity and resources of each service, showing relevant figures at first sight in a single website.

Abstract

There is a need in the IT-PES-PS section to improve the current situation of having to consult the relevant information from several heterogeneous websites by introducing interactive homogeneous dashboards, accessible from a single web application where the information needed can be quickly accessed.

The goal of this openlab Summer Student project was to create a website with homogeneous and interactive dashboards. Its architecture had to allow the creation of new dashboards easily.

Table of Contents

1 Introduction

The first goal of this project was to assemble the components necessary to display dashboards for the Batch Service run by IT-PES-PS. But the structure of this website would later on allow adding new dashboards by following a few steps which will be described later in this report.

2 Technologies

2.1 Technologies: What did we have in the beginning?

The first technology to introduce is OpenTSDB which is a distributed, scalable Time Series Database (TSDB) written on top of Hbase, where data is currently stored for the Batch Service.

There is a website where you can query directly the DB. It communicates with OpenTSDB using an HTTP API (Figure 1). Interacting with the user interface isn't optimal and requires the user to perform quite a few operations before he sees what he wants. For example, the user can only see one plot at a time.

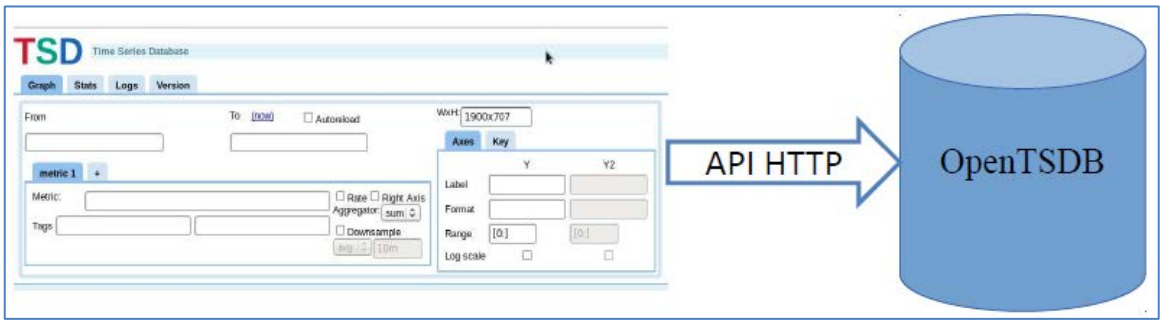
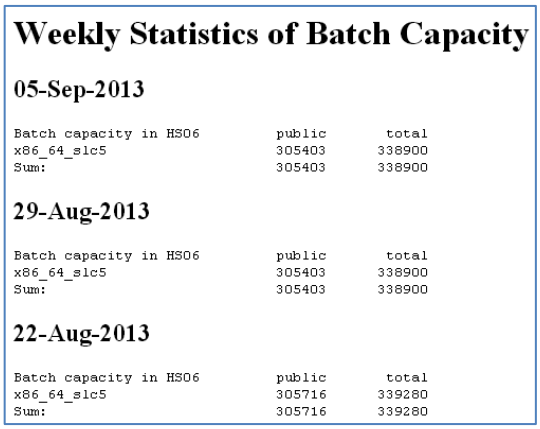


Figure 1. OpenTSDB is bundled with a web interface, in addition to the storage and the API.

SLS and Lemon are used as sources of information for the other services run by IT-PES-PS.



Weekly Statistics of Batch Capacity

05-Sep-2013

Batch capacity in HS06	public	total
x86_64_slc5	305403	338900
Sum:	305403	338900

29-Aug-2013

Batch capacity in HS06	public	total
x86_64_slc5	305403	338900
Sum:	305403	338900

22-Aug-2013

Batch capacity in HS06	public	total
x86_64_slc5	305716	339280
Sum:	305716	339280

Figure 2. Weekly Statistics of Batch Capacity.

