PROJECT REPORT

'SIX DEGREES OF SEPARATION'

Aaron Fernandes (apf2114)
Amortya Ray (ar2566)
Joshua Poritz (jsp2104)
Ritika Virmani (rv2171)
Saahil Peerbhoy (sap2126)
Swapneel Sheth (sks2142)

Table of Contents

Introduction

For the project, we compared different programming languages and web application development frameworks.

In the first phase, we individually developed a CRUD (Create Read Update Delete) Application with exactly the same functionality and features. It is a Music Cataloging Application along the lines of a typical 3-tier (or MVC) web application with basic features like form validation, searching and sorting.

In the second phase, we compared these applications against some benchmarks which are explained below. The various tools used for benchmarking are also mentioned later.

To assist in the project we set up an SVN and a CVS repository on the CLIC Machines. We used BaseCamp for Project Management. Our project can be found at http://whimproject.projectpath.com

Languages and Platforms

Team Member	Language	Framework
Aaron Fernandes	PHP	Symfony
Amortya Ray	Python	Turbogears
Josh Poritz	Perl	Catalyst
Ritika Virmani	Python	Django
Saahil Peerbhoy	Java	Servlets
Swapneel Sheth	Ruby	Ruby on Rails

Architecture

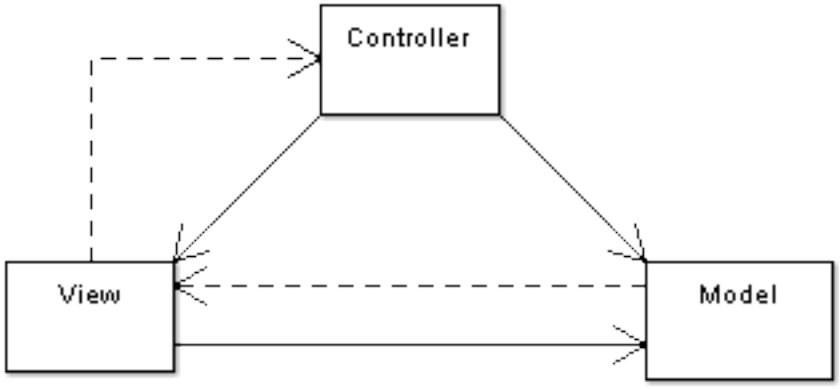
Model-View-Controller (MVC) is an architectural pattern used for developing software applications. The main advantage of using a MVC framework is that it separates the data from the user interface. The data is generally referred to as the 'model'. The user interface is called the 'view'. The interface between the view and the model is essentially a contract which is maintained by both sides of the interface. Normally, in any complicated software application, the data model and the UI are subject to changes over the course of development of the project. Changes on one side would often warrant a change in the other side. MVC frameworks eliminate the need for changing the other side by introducing a new layer of abstraction called the controller. Each side can be changed or reorganized without impacting the other side. As long as the interface is standard, changes to either side don't matter.

Model: The model represents the domain-specific representation of the information on which the application operates. Domain logic adds meaning to raw data (e.g., calculating if the current year is a leap year or the taxes applicable to a particular product).

Many applications use a database to store data. MVC does not specifically mention the data access layer because it is understood to be underneath or encapsulated by the Model.

View: The view renders the model into a form suitable for interaction, typically a user interface element.

Controller: The controller processes and responds to events, typically user actions, and may invoke changes on the model.



MVC in Web-based Applications

MVC is often seen in web-based applications, where the view is the actual HTML page and the code which gathers dynamic data and generates the content within the HTML is the controller. Finally the model is represented by the actual content, usually stored in some sort of a database or an XML file.

Though an MVC comes in different flavors, control flow generally works as follows:

1. The user visits the web page and interacts with it in some manner such as filling out a form and pressing the submit button.
2. A controller module handles the input from the user interface and processes it in some predefined way.
3. The controller may also access the model to retrieve or generate or update data.
4. The view uses the model to generate the appropriate user interface to return to the user.
5. The user interacts further with the application, and the cycle continues.

The messages passed between the various levels of abstraction for the application we have developed are explained below:

1. The user's interaction with the UI can be classified in the following types:
    a. Sort the displayed songs
    b. Search for a song
    c. Add a new song
    d. Edit/ delete an existing song
2. The view layer passes control to the controller and the controller executes the appropriate method to perform the action requested by the user. Depending on what the user wishes to do, the controller will either request data from the model (database) or send data to the model to

update the database.

3. The controller then formats the data appropriately and then sends the control to the view layer. The view layer reads the data and displays it to the user in the format specified by the programmer.
4. The cycle continues as the user continues interacting with the application.

A typical flow of messages is illustrated as follows:

