

Uknow InfoHub

Test Report

BIXLRSMB Team

Contents

1	Introduction	2
1.1	Software Introduction	2
1.2	Document Introduction	2
2	Test Scheme	2
2.1	Test Level	2
2.2	Test Environment	2
2.2.1	Software Environment	2
2.2.2	Hardware Environment	3
3	Test Procedure	3
3.1	Testers	3
3.2	Period	3
3.3	Functional Tests	3
3.3.1	Test Case Designing	3
3.3.2	Environment Setting	3
3.3.3	Software Configuring	3
3.3.4	Test Configuring	3
3.3.5	Test Executing	3
3.3.6	Log Analyzing	4
3.4	Performance Tests	4
3.4.1	Test Case Designing	4
3.4.2	Environment Setting	4
3.4.3	Test Executing	4
3.4.4	Log Analyzing	4
4	Test Result	4
4.1	Functional	4
4.2	Performance	4
4.3	Converge	5

5	Test Evaluation	6
5.1	Software	6
5.1.1	Functional	6
5.1.2	Performance	6
5.2	Test	6
5.2.1	Functional	6
5.2.2	Performance	6
A	Test Cases	7
A.1	Sample	7
A.2	User Management Test	8
A.3	Tab Test	8
A.4	Tag Test	9
A.5	User Fetcher Test	9
A.6	Misc	9
A.7	Performance Test	10
A.7.1	Heavy Load API	10
A.7.2	Common API	10
B	Benchmark Result	10

1 Introduction

1.1 Software Introduction

Component	Language	Environment	Lines	Remark
fetcher	Python2	POSIX	747	Including general and user ones
prefilter	Python2	POSIX	54	Not including Training material
api-website	Python2	POSIX	426	Not including execute scripts
frontend-page	HTML/JS/CSS	POSIX	558	CSS:212 JavaScript:153 HTML:193
manage	Python2/Shell etc	POSIX	117	Management scripts
library	Python2	POSIX	699	Common library for all components
tests	Python2/Shell	POSIX	336	Including unit test and benchmark
sum		POSIX	2437	

1.2 Document Introduction

In the following sections of this document, we will present test scheme and results in detail.

In [Sec. 2](#), the methodology and environment the test is conducted is described. Detailed and accurate test procedure will be present in [Sec. 3](#). Test result will be covered in [Sec. 4](#). At last, evaluation of the test will be discussed in [Sec. 5](#).

2 Test Scheme

2.1 Test Level

This test use method which simulating users' requests and determining whether whole project can response each request correctly. If project gives the correct response as design for one request, logic that get involved in generating response will be considered as tested.

So, this test is on API level.

2.2 Test Environment

2.2.1 Software Environment

Name	Version	Purpose
Python UnitTest	2.7	Simulating Request for each API
ApacheBench	>=2.3	Performance Benchmark

Table 1: Software Requirements

2.2.2 Hardware Environment

In order to run the test, a POSIX machine is required.

3 Test Procedure

3.1 Testers

zxytim, ppwwyyxx, vuryleo, jiakai

3.2 Period

18:00 Dec 13th, 2013 to 18:00 Dec 14th, 2013.

3.3 Functional Tests

3.3.1 Test Case Designing

First, read the API list, select out APIs that need to be tested. Then, design use cases for each API base on its function. At last, transform test cases into proper request format and add them into the test suite.

After all, several cases that designed to test the robustness of the APIs are recorded as well. These cases will perform illegal request such as wrong format, permission overstep and injection.

3.3.2 Environment Setting

At first, use virtualenv to build a virtual python environment for this project. Then use install tool located in manage suite to set up runtime environment for it.

In order to provide production environment, a mongodb server and a redis server is also required.

3.3.3 Software Configuring

In this test, the default configure is applied. So no action is needed in this step.

3.3.4 Test Configuring

In order to test some third party function, correct authentication information is required. The configure file can be found easily in test suite.

3.3.5 Test Executing

Execute test entry scripts to do the tests.

3.3.6 Log Analyzing

After getting the test log, parse the useful information out and ordinate them into proper format.

3.4 Performance Tests

3.4.1 Test Case Designing

Several APIs that may have heavy loads are picked out to get the worst performance data. Then a random set of APIs is generated in order to test the average performance. Besides, some APIs that may be requested most frequently are selected to get the common performance.

After all, translate the test requests in functional test into well formed Internet requests and put them in proper places.

3.4.2 Environment Setting

In propose of simulating real world performance, client and server are located on different machines, so the network transmission is considered.

To do so, deploy the project in production mode on one machine, and get the test suite on another.

Network throughput benchmark is tested using ApacheBench¹, with concurrency of 4 and 10,000 requests on each API. The CPU, memory and IO count usage of server program will be intensively examined.

3.4.3 Test Executing

Execute test entry scripts to do the tests.

3.4.4 Log Analyzing

Performance test will generate plenty of log. So a data visualization procedure is taken to display the data in different collections and patterns.

4 Test Result

4.1 Functional

All functional tests are passed. For more details please refer to [App. A](#).

4.2 Performance

Here a sample of test result is presented. For detailed all plots, see appendix.

¹<http://httpd.apache.org/docs/2.2/programs/ab.html>

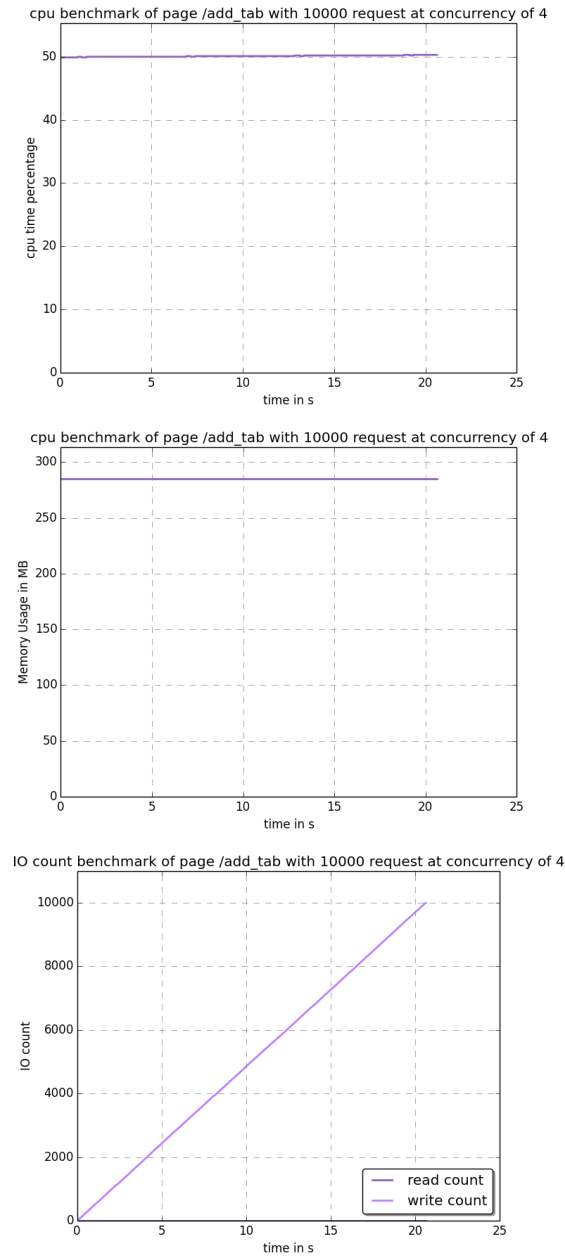


Figure 1: Test result of **add_tab** API

4.3 Converge

The converge ratio reached 90%. For more details please refer to appendix converge sheet.

