Massimo Di Pierro
School of Computing
DePaul University
243 S Wabash Ave
Chicago, IL 60604 USA

web2py for Scientific Applications

Abstract

The needs of scientists to communicate effectively, collaborate over the internet, organize and present data in a structured and accessible manner have become critically important. web2py is a framework for rapid development and prototyping of secure database driven web applications that was developed specifically with the scientific and academic communities in mind. In this this article we provide a short introduction to this tool and use it, as an example, to create a web based application that stores DNA strings, looks for similarities, generates Needleman-Wunsh plots, and exposes XMLRPC web services. This article is based on a tutorial presented at SuperComputing 2009.

Introduction

When talking about scientific applications, we often refer to large scale numerical computations. In recent times, things have changed and the needs of the scientific community has broadened. In many sectors the needs of scientists to communicate effectively, increase access, organize and present data are more important that the need of writing fast code. For example, using scp/rsync to transfer data from one place to another was a good solution when a handful of scien-

tists were working on the data but it no longer a good solution when hundred of them are working together. Data has to be tagged, access has to be tracked, and errors have to be logged. Applications that access the data must be given a face in the form of an accessible web based user interface and/or a web service interface. Through the web interface, users can more easily and efficiently interact with the data. Grid tools provide a solution to some these problems but they have a steep learning curve, moreover, there is not one solution that fits all problems.

A relatively new class of tools that comes to the rescue of the scientists are web frameworks. These are libraries that allow rapid development of web based applications. Most of the frameworks provide API to handle concurrency and scalability (serve multiple users at the same time), security (authentication, authorization, XSS and Injection prevention), storage (cookies, sessions, database access via a database abstraction layer), and interface with third party programs via standard communication protocols. A web framework can turn a scientific software library into a complex interactive application. Web frameworks already find wide applications in science and they have been used to build web interfaces for data acquisition systems, computing clusters and computer algebra systems.

Many web frameworks have been developed by different communities in almost every programming language. The most common Open Source ones are probably Struts for Java, Symphony for PHP, Ruby on Rails for Ruby and Django for Python.

In this article we focus on web2py [1]. It is new web framework created in Python and programmable in Python. It is the only framework born in an academic environment; it is being used, for example, in projects supported by the US Department of Energy [2], and it counts more than 1700 registered users.

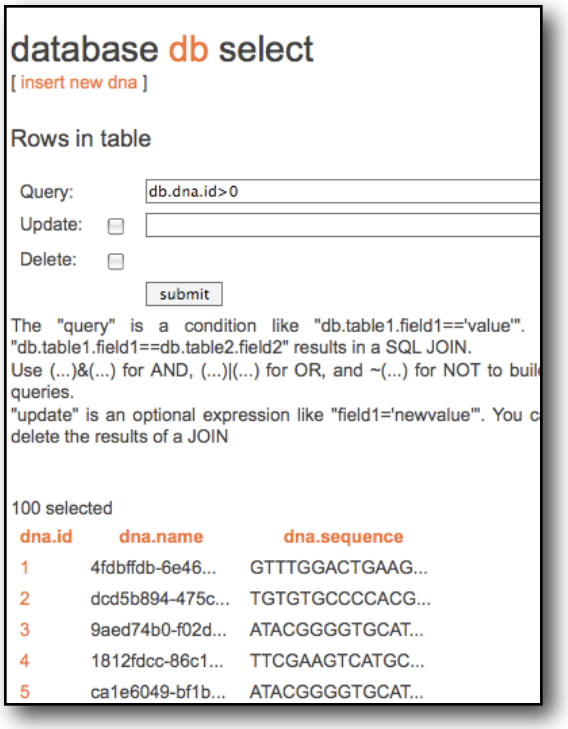
web2py differs from some of the other frameworks mentioned above for three reasons: 1) its main objective is not simply to provide a utility for web development, but it is to make it as easy as possible. This is achieved by forcing the developer to follow good practice (for example form self-submission), by relieving the developer from any decision that has security implications, and by providing everything in one package, including a web based IDE. 2) web2py is the only framework to provide a Database Abstraction Layer that works on both relational databases and non-relational databases (only Google App Engine at the moment but more are in the work). 3) web2py is written in Python and this allows to leverage on very power scientific libraries such such as NumPy, SciPy, matplotlib, PyTable/HDF5 and others.