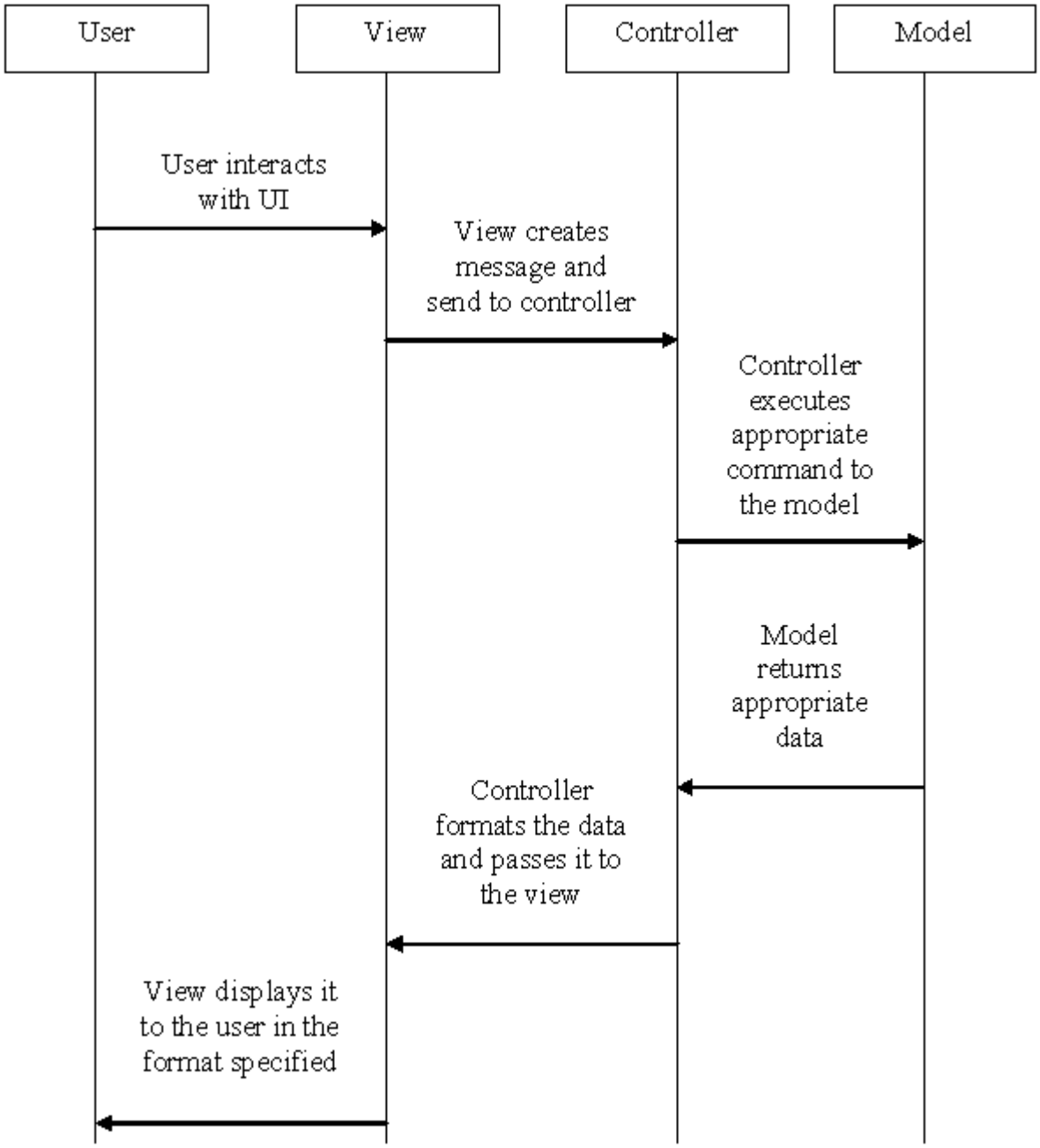


Illustration 1: Sequence Diagram

Database Schema

Field	Type	Null
Id	int(11)	NO
title	varchar(100)	NO
artist	varchar(100)	NO
album	varchar(100)	YES
year	int(11)	YES
genre	varchar(255)	YES
rating	int(11)	YES
length	time	YES

Components/Test Environment

1. Operating System – Ubuntu Edgy (Linux for human beings) 2.6.17-10 running on VMWare 1.0.2 on Windows XP
2. Web Server
   i. Aaron – Apache 2.0
   ii. Amortya – Turbogears built-in webserver
   iii. Josh – Catalyst built-in webserver
   iv. Ritika – Django built-in webserver
   v. Saahil – Apache Tomcat 5.5.9
   vi. Swapneel – Mongrel 1.0.1
   vii. Database – MySql 5.0.24a
3. The language/framework specific components are explained below

PHP 5.2.1 – Symfony 1.0.2

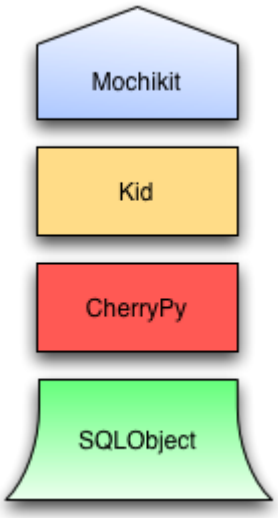
The underlying semantics of PHP provides just for the basics like setting and retrieving variables, working with loops, etc. The real punch behind PHP comes from the extensions, which are listed below:

1. **Core extensions** - These are bundled with PHP and are enabled by default. For all purposes, they can be considered part of PHP itself as they are available inside PHP unless explicitly disabled. Features like opening, reading and saving files, etc. are handled by the core extensions.
2. **Bundled extensions** – These extensions are bundled with PHP but not enabled by default. These features graphics creation and editing, etc.

3. **PECL** – PECL stands for "PHP Extension Code Library", which contain extensions that are rarely used, or those bundled extensions that are no longer consider ed relevant.
4. **Third Party Extensions** – These include extension that are built by other users.

For the purposes of this project, only PECL was used which interacted with the database.

Python 2.4– Turbogears 1.0.1



Turbogears uses the following components that allow us to define the various pieces and join them together and make them work seamlessly together.

1. Mochikit is an extremely powerful Javascript library.
2. Kid is the designer and templating system.
3. CherryPy makes doing web input/output as easy as writing a Python function.
4. SqlObject lets us access the database as you would normal Python classes, without obscuring the database itself.

Perl 5.8.8– Catalyst 5.7006 version

Catalyst is a Perl MVC (Model, View, Controller) framework that operates on three core principles:

1. **Don't Repeat Yourself** - Definitions only need be made once. Catalyst allows easy separation of concerns between the content,presentation, and flow control layers, hence eliminating the necessity for repetition of application logic. Catalyst, further,automatically discovers and loads all components (no need for use statements). Catalyst enables re-use of components across divergent applications.
2. **Flexibility** - Catalyst permits use of as many Models,Views, and Controllers as the programmer wishes within the same application; for example, she can easily present data from the same model using Template Toolkit and PDF::Template. Catalyst's model for dispatching URLs employs not class/method names but direct addressing from the client, hence allowing invocation of actions through regular expression matching. Finally, any web server with CGI or FastCGI will jive with Catalyst, though Apache or lighttpd with FastCGI or mod_perl support is recommended. We use Apache/mod_perl in this project.
3. **Simplicity** - Catalyst contains pre-built components for popular models, a global Content object for request access and data sharing between components, and a built-in web server for easy unit testing.

Catalyst applications comprise three types of components:

1. **Models** provide data. Models typically represent a database table, although his data could come from anywhere (search engine index, spreadsheet, file system, etc.). Catalyst alleviates the need for coding to operate the database side by providing the option to load all table layouts/relationships into a static schema through DBIx::Class::Schema::Loader; the programmer may, however, tweak the low-level SQL statements if needed. Other potential classes for handling the Model include Plucene and Net::LDAP.

2. **Views** present data to the end-user. Catalyst can define and interface them with Template Toolkit, Mason, or HTML::Template, among other modules.
3. **Controllers** dictate application flow. Employing multiple controllers is a suitable way to separate logical domains of an application. Large chunks of Controller functionality can usually be deferred to one of the many Catalyst PlugIns (e.g., Catalyst::Plugin::FormValidator, Catalyst::Plugin::Prototype, Catalyst::Plugin::Account::AutoDiscovery, etc.).

Python 2.4 – Django 0.96

Django based on Python (not the kind that wriggles), adopts the standard MVC (Model-View-Controller) design pattern, but instead the acronym is MTV (Model-Template-View). Essentially, the code is broken up into three main components.

The **model** is an object relational mapping to the database schema. So each model is a class which represents a table in the database. This abstracts the underlying database management system and also generates the SQL statements based on the methods you call on your model. This is attractive to those who are afraid of touching SQL as virtually little to no SQL knowledge is required by using Django's model API. However you can still mess around with SQL if you want to.

The **template** is simply the HTML for your view. In addition to HTML, Django's template system allows you to use simple constructs for template-specific logic. For instance, you could display different messages depending on whether or not a user is logged in.

The **view** can be thought of as a page in your application. It could be the homepage or any other user specific page. This is where you write your code for functionalities such as sorting the data returned from the database.

The idea behind the framework is rapid web development. It has a lot of helper functions so that the developer can concentrate on application specific tasks rather than mundane functionalities inherent in every web-based application. One of the most useful features is the "admin" site which allows you the CRUD functionalities required for every web application with a database back end.

Java 1.5 - Servlets

The implementation has three tiers - the front end which is HTML, the middle tier which are servlets and the back end which is the database server. The flow is as follows: The user interacts with the front end(i.e HTML pages). When any processing needs to be performed, the data is sent to the appropriate servlets using GET or POST methods. The servlets (java code) then access the database using JDBC and format the results returned from the database server (MySQL) in the form of HTML pages and return the to the client. Thus the servlets act as the controllers. Mapping a request to the appropriate servlet is performed by the web server (Apache Tomcat). Servlets are essentially Java code that use the Servlet API, contained in the Java package hierarchy javax.servlet. This API defines the expected interactions of a Web container and a servlet. A Web container is essentially the component of a Web server that interacts with the servlets. The Web container is responsible for managing the life cycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

Advantages of servlets:

- Portable : Servlets are written in Java and hence are highly portable. The same servlet can be

run on virtually any server.