The code that are not converged is mainly exception handle phase. But for each kind of exception handle, at least one case is designed to test it. For example, all

most all APIs are login required, but only one test case that try to logout while not logged in is designed. For one kind of exception handle procedure is the same, but may occur in many situation. Designing duplicate test cases for each API doesn't means a lot but cause plenty of time.

For others, they are designed for further widening usage that not implement yet. Such as more configurable fetcher, user defined color theme and so on.

5 Test Evaluation

5.1 Software

5.1.1 Functional

All APIs do the right work as what they are designed for. And all APIs is robust even under sedulous injection or other kind of inroads. Getting this result shows that the develop team knows lots of knowledge in network security field, and does a good design at the very beginning.

5.1.2 Performance

Both average performance and common performance are quite well, but there are still some work to do to speed worst case up. However, the enhancement is limited, for in worst case the API will return plenty of data, and the bottleneck is network transform.

So, may a increment request API is needed in the future. This kind of API will remember what they has provided and won't give them out again and again but it has changed.

Even though, the worst response time is admissible by user, so further enhancement can be arrested until it really make sense.

5.2 Test

5.2.1 Functional

Functional test includes legal and illegal cases, considered boundary situation and uncommon use case. Besides, many common injection case are used, to test the security of the APIs. This part of test cases are really completed and cover many professional assault method.

5.2.2 Performance

For better result that determines where the bottle neck is maybe a enhancement for the performance test. Current result only shows the response time for whole APIs but doesn't have details like network transform time and database query time.

So further enhancement may don't have critical target or was done on unimportant part.

A Test Cases

A.1 Sample

Name	Purpose	Input	Expect	Real	status
Sample	Whether the API website works	None	answer=42	answer=42	

A.2 User Management Test

Name	Purpose	Input	Expect	Real	status
Login Fail	Login with unregistered username	Fake user	no such user	no such user	
Register	Register a new user	Unregistered user	User's information	User's information	Logged in
Register Duplicate	Register with a registered username	Register user	user already exists	user already exists	
Login	Login	Register user	Users' information	Users' information	Logged in
Logout	Logout when logged in	None	Login Page	Login Page	Logged out
Logout Fail	Logout when not logged in	None	401 error	401 error	
Login Fail	Login with incorrect password	Incorrect user	wrong password	wrong password	
Login Illegal	Login with illegal format	Illegal request	400 error	400 error	
Register Illegal	Register with illegal format	Illegal request	400 error	400 error	
Register Too Short	Register with short field	Short field	field too short	field too short	
Login Test	Determine whether logged in	Logged in required API	200	200	

A.3 Tab Test

Name	Purpose	Input	Expect	Real	status
Add Fail	Add Tab when not logged in	Tab information	401 error	401 error	
Add Fail	Add Tab with illegal format	Illegal request	400 error	400 error	
Add	Add Tab	Tab information	Tab information	Tab information	
Add	Add Tab that exists	Tab information	tab already exists	tab already exists	
Get All	Get all Tabs	None	Tab information	Tab information	
Get	Get specific Tab	Tab name	Tab information	Tab information	
Delete	Delete specific Tab	Tab name	Tab information	Tab information	
Delete	Delete not exists Tab	Tab name	Ignored	Ignored	

A.4 Tag Test

Name	Purpose	Input	Expect	Real	status
Add Fail	Add Tag when not logged in	Tag information	401 error	401 error	
Add Fail	Add Tag with illegal format	Illegal request	400 error	400 error	
Add	Add Tag	Tag information	Tag information	Tag information	
Add	Add Tag that exists	Tag information	tab already exists	tab already exists	
Get All	Get all Tags	None	Tag information	Tag information	
Get	Get specific Tag	Tag name	Tag information	Tag information	
Delete	Delete specific Tag	Tag name	Tag information	Tag information	
Delete	Delete not exists Tag	Tag name	Ignored	Ignored	

A.5 User Fetcher Test

Name	Purpose	Input	Expect	Real	status
Get All	Get all Fetcher	None	Fetcher information	Fetcher information	
Enable	Enable a Fetcher	ID and configure	Fetcher information	Fetcher information	
Refresh	Refresh Fetcher	None	Fetch result	Fetch result	
Disable	Disable a Fetcher	ID	Fetcher information	Fetcher information	

A.6 Misc

Name	Purpose	Input	Expect	Real	status
Get All Entries	Get all Entries	None	Entries information	Entries information	