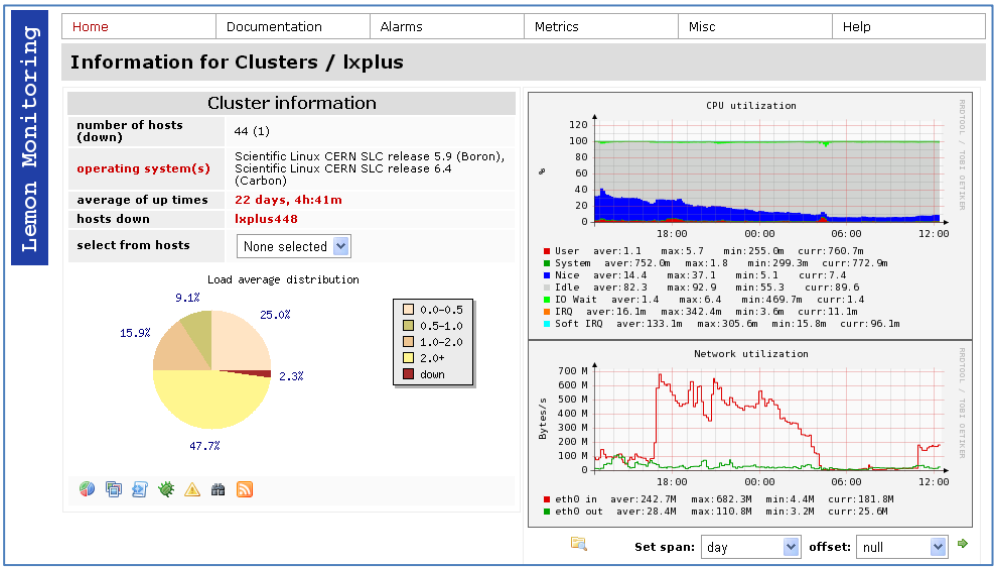


Figure 3. Example of Lemon page.

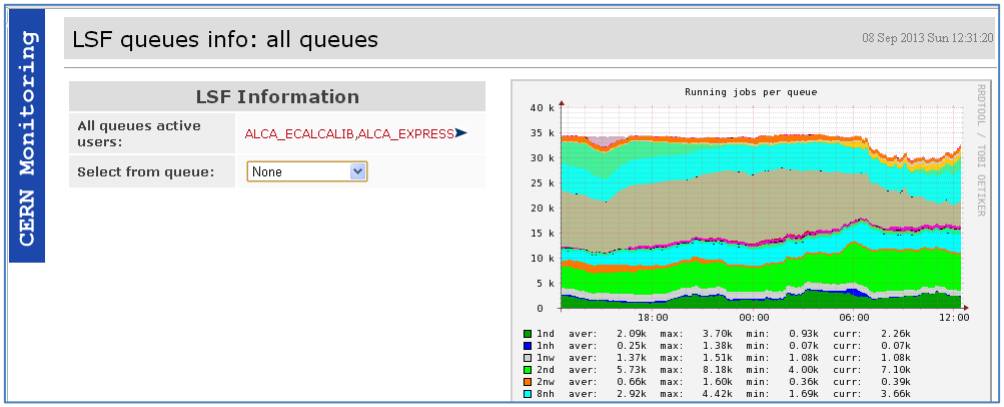


Figure 4. LSF Queues.

2.2 Technologies: What did I have to do?

OpenTSDB has been used as storage backend for my project. Being free software and fast for the time series data we have to handle, this was a sensible choice.

For the Application backend it was decided to use Django, a web application framework written in Python. It takes care of many web application-related security issues, including authentication and authorisation, which could be useful if we want to share the dashboards with people outside the section. The application backend communicates with OpenTSDB using the HTTP API. To query the HTTP API, we chose Request which is an elegant and easy to use HTTP library for Python.

Regarding the application frontend, we chose to use Ext JS, a JavaScript library which provides a very interesting set of widgets (plots, grids, layouts...) for building interactive web applications.