In this paper will build, as an example, a web application that stores a database of DNA strings and looks for similarities using Needleman-Wunsh plots [3]. This article is based on a tutorial given at Supercomputing 2009 in Portland. The code we show here, consists of a complete web2py application.

Overview

web2py is a Model-View-Controller framework. It means that applications built with the framework are divided into files organized by their role. The three most important roles are: *Models* (files that describe how data should be represented in the system), *Views* (files that describe how data should be pre-

sented to users), and *Controllers* (files that describe the workflow of the application and run the core algorithms). web2py also handles other types of files: *modules* (regular Python extension modules), *static files* (for example images and other files that do not change), *sessions* (files that store that current state of each user of the system), *cache* (the short term memory), *databases* (the long term memory), *uploads* (like static files but they can be uploaded and replaced by users), *crontab* (files that describe tasks that need to be executed on a fixed schedule), and *languages* (files that contain translation of the application for internationalization support). Files are grouped under subfolders of the application folder. Web2py enforces the above structure and other good software engineering practices. It also provides a web based Integrated Development Environment (IDE) and web based testing and debugging tools.



Example Application

Our goal it to develop a web application that allows user to post and retrieve DNA strings, compare them using the Needleman-Wunsh algorithm[3], and plot the result. Users will have to be authenticated to use the system. For plotting we will use the matplotlib library.

We will call our application “dna”. A new empty application can easily be created via the web based IDE and it comes pre-populated with some scaffolding files which can be edited. We

will make it clear when we are editing a scaffolding file or creating a new file. Once web2py is installed and running the web based IDE can be accessed at URL http://127.0.0.1:8000/admin and a screenshot is reported in Fig.1.

The Model

We start by creating a new model file "models/db_dna.py" containing:

```
db.define_table('dna',
   Field('name', requires=(IS_NOT_EMPTY(),IS_NOT_IN_DB(db,'dna.name'))),
   Field('sequence', 'text', requires=IS_MATCH('[ATGC]*')))
```

The function "define_table" dynamically generates and executes the SQL code[1] to create the "dna" table (or alter the table in case it exists but has a different field content). The "dna" table has two fields "name" and "sequence" which have requirements (they will be enforced by the framework at the form level): name has to be non empty and not already in database, sequence has to contain only "ATGC". The expressions in upper case are called validators.

Just by typing this text in the model file, triggers web2py to create the table and generate a web based interface (*appadmin*) to perform inserts, updates, deletes and selects from the database. *appadmin* can be accessed at the URL http://127.0.0.1:8000/and/appadmin and a screenshot is in Fig.2 (we assume web2py is running on 17.0.0.1:8000).

The next step consists of generating some dummy data to populate the database. We will assuming each DNA sequence is comprised of 50 genes each picked at random from 10 possible different ones. Each gene is made of 20 random bases (ATGC). We will name each sequence

[1] For any of the supported back-ends: SQLite, PostgreSQL, MySQL, MSSQL, FireBird, Oracle, Informix, DB2, SyBase and the Google App Engine.

with a random universal unique identified (uuid). Here is the code to append to the "models/db_dna.py" file:

```
if not db(db.dna.id>0).count():
   import random, uuid
   genes = []
   for k in range(10):
       bases = ['ATGC'[random.randint(0,3)] for i in range(20)]
       genes.append(''.join(bases))
   for k in range(100):
       sequence = ''.join([genes[random.randint(0,9)] for i in range(50)])
       db.dna.insert(name=uuid.uuid4(), sequence=sequence)
```

The `if` statement check whether the table contains any record and, if not, the code inserts 100 new random DNA sequences in the table. Ideally, real data should be used to populate the database. Visit the *appadmin* interface to browse through the generated sequences (Fig.2).