A.7 Performance Test

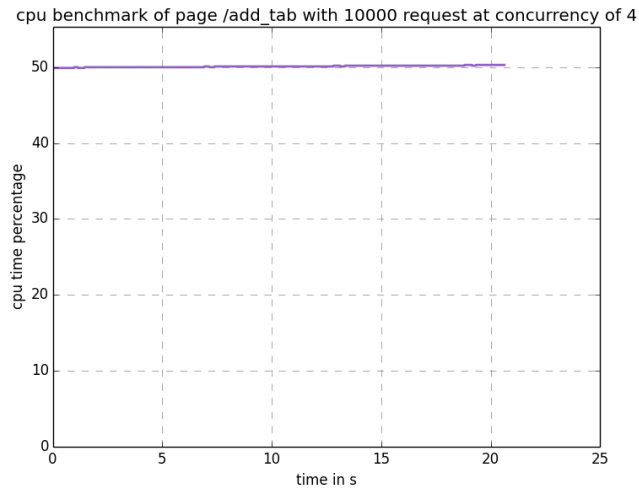
A.7.1 Heavy Load API

1. Get All Fetcher
2. Get All Tab
3. Get All Tag
4. Get All Entries

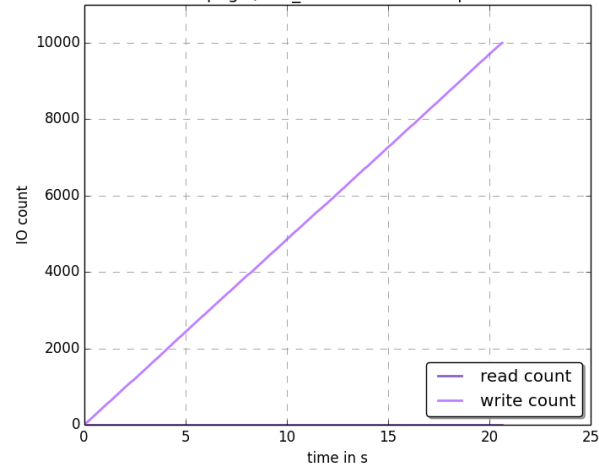
A.7.2 Common API

1. Login
2. Register
3. Logout
4. Get All Entries

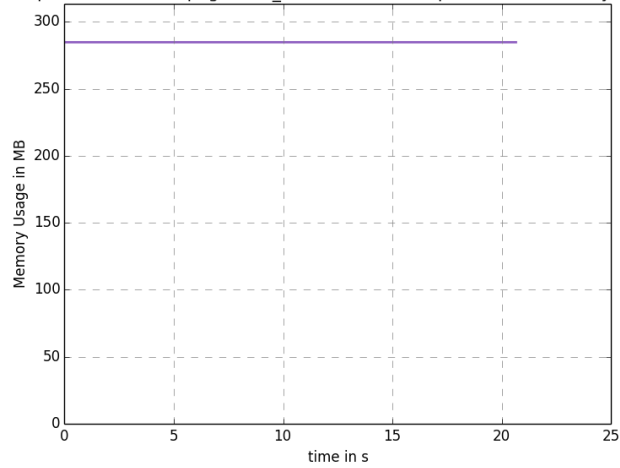
B Benchmark Result



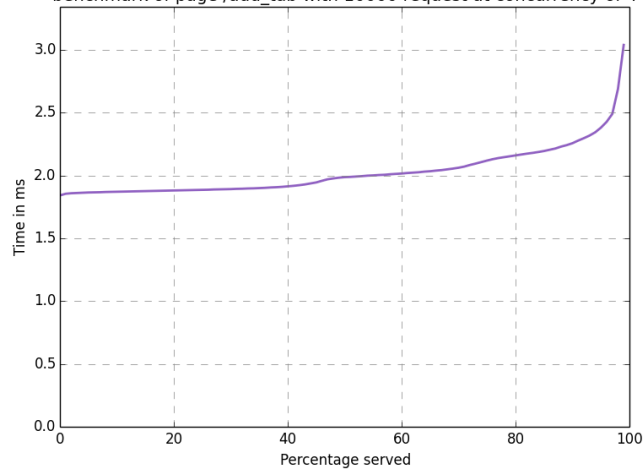
IO count benchmark of page /add_tab with 10000 request at concurrency of 4



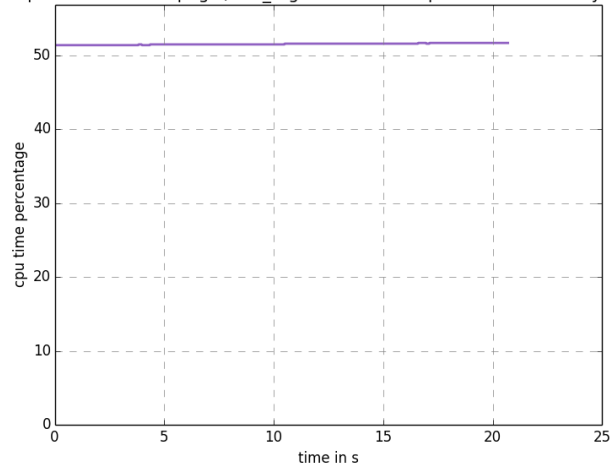
cpu benchmark of page /add_tab with 10000 request at concurrency of 4



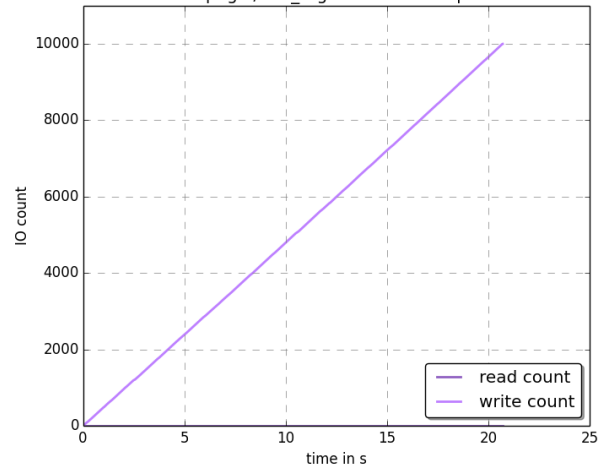
benchmark of page /add_tab with 10000 request at concurrency of 4



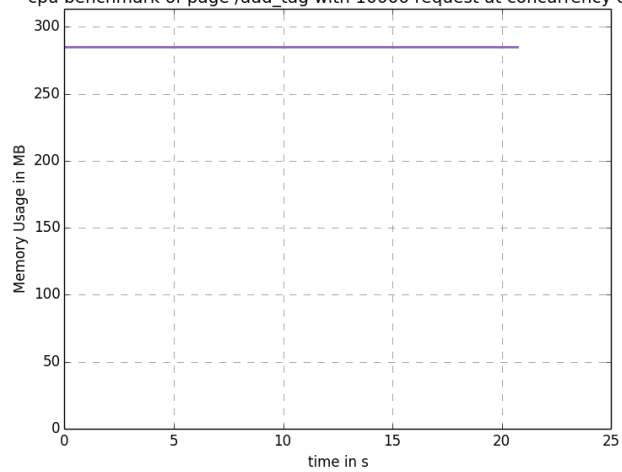
cpu benchmark of page /add_tag with 10000 request at concurrency of 4

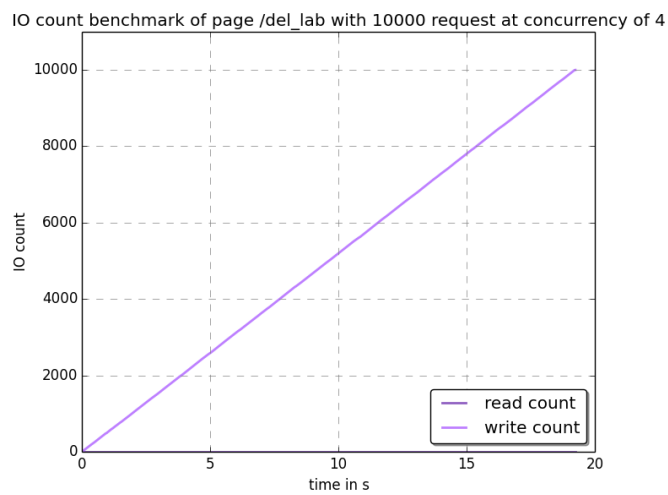
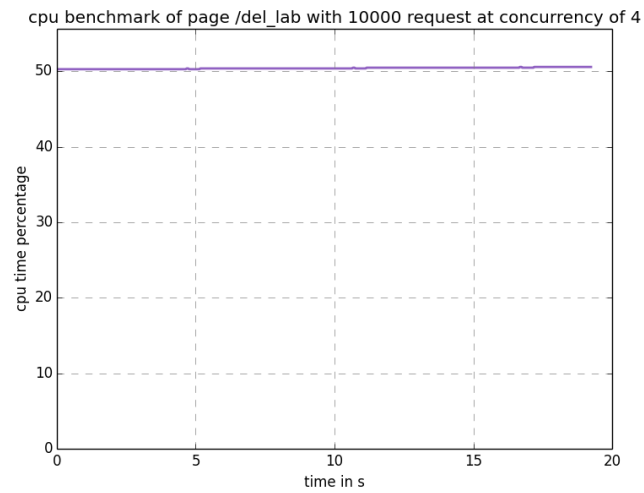
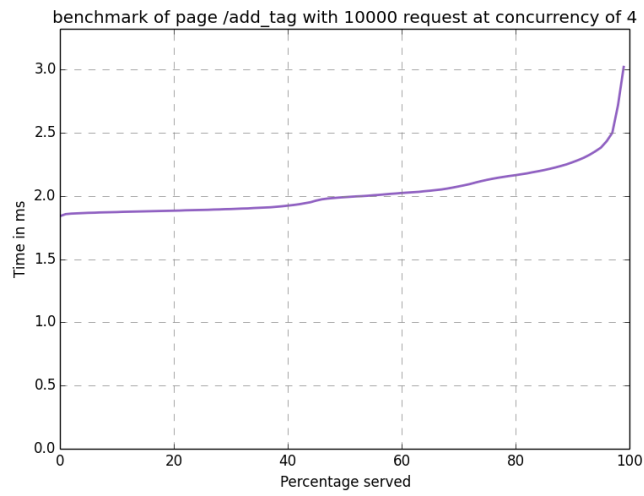