- Secure : Servlets are executed within the confines of the JVM and are thus inherently safe.
- Convenient : Since it is written in Java, there are many packages available which can be used. For example, for more complex applications, a tool like Lucene which build indexes can be used.

Ruby 1.8.4 – Ruby on Rails 1.2.3

Ruby on Rails is an MVC Framework for Ruby. Some of the components it uses are:

1. **Model** - Ruby on Rails uses ActiveRecord at the Model Layer. ActiveRecord is an Object – Relational Mapping Layer which closely follows the ORM Model – tables map to classes, rows to objects and columns to object attributes. ActiveRecord is also used for Transactions, Validations, etc.
2. **View** - Ruby on Rails uses ActiveView at the View Layer. ActiveView encapsulates all the functionality needed to render templates, mostly HTML and XML, to the user. It has support for templates, pagination, helpers (formatting helpers, form helpers, etc.), layouts, components, etc. Further, Ruby code can be embedded in the templates using RHTML (similar to JSPs).
3. **Controller** - Ruby on Rails uses ActionController at the Controller Layer. ActionController is used for Routing Requests (mapping an incoming request to a particular controller and action), Cookies and Sessions, Filters, Verification, Flash (used for communication between actions), Caching, etc.

Benchmarks – Parameters and Tools

Parameter	Tool

Lines
● Lines of Code
  Model
  View
  Controller
● Number of methods
	N/A

Request Per Second	Apache Benchmark
Time Per Request	Apache Benchmark
Throughput	Siege
Response Time	Siege
Transaction Rate	Siege
Memory Usage	FunkLoad
CPU Usage (Load average)	FunkLoad
Page Response Time	FunkLoad

Demo Plan

The demo contained the following:

1. A description of what was done by each member – 2 minutes
2. Look and feel of our application – 2 minutes
3. Mentioned/Showed one 'killer' feature of our respective applications – 2 minutes
4. Ran some benchmarking tests to give a feel of how we did them – 1 minute
5. Illustrate and describe the various benchmark results and their impact – 5 minutes

Benchmark Results

Lines of Code

Frameworks	Symfony	Turbogears	Catalyst	Django	Java Servlets	Ruby on Rails
Model	11	21	4	14	260	11
View	80	354	88	41		112
Controller	97	795	188	12		111

No. of Methods

Frameworks	Symfony	Turbogears	Catalyst	Django	Java Servlets	Ruby on Rails
No Of Methods	9	23	15	3	4	14

Performance Benchmarks: Tools & Results

Apache Benchmark

`ab` is a simple tool, available from Apache's website, designed to perform rudimentary load benchmarking of an HTTP server. `ab` can provide an impression of how a server performs under varying loads, measuring how many requests per second (of a provided URL) the server is capable of fulfilling. Each load test output has the following form:

```
Server Hostname:        localhost
Sever Port:             4000

Document Path:          /songs/list
Document Length:        103372

Concurrency Level:      10
```

```
Time taken for tests:   403.19551 seconds
Complete requests:      1800
Failed requests:        0
Write errors:           0
Total transferred:      186379200 bytes
HTML transferred:       186069600 bytes
Requests per second:    4.47 [#/sec] (mean)
Time per request:       2238.998 [ms] (mean)
```

Tests Performed

We launched two tests configurations against each framework:

1. 3000 requests spread across 10 concurrent users
2. 7500 requests spread across 25 concurrent users

The throughputs of the six frameworks ranked the same across both configurations:

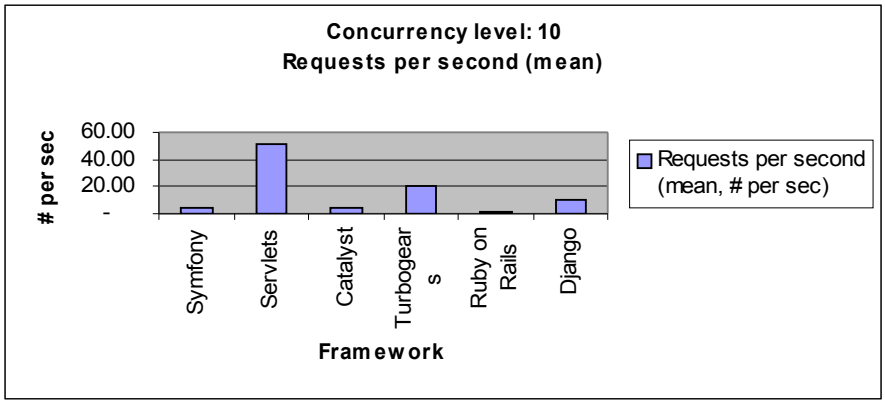
1. Java Servlets (best)
2. Turbogears
3. Django
4. Catalyst*
5. Symfony
6. Ruby on Rails (worst)

* = Due to a memory leak in this framework, we could only perform 1500 requests before all 256 MB of physical memory on the machine were maxed out.

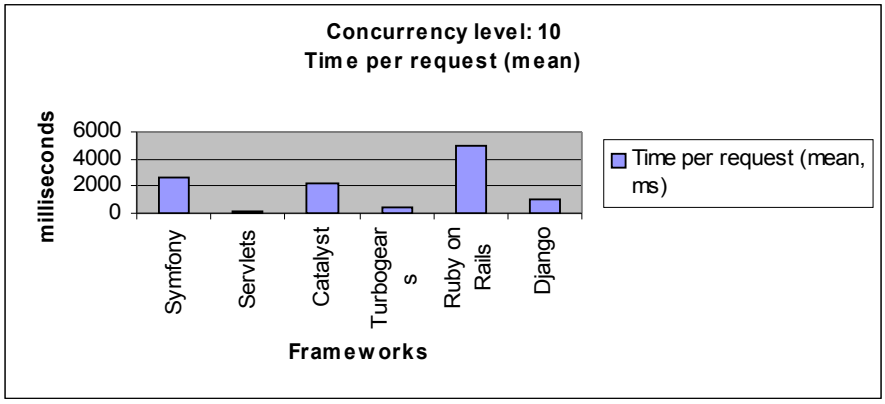
Results & Analysis

Below are the numerical results from test configuration 1:

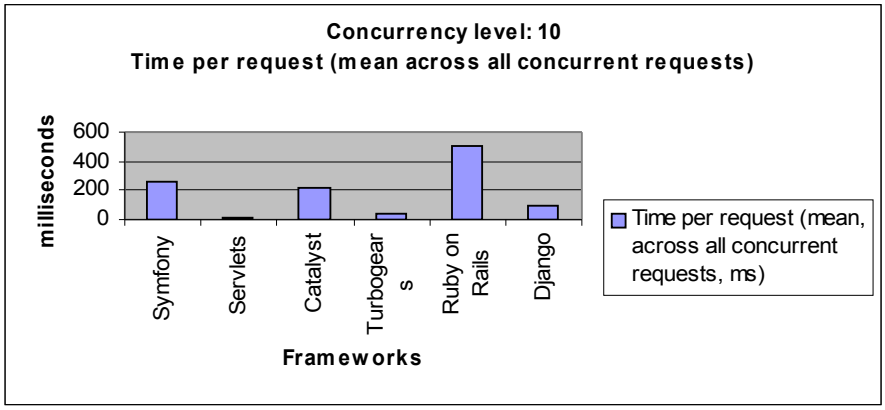
Requests per Second:



Time per Request:



Time per Request (mean across all concurrent requests:



As one can observe, the throughput rates of the frameworks diverged by up to an entire order of magnitude. Although one might expect Java bytecode to run more speedily than programs written in a scripted language, pure execution time cannot possibly account for such a broad gap; rather, it must be the case that certain frameworks perform special optimizations, such as caching of database query results or HTML responses, in anticipation of substantial load consisting repeatedly of the same request. It might also be useful to examine process management of each framework (i.e. does it employ processes, threads, or multiplexing to achieve concurrency?) and each programmers' implementation of the View layer in his/her respective system. Populating a Template Toolkit file with data extracted from a MySQL DB, for example, might require substantially more time than simply generating an HTML page listing songs on-the-fly due to the extra TT directive parsing step necessitated by the former model.