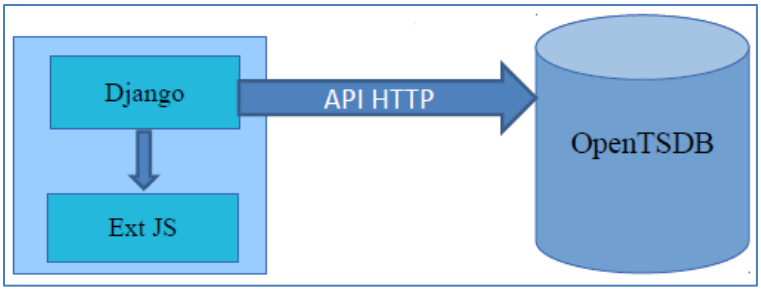


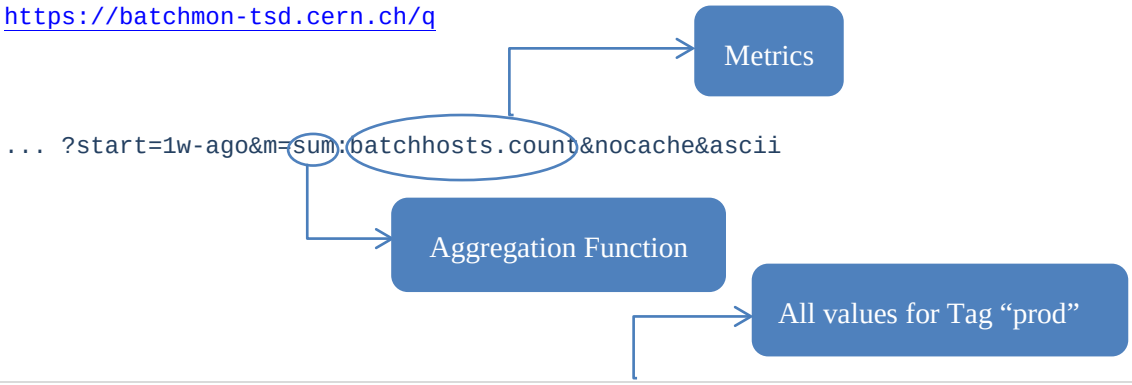
Figure 5. Application Architecture.

2.2.1 More about OpenTSDB's HTTP API

OpenTSDB provides an HTTP API to establish a communication with an application and get the desired data.The available HTTP end-points can return results in 3 different formats:

- The default format (either HTML or plain text).
- JSON, if the URL contains the *json* query string parameter.
- PNG, if the URL contains the *png* query string parameter.

Below you can see some queries used in the project and the explanation of their string parameters:



```
... ?start=1w-ago&m=sum:batchhosts.count{prod=*}&nocache&ascii
```

```
... ?start=1w-ago&m=sum:10m-avg:batchhosts.count&nocache&ascii
```

(*) Downsample the data points using a 10-minute average.

The application allows the following downsampling functions (*avg*, *dev*, *max*, *min*, and *sum*).

```
...?start=2013/07/01-12:00:00&end=2013/08/01-15:30:00&m=sum:10m-
avg:batchjobs.ended{queue=*}&nocache&ascii
```

`Start`	The query's start date.
`End`	The query's end date.
`M`	The query itself.
`Nocache`	Forces TSD to ignore cache and fetch results from Hbase.
`Ascii`	The plain text output format produced.