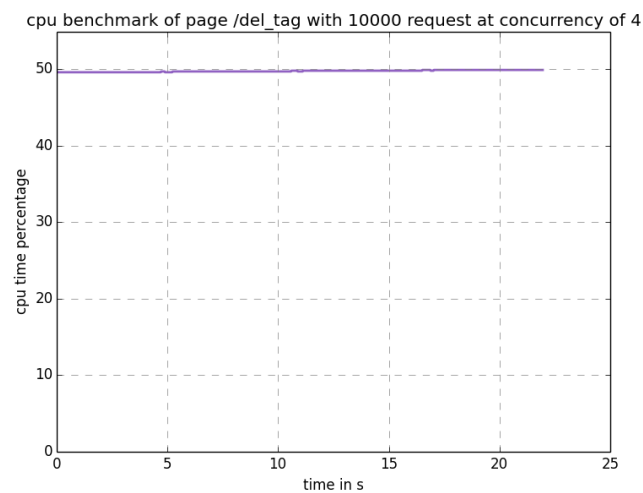
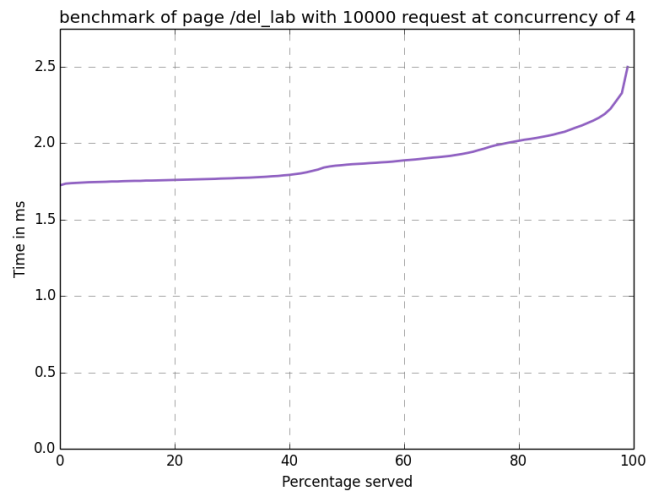
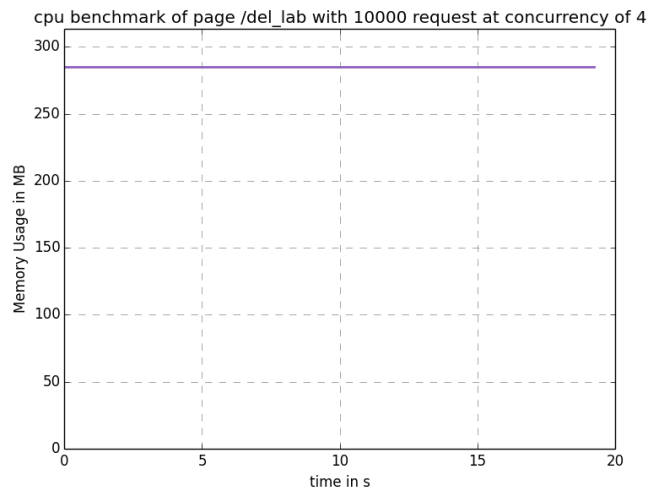
IO count benchmark of page /add_tag with 10000 request at concurrency of 4



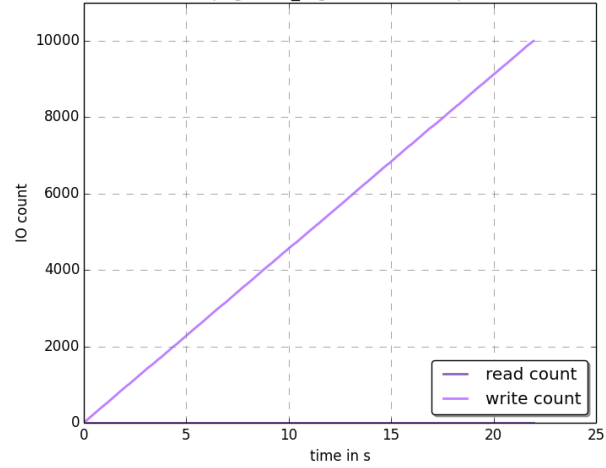
cpu benchmark of page /add_tag with 10000 request at concurrency of 4



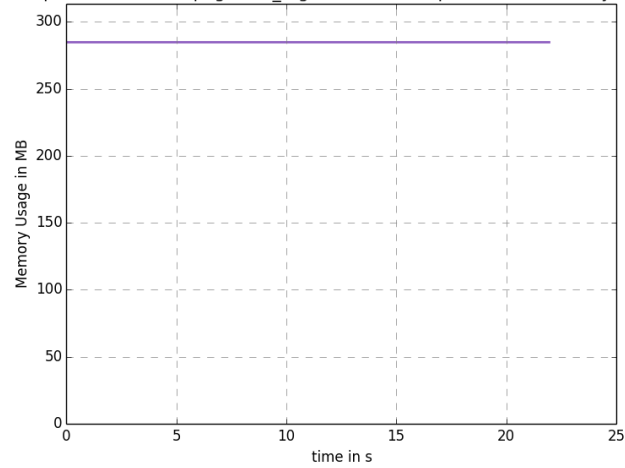




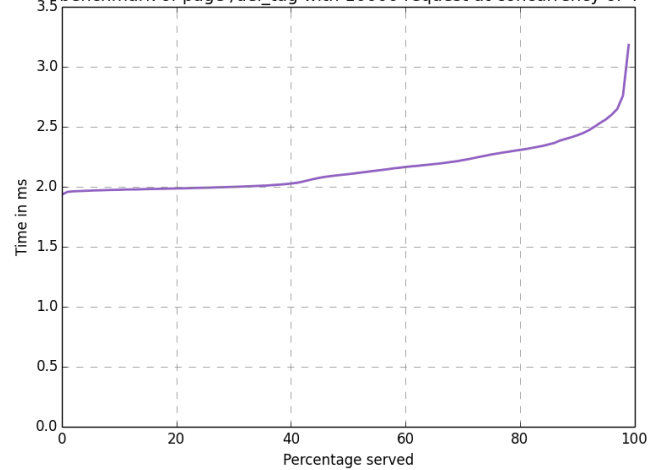
IO count benchmark of page /del_tag with 10000 request at concurrency of 4