Siege

Siege is an http regression testing and benchmarking utility. It was designed to let web developers measure the performance of their code under duress, to see how it will stand up to load on the Internet. It lets the user hit a web server with a configurable number of

concurrent simulated users.

In effect, we place the web server under siege. The siege is measured in terms of number of transactions which is the product of the number of concurrent users and the number of times each simulated user hits the server.

Siege was written on GNU/Linux. It relies on POSIX1.b, a feature not supported by Microsoft. Thus we could not run Siege on our own local Windows machines.

We ran the test with the following configuration:

Number of Repetitions : 300
Concurrent Users : 10

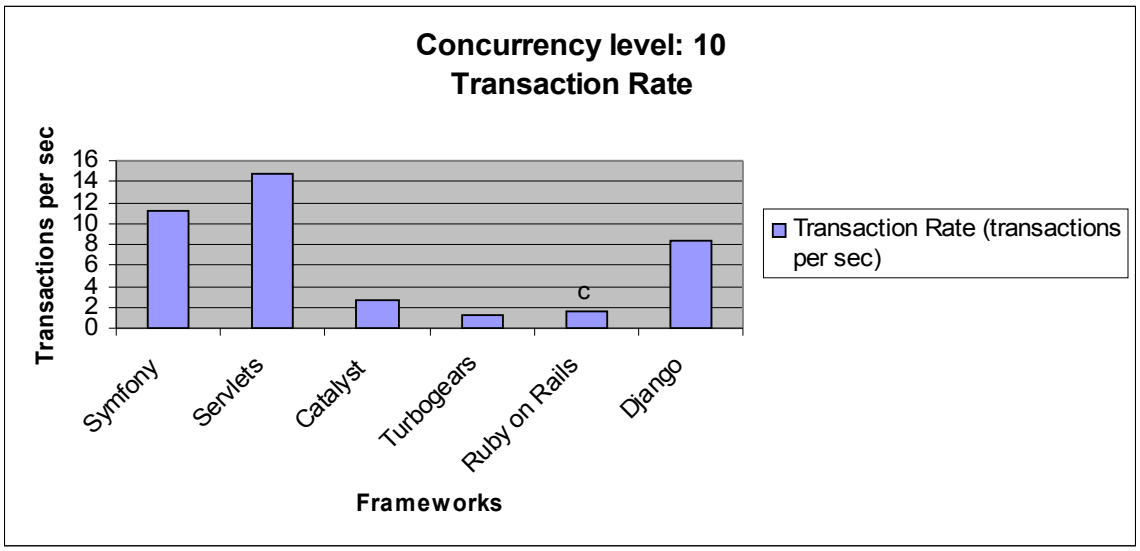
Results:

Framework	Transaction Rate (transactions per sec)	Throughput (MB per sec)	Response Time (sec)
Symfony	11.27	0.03	0.31
Servlets	14.74	0.63	0.14
Catalyst	2.67	0.26	3.23
Turbogears	1.19	0.14	7.88
Ruby on Rails	1.65	0.25	5.55
Django	8.27	0.35	0.64

Graphs:

Transaction Rate:

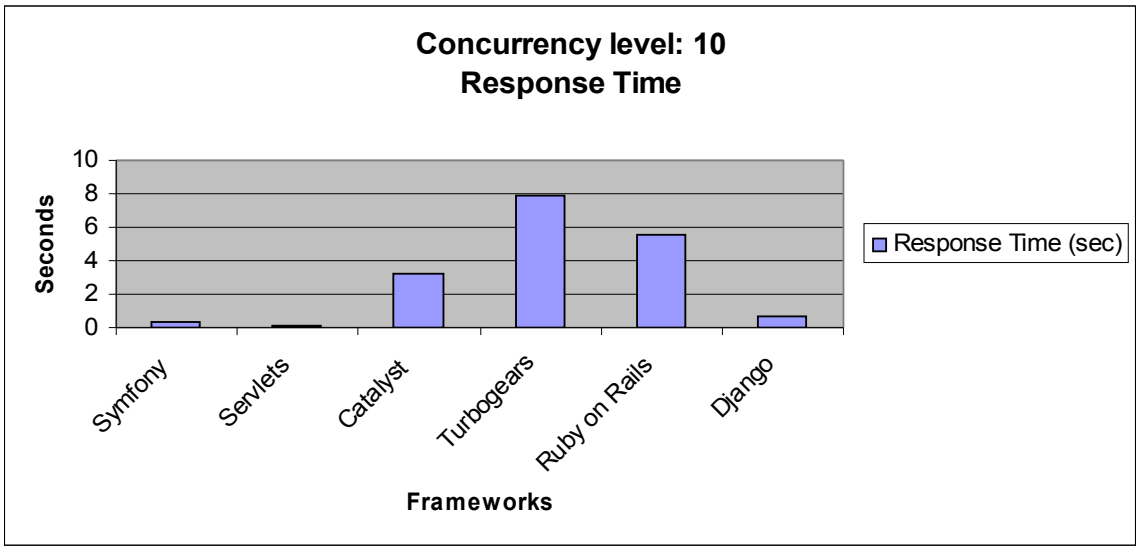
Java servlets perform the best. This is because they are not bogged down by a framework. Symfony and Django also show pretty good results with Turbogears bringing up the rear in last position.



Response Time:

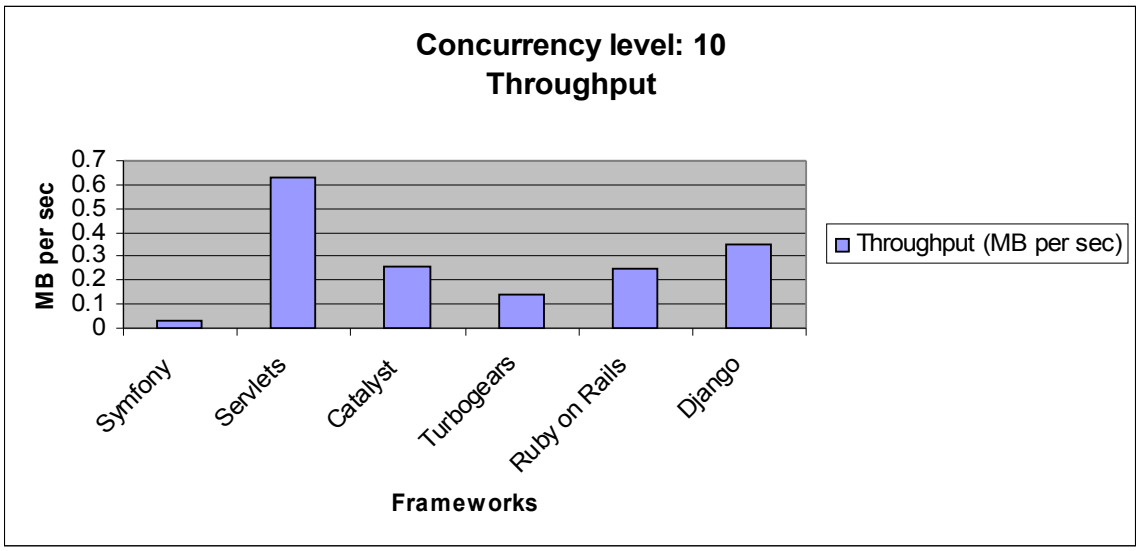
This is the time the server takes to respond to a request.