2.2.2 More about Django

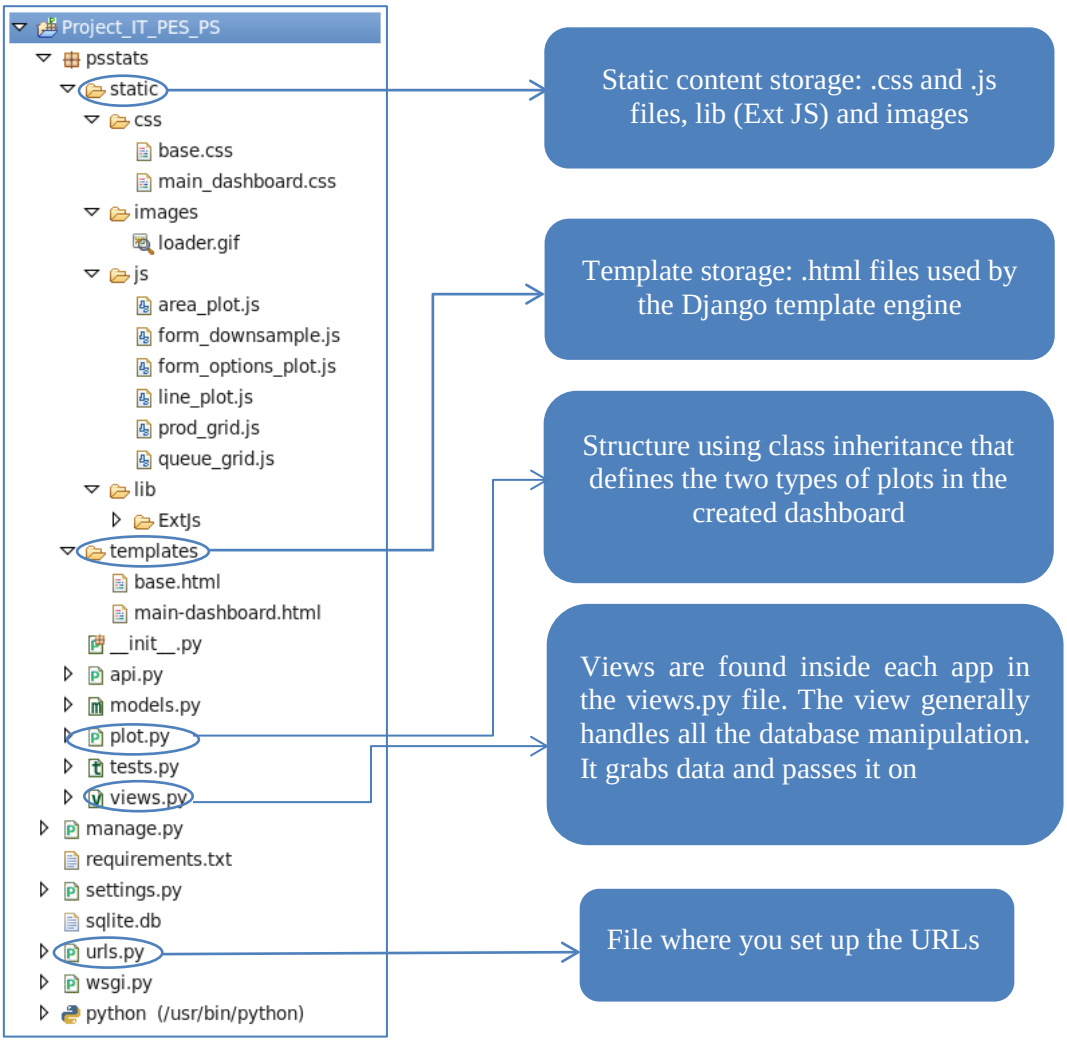
Django is an open source web application framework, written in Python. It follows the Model–View–Controller (MVC) architectural pattern. It is maintained by the Django Software Foundation (DSF). Django has a BSD License. This web framework was designed to simplify the creation of complex, database-driven websites. Python programming language is used throughout, even for settings, data models, etc.

2.2.3 More about Ext JS

Ext JS is a pure JavaScript application framework used for building interactive web applications using techniques such as Ajax, DHTML and DOM scripting. It is maintained by Sencha Inc. Ext JS has a GPLv3 License or commercial (if the project is not open source).

3 Project Structure

In the following diagram you can see the project structure:



3.1 Django flow

Django operates as follows:

- Users access a URL from a browser.
- Django matches the request against its urls.py file.
- If a match is found, Django moves on to the view that is associated with that URL. Views are generally found inside each app in the views.py file.
- A view is a Python method or class that generally handles all the database manipulation. It fetches data, performs (if required) some changes and passes data on to the templates.
- A template, specified in the view, renders the html page with some dynamic data.

3.2 Use of Class-based views

It was decided to use Class-based views in the views.py file, since you can use inheritance and the final solution is more elegant. At the same time the number of code lines is greatly reduced.

3.2.1 More about Class-based views

Class-based views provide an alternative solution to implement views as Python objects rather than functions. They do not replace function-based views; however some differences can be observed.