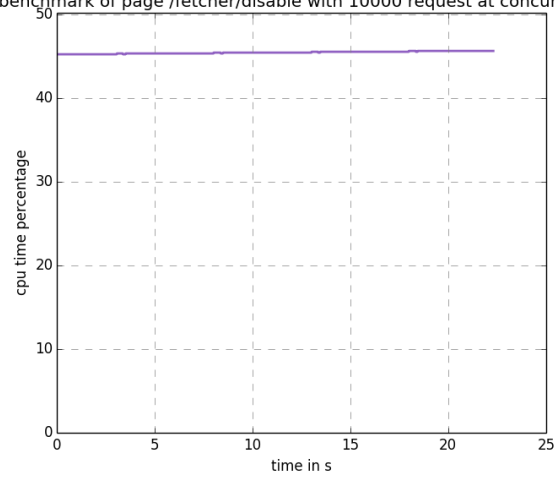
cpu benchmark of page /del_tag with 10000 request at concurrency of 4



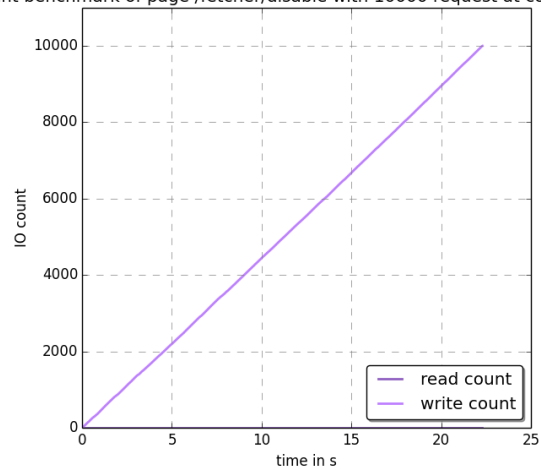
benchmark of page /del_tag with 10000 request at concurrency of 4



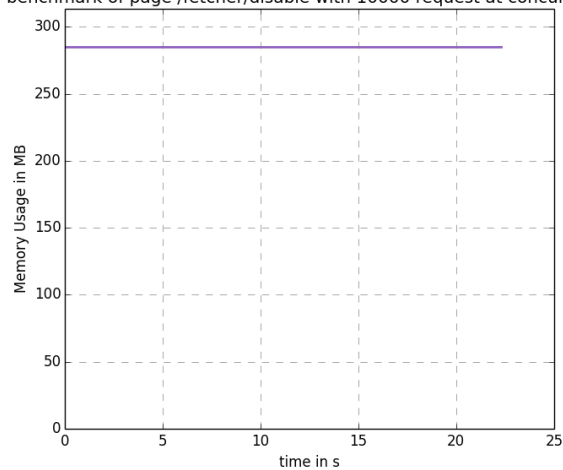
cpu benchmark of page /fetcher/disable with 10000 request at concurrency of 4



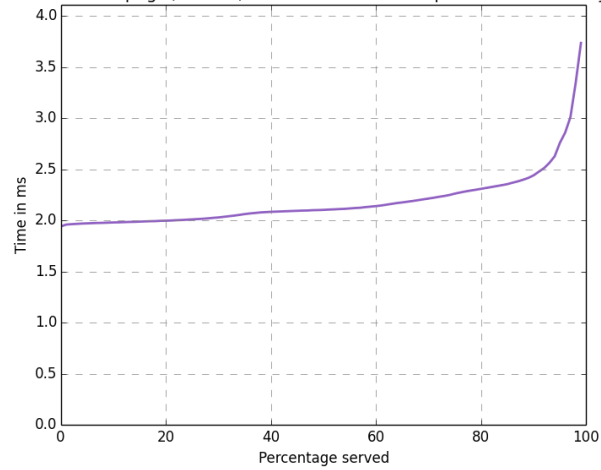
count benchmark of page /fetcher/disable with 10000 request at concurrency o



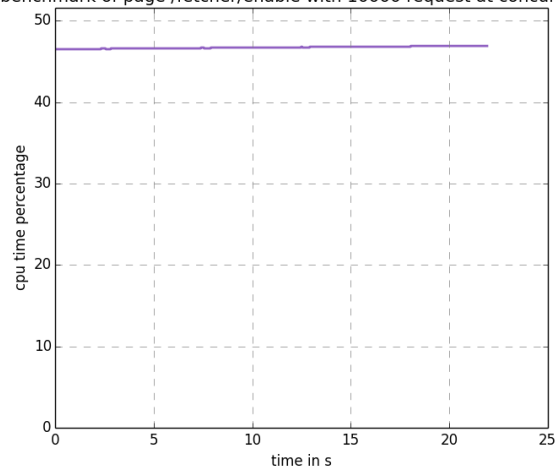
cpu benchmark of page /fetcher/disable with 10000 request at concurrency of 4



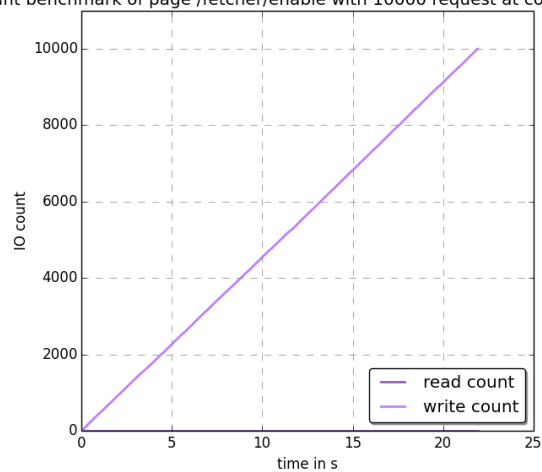
benchmark of page /fetcher/disable with 10000 request at concurrency of 4



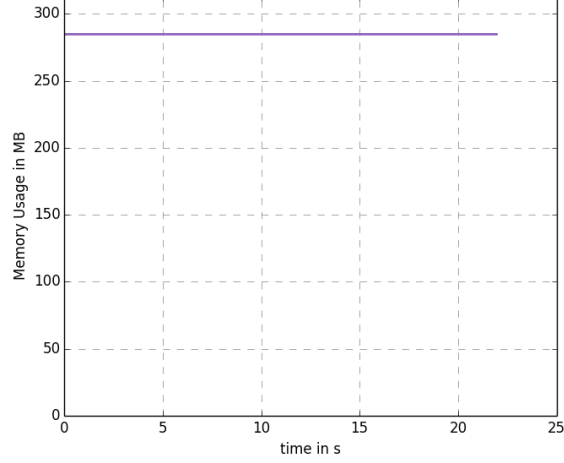
cpu benchmark of page /fetcher/enable with 10000 request at concurrency of 4