Once again, Java servlets perform the best. Symfony and Django again do well along with Catalyst, with Turbogears performing the worst again.



Throughput:

Throughput is the average number of bytes transferred every second from the server to all the simulated users. As expected, servlets once again perform the best. Django again does well. However, this time around it is Symfony that brings up the rear.



Funkload

Funkload is a functional and load Web tester written in Python. Its main use cases enable functional testing of Web projects, performance testing, stress testing and load testing. Funkload performs performance testing by loading the web application and monitoring the servers. This allows it to pinpoint bottlenecks and it then gives a detailed report of the performance. Funkload is a free software distributed under the GNU GPL. All the functional tests that are a part of Funkload are pure Python scripts using the pyUnit framework. Funkload outputs its bench reports in HTML containing the following:

- The bench configuration
- Tests, pages, requests stats and charts.
- The 5 slowest requests.
- Servers CPU usage, load average, memory/swap usage and network traffic charts.
- An http error summary list

Tests performed

We performed several tests using Funkload of which only the load average and the memory usage are of concern to us.

Results

Load Average

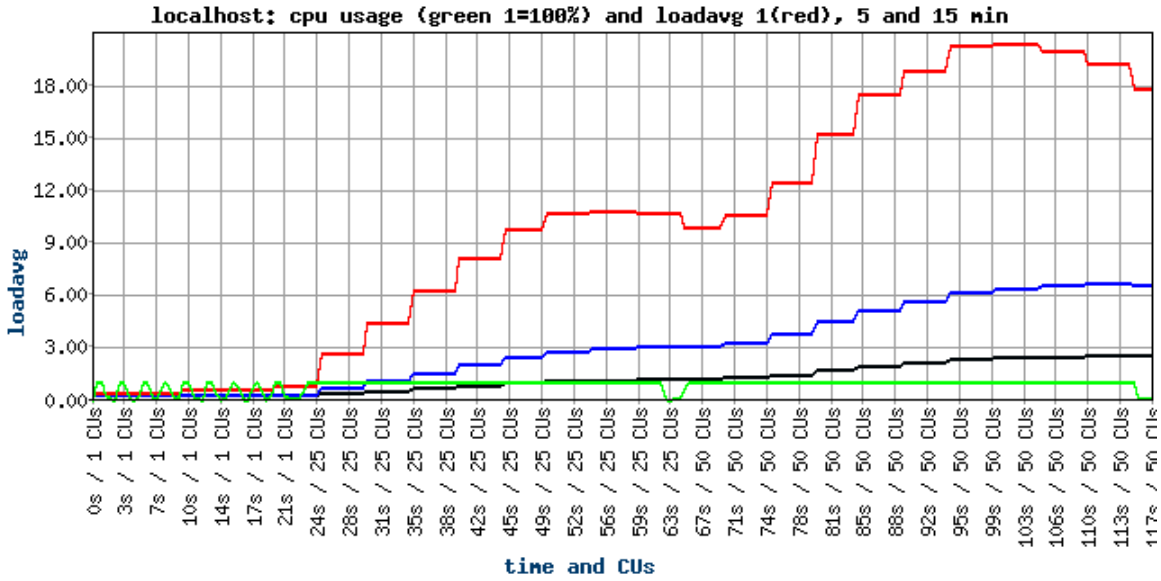


Illustration 2: Symfony

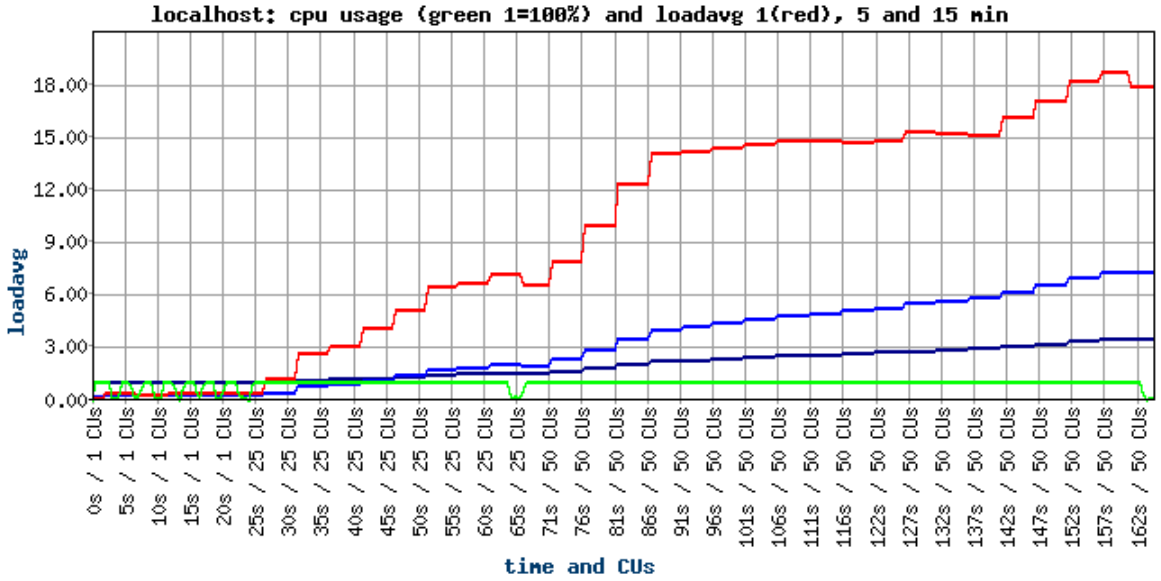


Illustration 3: Turbogears

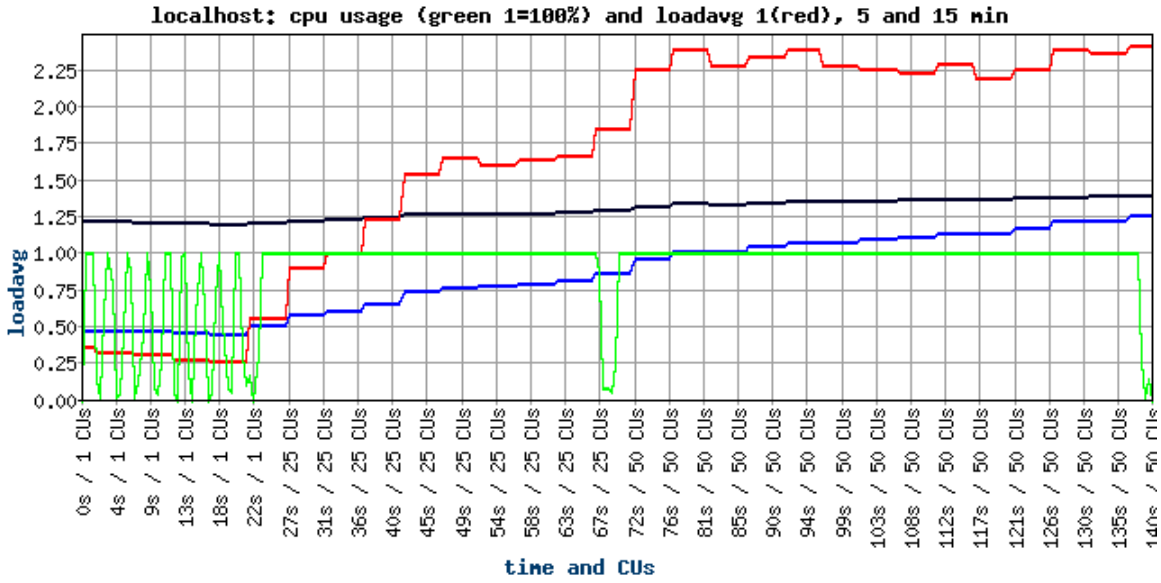


Illustration 4: Catalyst

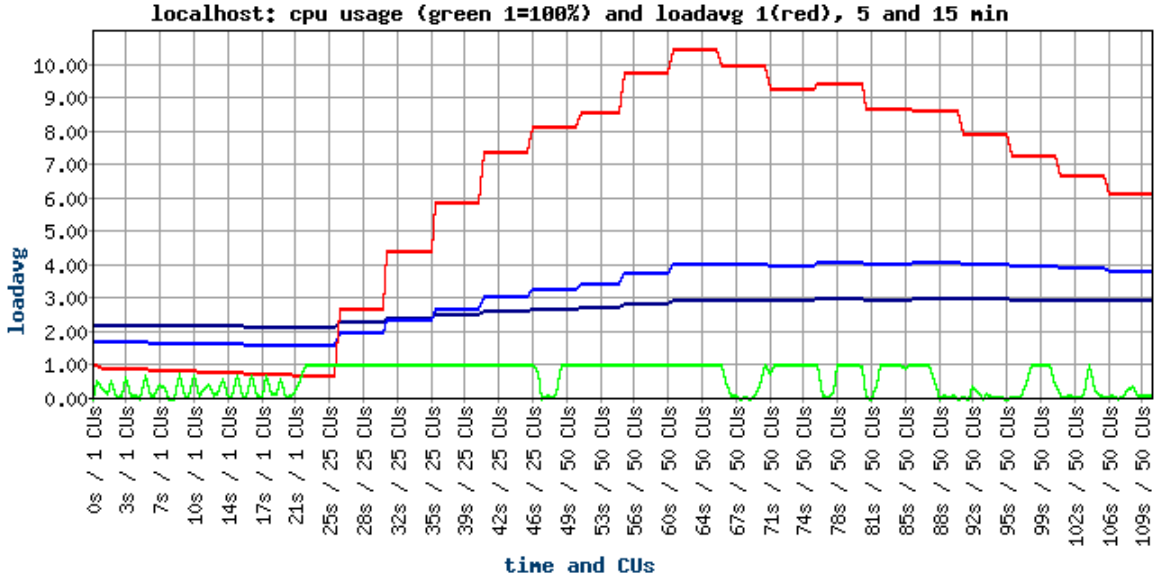


Illustration 5: Django

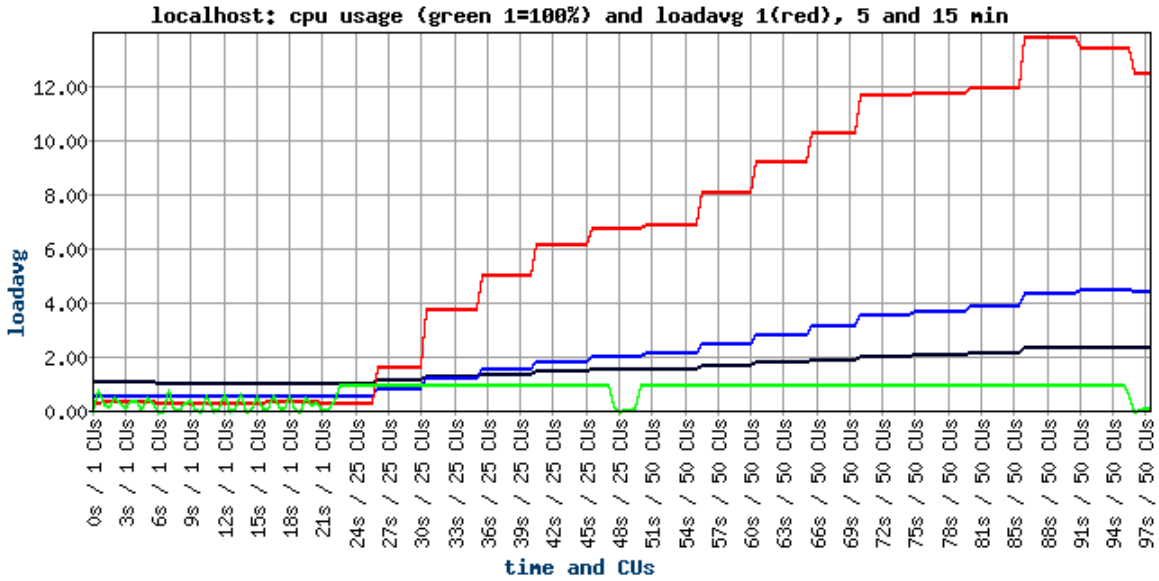


Illustration 6: Java Servlets

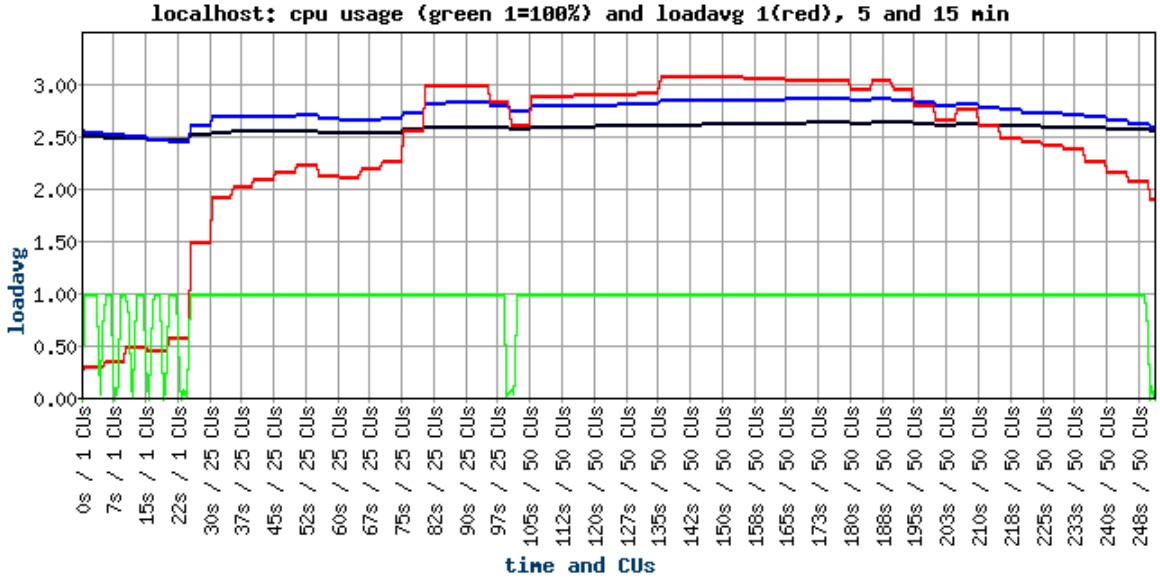


Illustration 7: Ruby on Rails

Analysis

The tests were run inside a virtual machine running Ubuntu Linux. Also, since all the servers were up and running at the time of running the test script, the CPU was slightly more loaded.