- Organization of code related to specific HTTP methods (GET, POST, etc.) can be addressed by separate methods instead of conditional branching.
- Object oriented techniques, such as multiple inheritances, can be used to factor code into reusable components.

3.2.2 Example of Class-based views

urls.py

```
from django.conf.urls import patterns
from psstats.views import Index
from psstats.views import BatchhostCount

...

urlpatterns = patterns('',

    (r'^$', Index.as_view()),

    (r'^api/batchhostscount/$', BatchhostCount.as_view()),

...

)
```

views.py

```
from django.shortcuts import render_to_response
from psstats.plot import LinePlot
from django.http.response import HttpResponse
from django.utils import simplejson
from django.views.generic.base import View


class Index(View):

    def get(self, request):

        return render_to_response('main-dashboard.html')
```

```
class BatchhostCount(View):

    def get(self, request):

        ...

        plot_data, grid_data = LinePlot(params, include_grid).get_data()

        return    HttpResponse(simplejson.dumps({'plot':    plot_data,    'grid':
grid_data}))
```

3.3 The Django template system

Django provides a template engine that manages .html files, in which there are special tags that belong to the Django template language.

Django's template language is designed to strike a balance between power and ease. It's designed to feel comfortable to those used to working with HTML.

3.3.1 More about the Django template language

A template is a text document, or a normal Python string, that is marked-up using the Django template language. A template can contain block tags or variables.

A block tag is a symbol within a template that does something.

This definition is deliberately vague. For example, a block tag can output content, serve as a control structure (an "if" statement or "for" loop), grab content from a database or enable access to other template tags.

Block tags are surrounded by "{%" and "%}".

A variable is a symbol within a template that outputs a value. Variable tags are surrounded by "{{" and "}}".

A context is a "variable name" -> "variable value" mapping that is passed to a template.

A template renders a context by replacing the variable holes with values from the context and executing all block tags.

3.3.2 Examples of templates

In this project there are two templates: base.html which contains basic information of any template that inherits from it, and main-dashboard.html which is the template for the created dashboard. This template inherits from the previous one.

3.4 Add a new plot in the dashboard

In order to add a new plot in the created single dashboard, follow the steps described below:

1. In urls.py file add the following:

```
from django.conf.urls import patterns
from psstats.views import Index
from psstats.views import NewPlot

...

urlpatterns = patterns('',

(r'^$', Index.as_view()),

(r'^api/newplot/$', NewPlot.as_view()),

...

)
```

2. In the views.py file add a Class-based view inheriting at least View (you can reuse the code using inheritance if the plot is one of the types created in this dashboard).

```
from django.http.response import HttpResponse
from django.views.generic.base import View

...

class NewPlot(View,...):

       def get(self, request):

       ...

       return HttpResponse(...)
```

3. Query DB from plot.py file.

4. Structure the results as required by Ext JS.

5. Place a JS function call in template main-dashboard.html.

6. Add the corresponding html code in the template main-dashboard.html.

7. Update main_dashboard .css file.

4 Look and feel of the website

The website has been designed with graphic elements provided by Ext JS. Besides it is navigable vertically and is located in the centre of the browser. It uses a width of 960 px as it is the average size used in websites.

The image below displays the first dashboard created with the new application. It contains generated plots for nine different metrics. As observed all the information is available at once without the need of querying different systems.