The Algorithm

The core of our application is the Needleman-Wunsh algorithm listed below:

```
def needleman_wunsch(a,b,p=0.97):
   z=[]
   for i,r in enumerate(a):
       z.append([])
       for j,c in enumerate(b):
           if r==c: z[-1].append(z[i-1][j-1]+1 if i*j>0 else 1)
           else: z[-1].append(p*max(z[i-1][j] if i>0 else 0,
                                    z[i][j-1] if j>0 else 0))
   return z
```

We implement this algorithm in the "model/db_dna.py" file.

This algorithm takes two strings (`a` and `b`, in our case two DNA sequences) and returns a table `z`, where `z[i][j]` represent a similarity factor between the substrings `a[:i]` and `b[:j]`. This empirical algorithm works by defining the similarity between `a[:i]` and `b[:j]` as the similarity between `a[:i-1]` and `b[:j-1]` times `p=0.97`, if `a[i]==b[j]` or, otherwise, as the maximum of

the similarities between `a[:i],b[:j-1]` and `a[:i-1],b[:j]`. Our application will visualize `z` to allow users to locate similar genes in each couple of DNA sequences (Fig.3).

The Controller

The role of the controller is to implement the workflow of the application, decide which functions/algorithms to expose and call these algorithms. We edit the scaffolding controller "controllers/default.py" and replace the "index" function with this one:

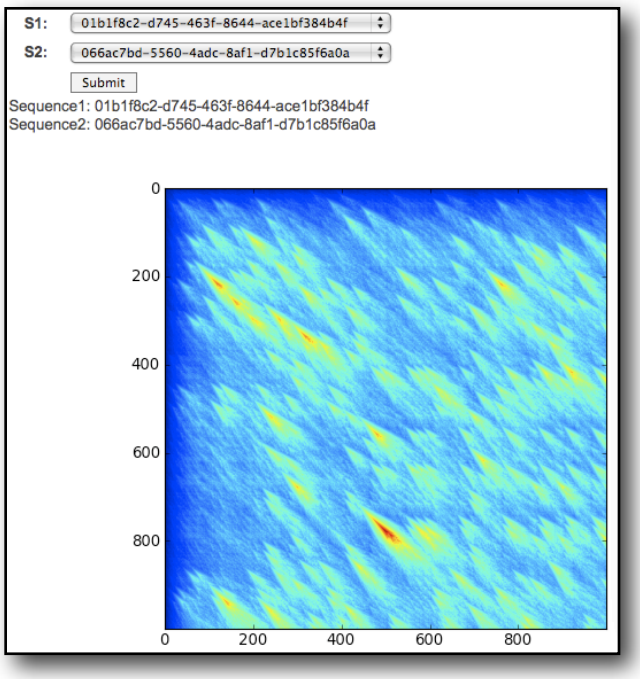
```
@auth.requires_login()
def index():
    form = SQLFORM.factory(
        Field('s1',db.dna,requires=IS_IN_DB(db,'dna.id','%(name)s')),
        Field('s2',db.dna,requires=IS_IN_DB(db,'dna.id','%(name)s')))
    if form.accepts(request.vars):
        image = URL(r=request,f='needleman_wunsch_plot',vars=form.vars)
    else:
        image = None
    return dict(form=form, image=image)
```

This functions defines a `form` which contains two fields (s1 and s2). Each of them is a reference to `db.dna` record. We require it to be represented by its `"%(name)s"` as opposed to its `"id"`. The function returns the form which is presented to the visitor. This function is called by visiting the URL

http://127.0.0.1:8000/dna/default/index (Fig.3). When the user picks two sequences and submits the form, the form is accepted (`if form.accepts(...)`) and it returns the variable `image` set to the dynamically generated URL calling another function (`needleman_wunsch_plot`) that generates the Needleman-Wunsh plot (we have not implemented this yet). The form variables s1 and s2 are also passed to that function via `vars=form.vars`.

The decorator (`@auth.requires_login`) guarantees that only logged users can visit the page. All other visitors will be required to go thought a login page and/or a registration page pro-

vided automatically by the framework (web2py implements Role Based Access Control mechanism with multiple authentication modules).