The ideal CPU usage would be represented by a near-flat usage shown by the red line the graph. However, after inspecting the graphs, we can see that Symphony, Turbogears and Servlets have a steadily increasing CPU usage. Catalyst also has an increasing graph but it is less steep than the ones previously mentioned. However, Ruby on Rails seems to have the best CPU usage. The curve is more or less steady and the CPU usage remains constant and close to 3.00 once it reaches the peak level.

Memory Usage

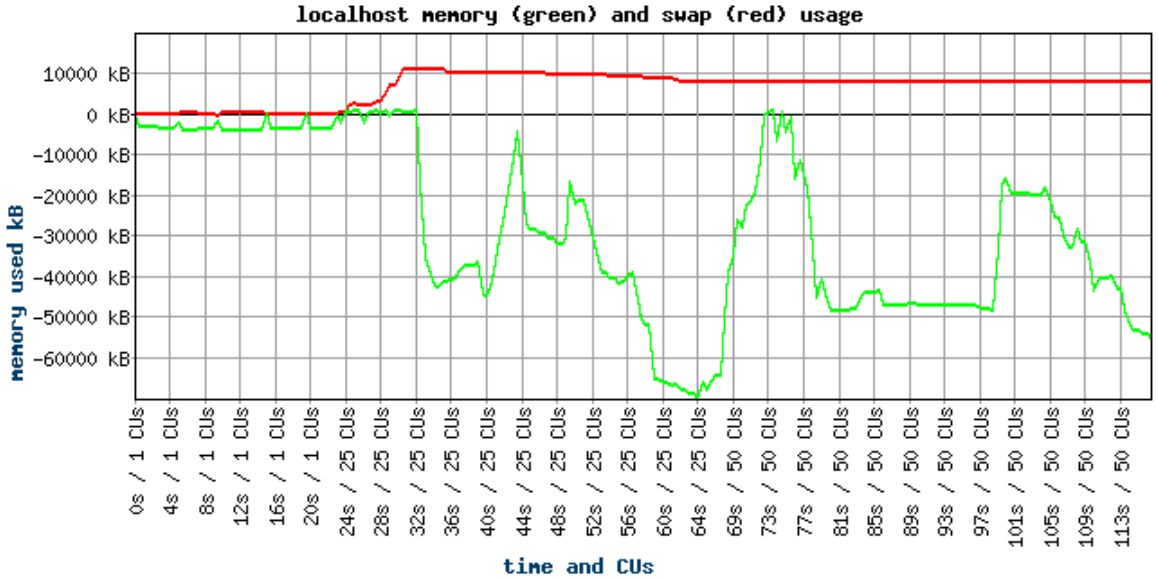


Illustration 8: Symfony

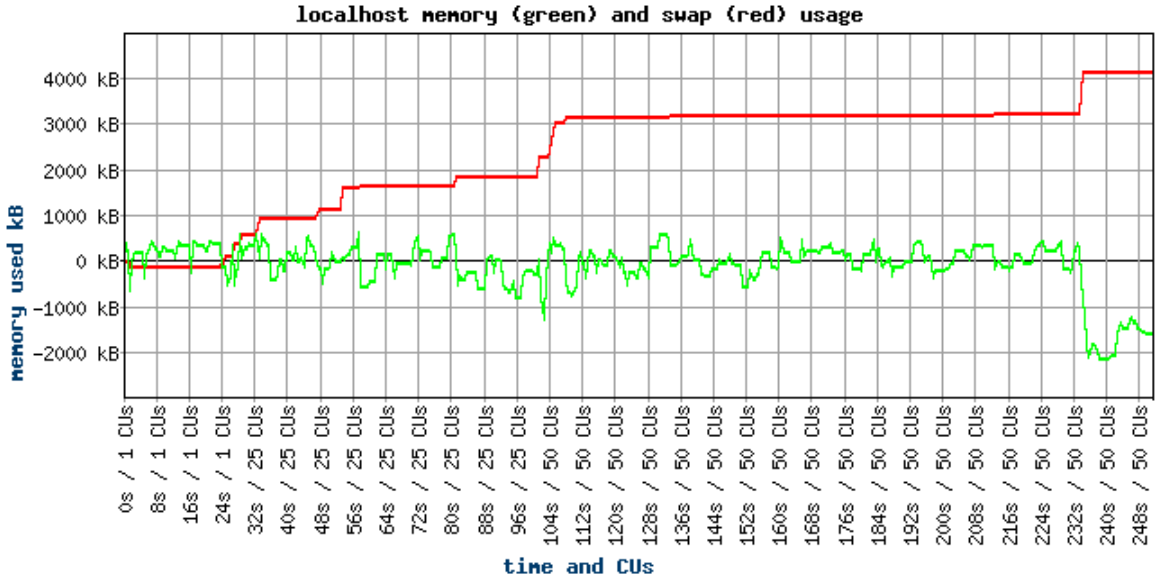


Illustration 9: Turbogears

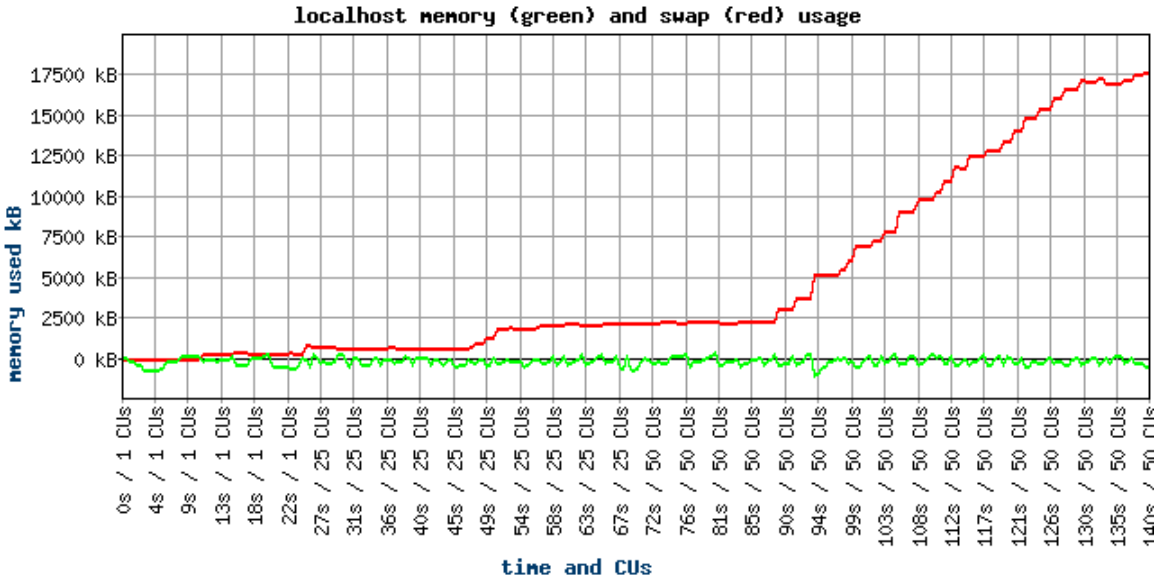


Illustration 10: Catalyst

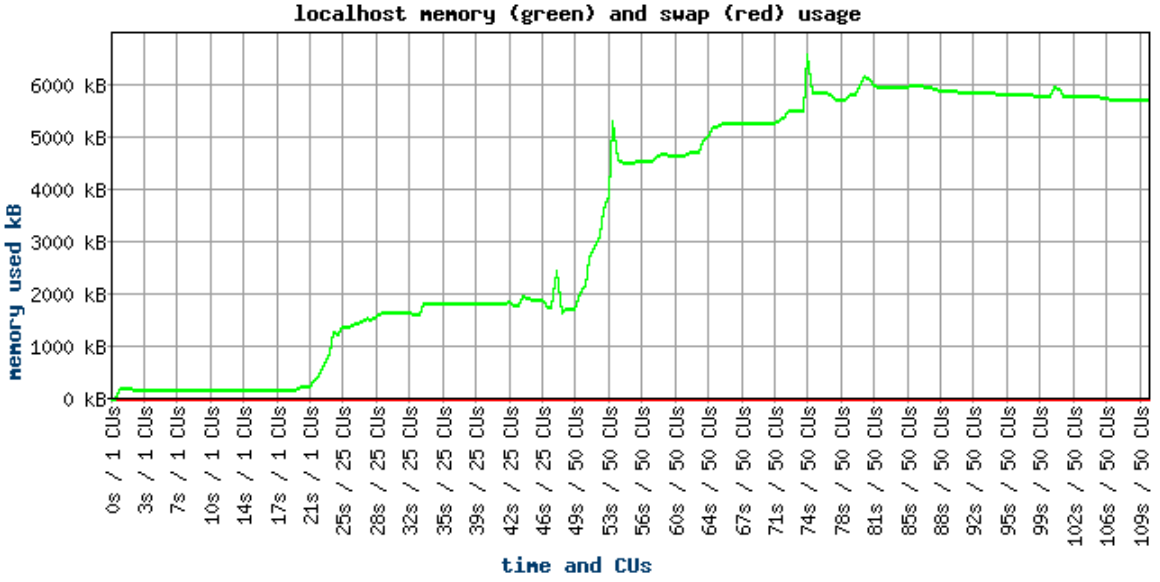


Illustration 11: Django

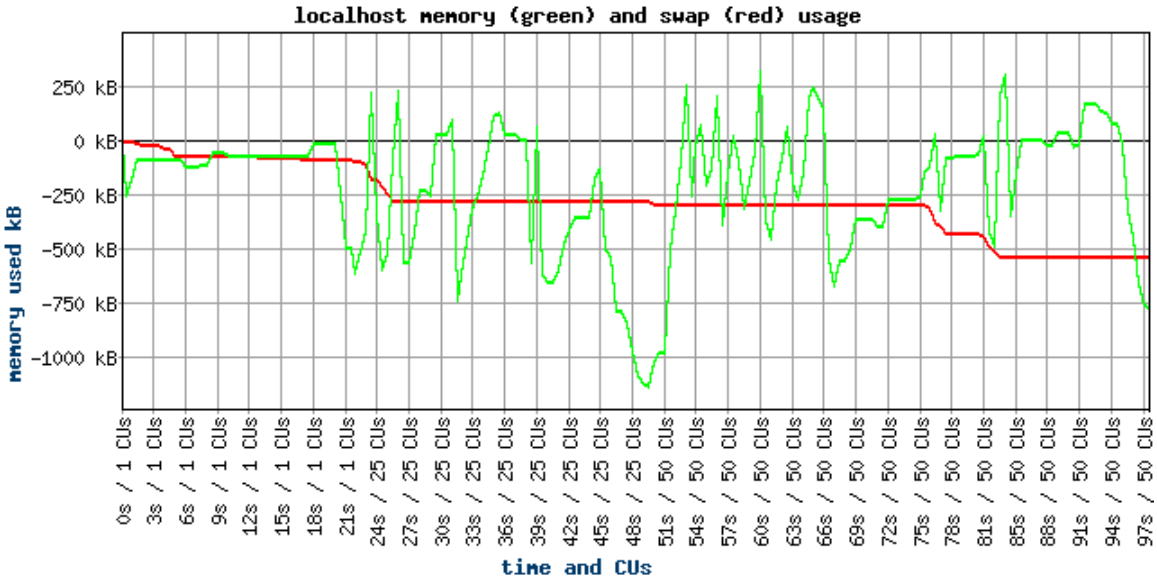


Illustration 12: Servlets

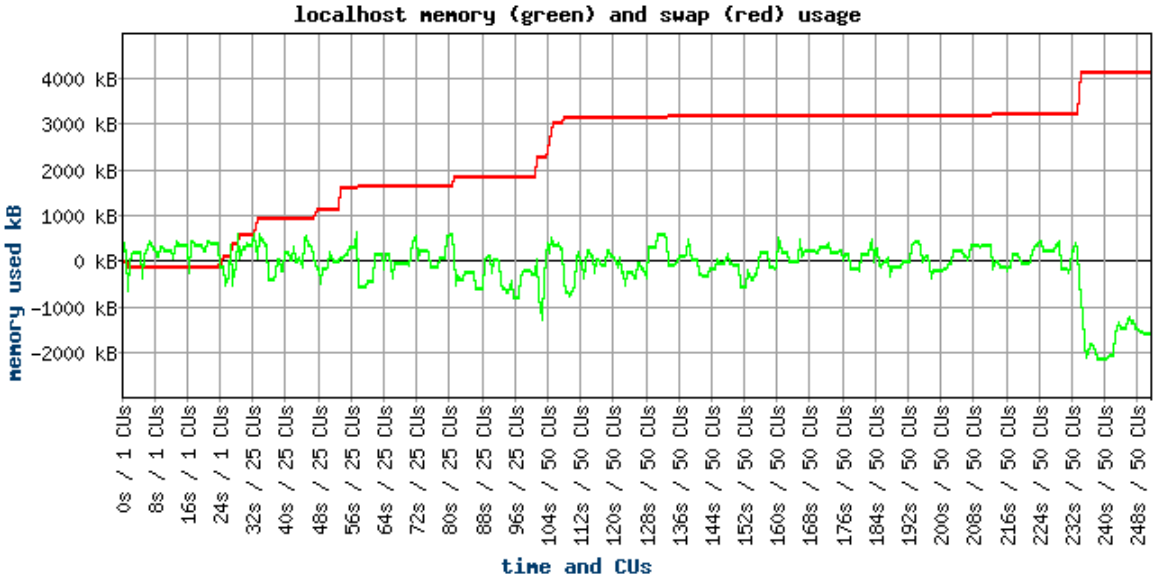


Illustration 13: Ruby on Rails

Analysis

The memory usage once again suffers from the shortfalls of the test environment. From the graphs, we can see that Django has the ideal memory usage curve. It barely needs to swap any memory at all. Turbogears comes close- it has steady memory usage till half way through the test. Symfony too has a steady swap memory usage curve.

Conclusion

We ranked each framework on all the performance parameters. The table below shows how they fared. We accumulate the results to get a final ranking.

Java Servlets comes out on top. This is justified by it not having the additional overheads that a normal framework would incorporate. However, Java being most widely used in web application development, provides a good baseline metric

Framework	Request per second	Time per request	Time per request (mean)	Transaction Rate	Response Time	Throughput	Load Avg	CPU Usage	Memory Swap	Lines of Code	Total	Rank
Symfony	2	2	2	5	5	1	2	6	2	5	32	**4**
Turbogears	5	5	5	1	1	2	2	6	4	1	32	**4**
Catalyst	3	3	3	3	3	4	6	6	1	2	34	**3**
Django	4	4	4	4	4	5	4	6	5	6	46	**2**
Servlets	6	6	6	6	6	6	3	6	6	3	54	**1**
Ruby on Rails	1	1	1	2	2	3	5	6	3	4	28	**6**

Overall Comparison Rankings

These results illustrate the performance of a lightweight application and may not hold true for applications in all cases.