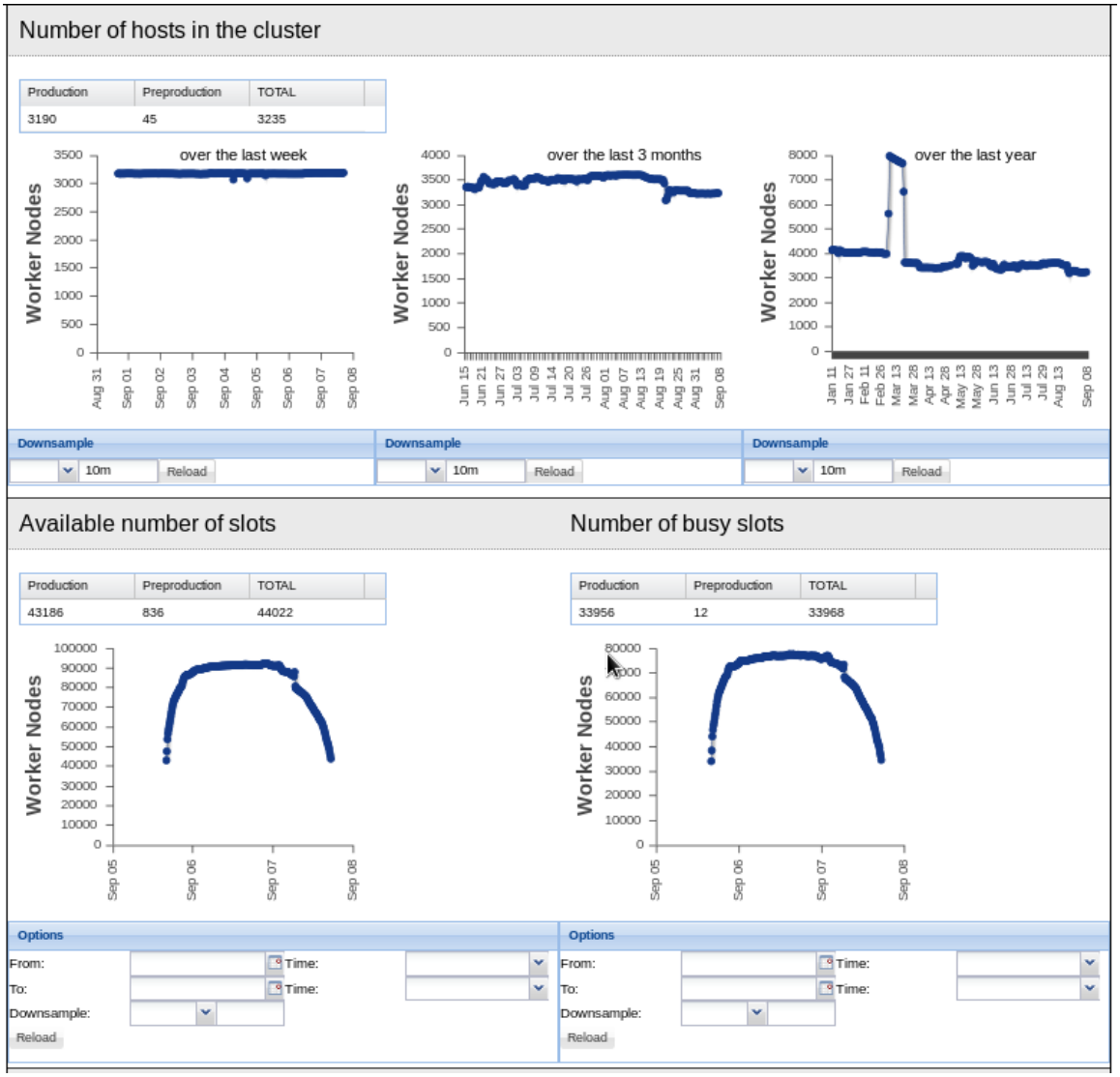


Figure 6. Line plots.