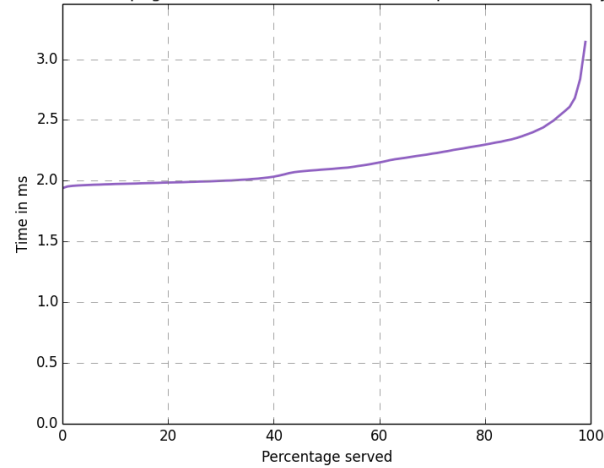
count benchmark of page /fetcher/enable with 10000 request at concurrency of 4



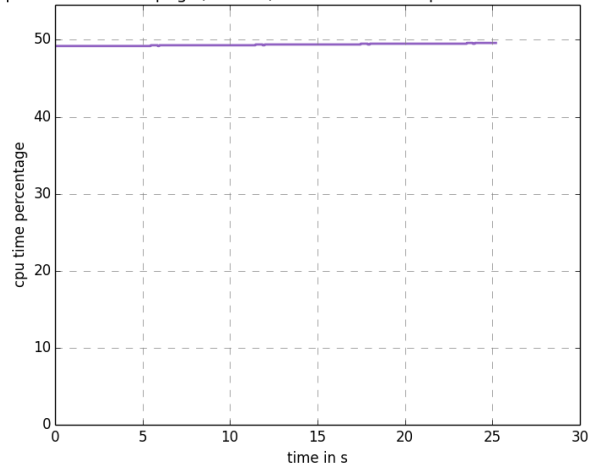
cpu benchmark of page /fetcher/enable with 10000 request at concurrency of 4



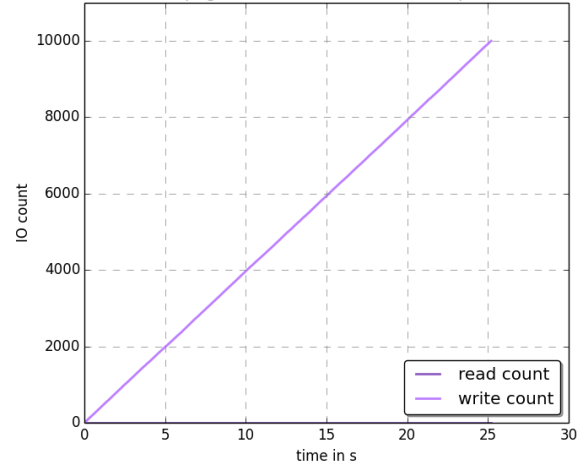
benchmark of page /fetcher/enable with 10000 request at concurrency of 4



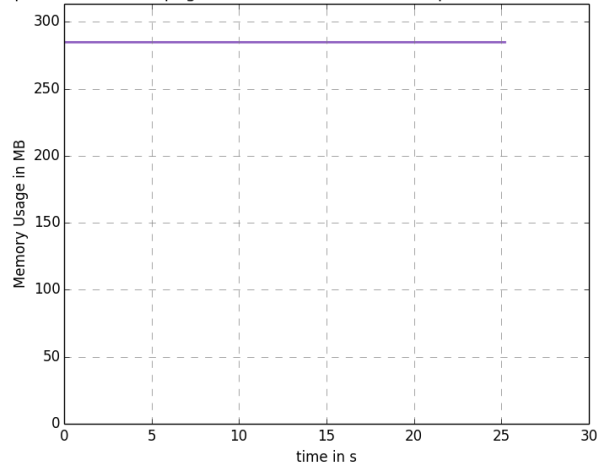
cpu benchmark of page /fetcher/list with 10000 request at concurrency of 4



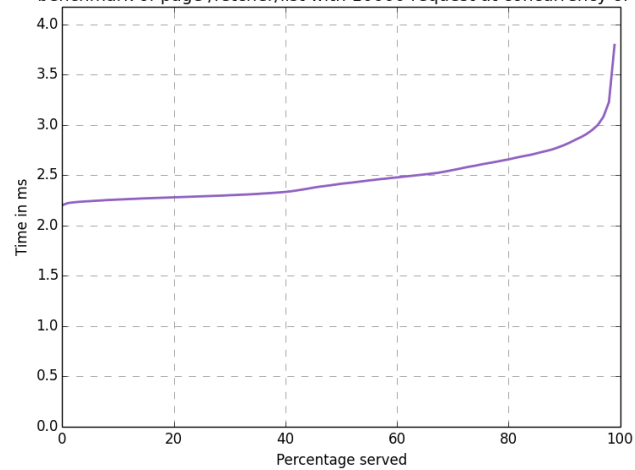
O count benchmark of page /fetcher/list with 10000 request at concurrency of 4



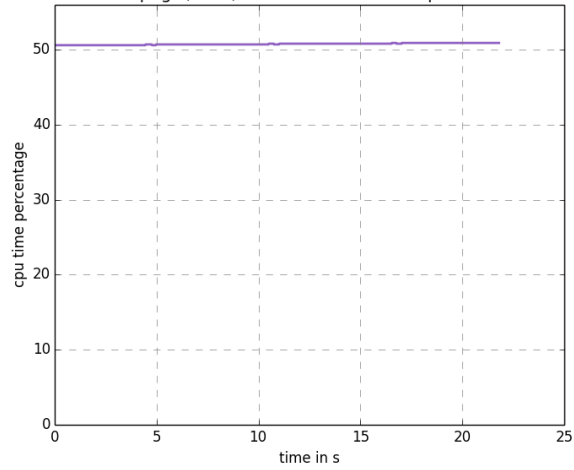
cpu benchmark of page /fetcher/list with 10000 request at concurrency of 4



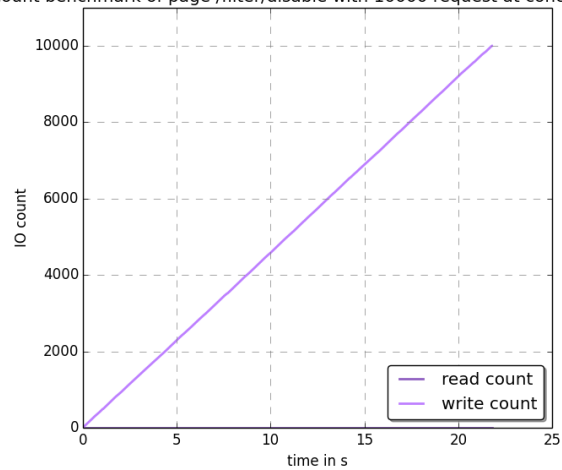
benchmark of page /fetcher/list with 10000 request at concurrency of 4



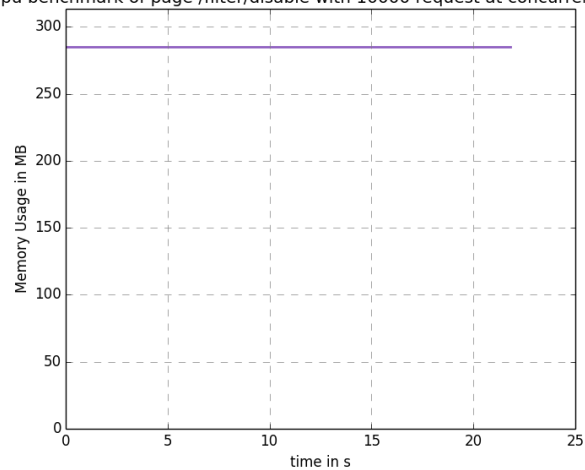
cpu benchmark of page /filter/disable with 10000 request at concurrency of 4