The View

web2py provides a default presentation for the output of controller functions but we can customize it in multiple ways. The easiest way is to provide a view for the "index" function by editing the file "views/default/index.html":

```
{{extend 'layout.html'}}
{{=form}}
{{if image:}}
Sequence1: {{=db.dna[request.vars.s1].name}}<br/>
Sequence2: {{=db.dna[request.vars.s2].name}}<br/>
<img src="{{=image}}" alt="loading..."/>
{{pass}}
```

In the view everything in `{{ ... }}` is pure Python code. Everything outside is HTML (because this is a .html view). Here we extended the default "layout.html" (a generic html files that ships with the scaffolding application with the role of giving all pages a consistent look and feel and that can also be customized. We embedded the form in the page `{{=form}}` and `{{if image:}}...{{pass}}` we embed the image pointing to the URL of the image.

Plotting

Finally, we implement the function "needleman_wunsch_plot" in "controllers/default.py". This function needs to: 1) retrieve the actual sequences from the database; 2) call the

Needleman-Wunsh algorithm; 3) print the data as plot on a canvas in memory; 4) print the canvas in a memory mapped PNG file (cStringIO); 5) return the PNG content. Here is the complete implementation:

```
@auth.requires_login()
def needleman_wunsch_plot():
    import cStringIO
    from matplotlib.backends.backend_agg import *
    from matplotlib.figure import *
    ### 1) fetch sequences
    dna1 = db.dna[request.vars.s1].sequence
    dna2 = db.dna[request.vars.s2].sequence
    ### 2) run algorithm
    z = needleman_wunsch(dna1,dna2)
    ### 3) make figure and plot on canvas
    fig = Figure()
    fig.add_subplot(111).imshow(z)
    ### 4) print canvas to memory mapped file
    stream = cStringIO.StringIO()
    FigureCanvasAgg(fig).print_png(stream)
    ### 5) return content of memory mapped file
    response.headers['Content-Type']='image/png'
    return stream.getvalue()
```

The matplotlib API are described in detail in Ref.[4]. The yellow and red pattens in the figure allow us to easily find couples of genes (identified by their X-Y coordinates) that contain similar DNA sub-sequences.

Web Services

At this point, our program is complete and fully functional, and so we want to add a web service. Specifically, we want to expose the ability to retrieve a DNA sequence from its uuid name via a XMLRPC service. This can be done very simply by placing the following code in "controllers/default.py":

```
@service.xmlrpc
def get_sequence(uuid):
    return db(db.dna.name==uuid).select().first().sequence
```

The decorator (`@service.xmlrpc`) does the job of exposing the function (the name of the function, `get_sequence`, is arbitrary). The service is available at the URL http://127.0.0.1:8000/dna/default/call/xmlrpc and it can be called from any programming language supporting XMLRPC. Notice how all database access is performed via the API of the Database Abstraction Layer, thus there is no need to write any SQL.

Linked Data

Linked Data [5] (LD) is a set of best practices for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and a Resource Description Framework (RDF). RDF is an XML based document file format. The idea of LD was proposed by Tim Berners-Lee (inventor of the World Wide Web) and consists of tagging database tables, fields and relations using a Web Ontology Language (OWL) and expose the data together with metadata that describe its meaning and relations. web2py provides support for LD by allowing to tag the data with OWD and automatically creating an RDF service. More information can be found in Ref. [6].

Conclusions

Web Frameworks provide a useful set of tools to help scientists expose their data and algorithms to the web, both for human consumption and for machine consumption. This can increase transparency, awareness and efficiency. Here we presented web2py, a web framework that we developed specifically with the scientific and academic communities in mind.

It is based on the Python language, it supports many protocols (a subset of which has been described here), and it can easily be integrated with other programs and libraries. web2py is released under the GPL2 license.

Acknowledgements

We thank Dr David Angulo for reviewing this manuscript and insightful conversations about the algorithm used here.

References

[1] http://web2py.com and M.DiPierro, “web2py Manual 2nd Ed”, Wiley (2009)

[2] M.DiPierro, “mc4qcd”, proceedings of the Advanced Computing and Analysis Techniques, Jaipur, India (2010)

[3] Needleman, Saul B.; and Wunsch, Christian D. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". *Journal of Molecular Biology* **48** (3): 443–53.

[4] http://matplotlib.sourceforge.net/

[5] http://linkeddata.org/

[6] http://web2py.com/semantic