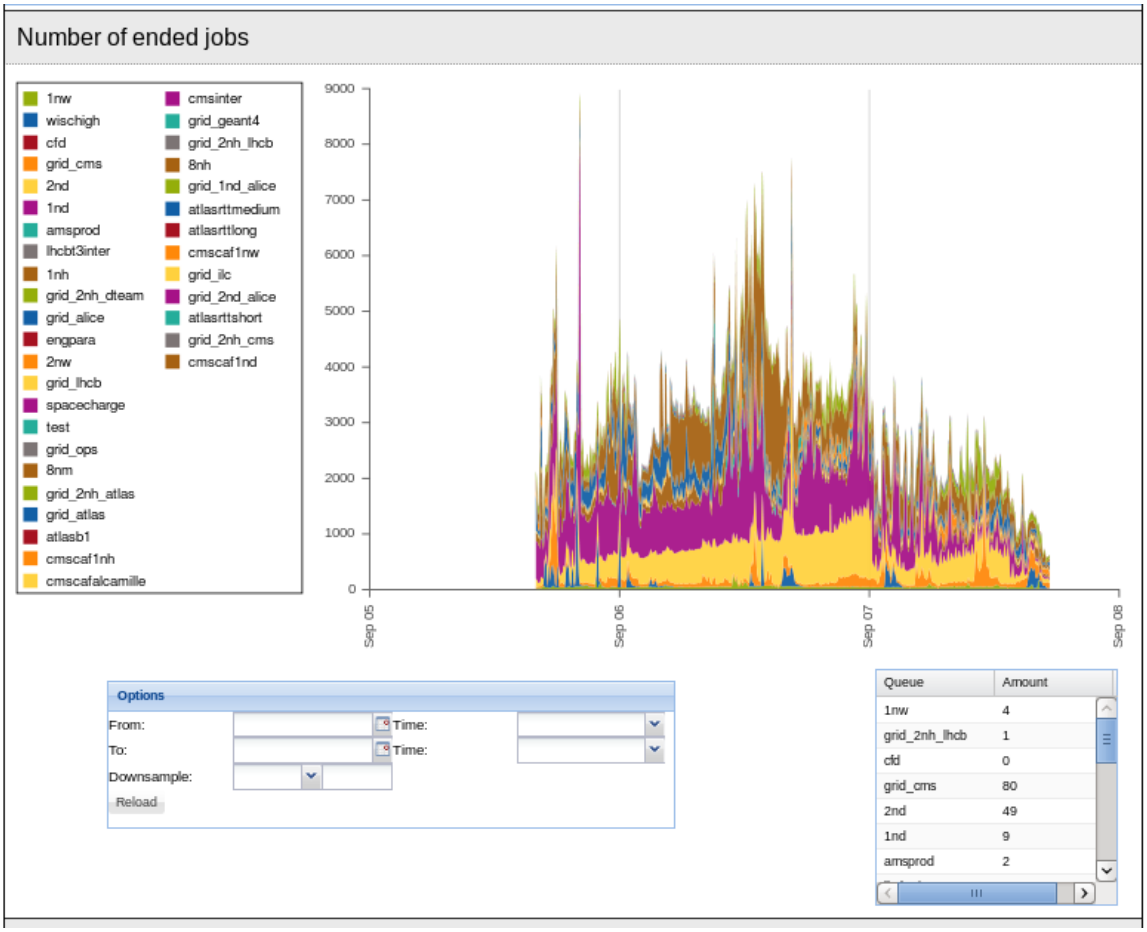


Figure 7. Area plot.

All plots and associated grids are loaded using AJAX. Initially the entire dashboard is loaded with default values.

When selecting options for a plot and pressing the reload button, only this plot is reloaded. The rest of the page remains the same. This is achieved using AJAX.

4.1 Problems during development

During the development of the application the following issues were found:

- Ext JS shows no errors while debugging the application code making it hard to find where the error comes from. In this particular case no solution was found. The only possible way to find the error was searching manually for it.

- The documentation provided by the Ext JS site is not very detailed and the number of code examples available is not very large.

- When the number of legends of a plot was high they were not properly displayed, not all the legends were visible in the screen. The solution was implemented using a column distribution of the legends.

- Another issue was the time needed to plot some charts. The browser sometimes freezes because Ext JS has to render too much data.

5 Software requirements

- Python (version 2.6.6 - used during development).
- Django (version 1.5.1 - used during development).
- Ext JS (version 4.2.1 - used during development).
- Request (version 1.2.3. - used during development).
- Single Sign On (SSO). You need to generate a cookie.

5.1 Generating an SSO Cookie

In order to generate a cookie follow the steps described below:

- Installation: Open a terminal and as root on your SLC5 or SLC6 system run:

  ```
  #yum install cern-get-sso-cookie
  ```

  A set of packages will be installed on your system.

- Usage: cern-get-sso-cookie acquires CERN Single Sign-On cookie using Kerberos credentials or user certificate and stores it in a file for later usage with tools like curl, wget or others - cookie file is stored in Netscape format understood by most web clients.

  In order to use this tool with Kerberos credentials a valid CERN Kerberos ticket must be acquired, for example using kinit. To check the validity of Kerberos credentials please use klist.