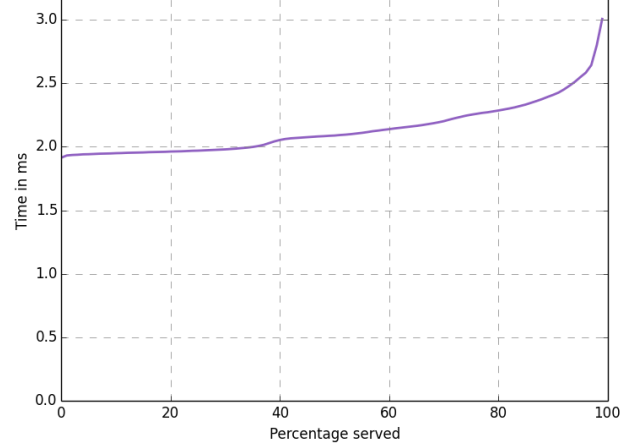
IO count benchmark of page /filter/disable with 10000 request at concurrency of 4



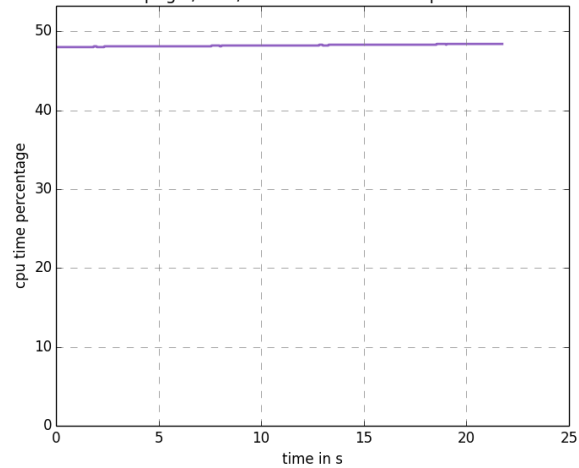
cpu benchmark of page /filter/disable with 10000 request at concurrency of 4



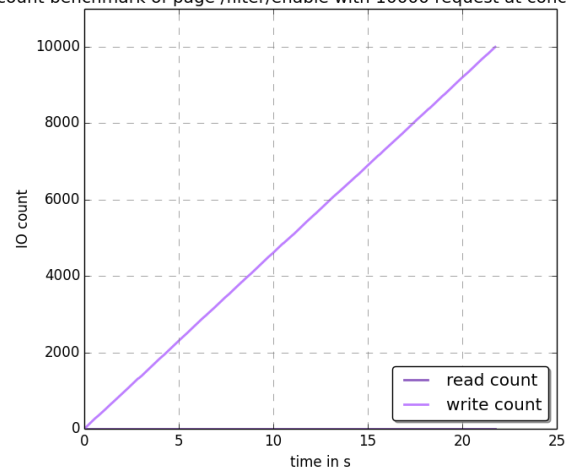
benchmark of page /filter/disable with 10000 request at concurrency of 4



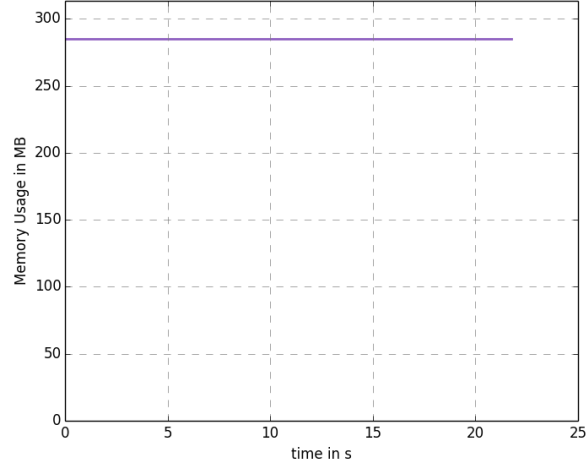
cpu benchmark of page /filter/enable with 10000 request at concurrency of 4



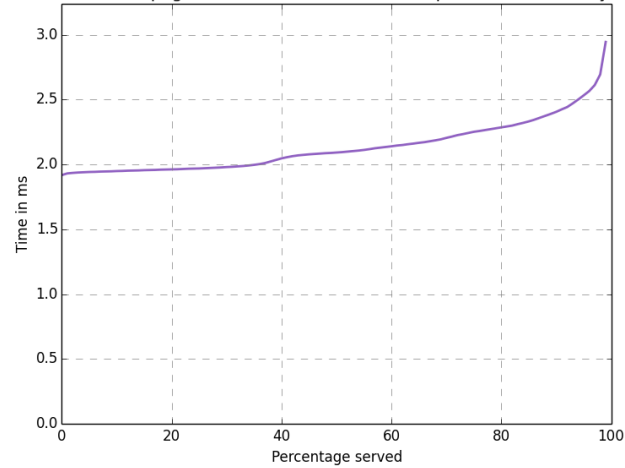
IO count benchmark of page /filter/enable with 10000 request at concurrency of 4



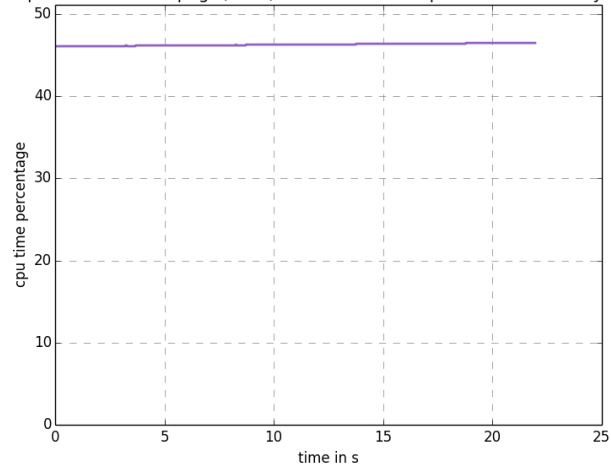
cpu benchmark of page /filter/enable with 10000 request at concurrency of 4



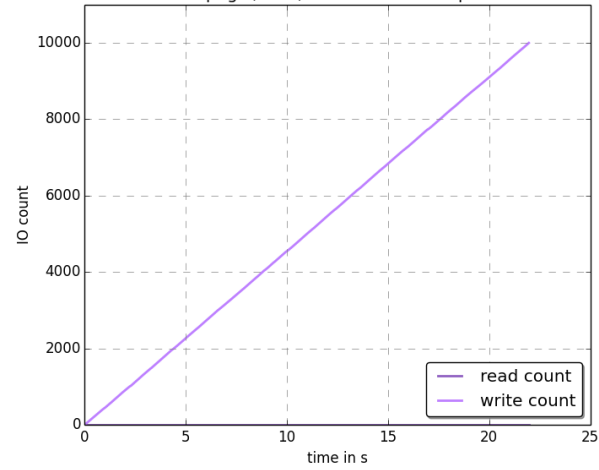
benchmark of page /filter/enable with 10000 request at concurrency of 4