- During the development of the project, the cookie is provisionally stored in the directory `~/private/ssocookie.txt`

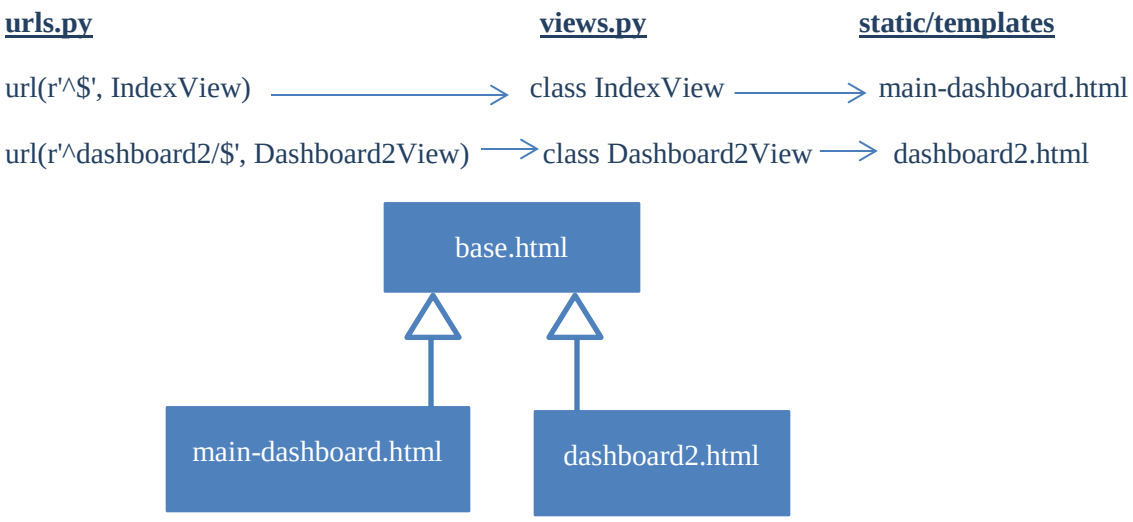
- Finally run in a terminal:

  ```
  #cern-get-sso-cookie -–nocertverify --krb -u https://batchmon-tsd.cern.ch
  –o ~/private/ssocookie.txt -r
  ```

6 Adding New Dashboards

The application was designed to allow adding new dashboards, and new plots to existing dashboards, in a very easy and quick way. In order to do so only a few steps are needed:

- In the base.html template you should add a tab system, a way to access the new dashboard you are about to create. HTML links or tabs are the most common examples of tab systems that can be used.
- Create a new HTML page that inherits from base.html and contains the new dashboard that you want to add.
- A new URL needs to be added to the url.py file.
- In the views.py file you should add a class or a view depending on the preferred implementation method.



7 Deployment

The web server used during the development phase was the one provided with Django. This server has some limitations as not being meant to serve static files and not having a load-balance module, etc. Therefore this server is not recommended for a production system, and a fully fledged is desirable. A good example to take into account would be the Apache server. More information about the interaction between Apache and Django can be found following reference number [8].

8 Conclusion

- Being able to participate in this project has brought me motivation because there was a real need for the development of the project. At the same time, I am pleased that the provided solution will simplify the daily work of everybody in the team.

- It was very glad to be part of a real project that will be used in a real work environment.

- I enjoyed learning new technologies, like Django and Ext JS and at the same time, I had the opportunity to get more relevant experience with Python.

9 References

[1] http://opentsdb.net/http-api.html

[2] http://en.wikipedia.org/wiki/Django_%28web_framework%29

[3] http://en.wikipedia.org/wiki/Ext_JS

[4] http://docs.python-requests.org/en/latest/user/quickstart/

[5] https://docs.djangoproject.com/en/dev/topics/class-based-views/

[6] https://docs.djangoproject.com/en/dev/ref/templates/api/

[7] http://linux.web.cern.ch/linux/docs/cernssocookie.shtml

[8] https://docs.djangoproject.com/en/1.1/howto/deployment/modwsgi/