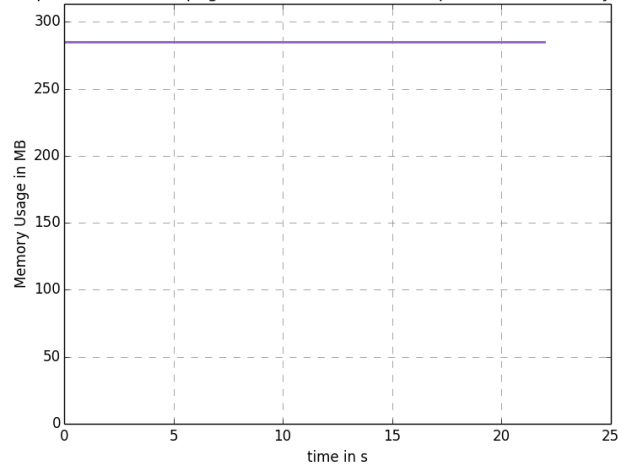
cpu benchmark of page /filter/list with 10000 request at concurrency of 4



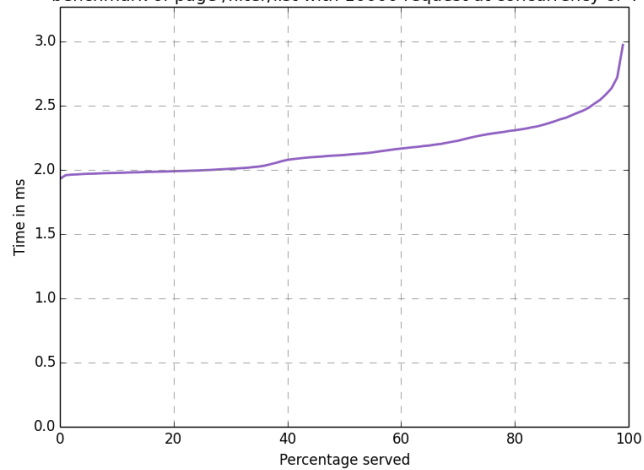
IO count benchmark of page /filter/list with 10000 request at concurrency of 4



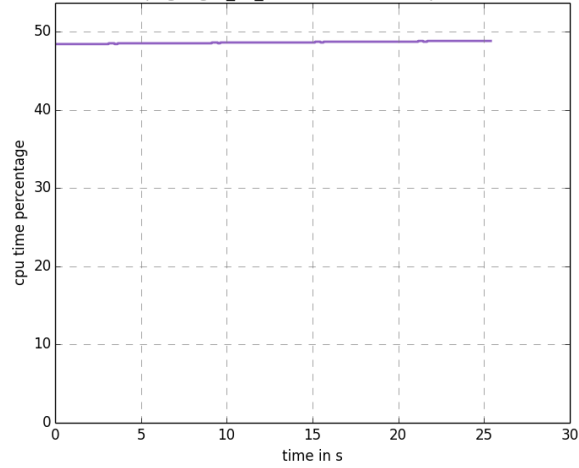
cpu benchmark of page /filter/list with 10000 request at concurrency of 4



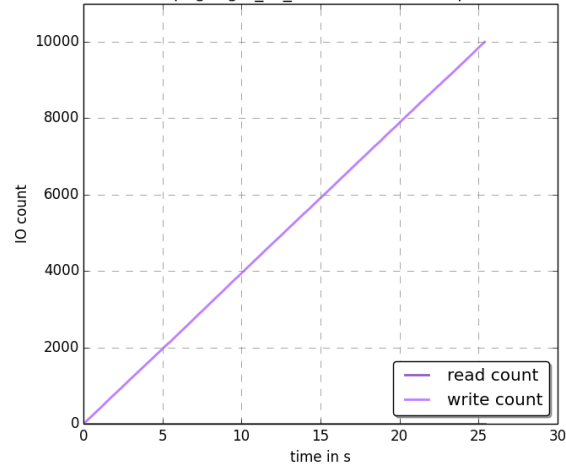
benchmark of page /filter/list with 10000 request at concurrency of 4



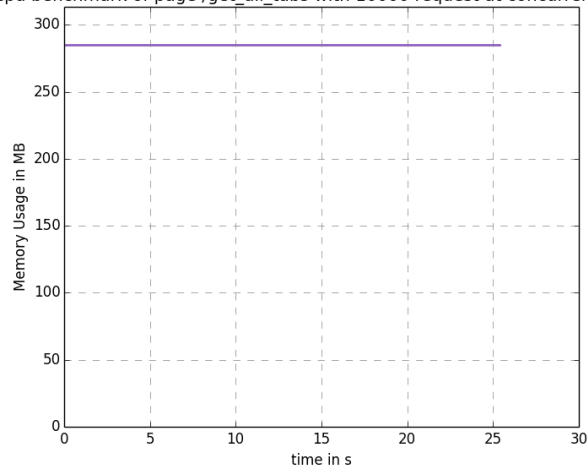
cpu benchmark of page /get_all_tabs with 10000 request at concurrency of 4



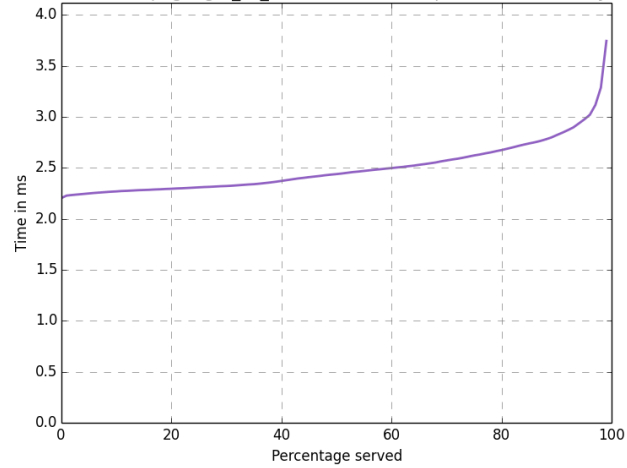
IO count benchmark of page /get_all_tabs with 10000 request at concurrency of 4



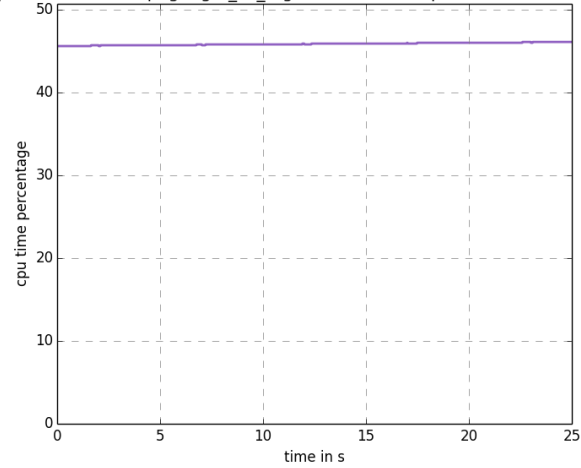
cpu benchmark of page /get_all_tabs with 10000 request at concurrency of 4



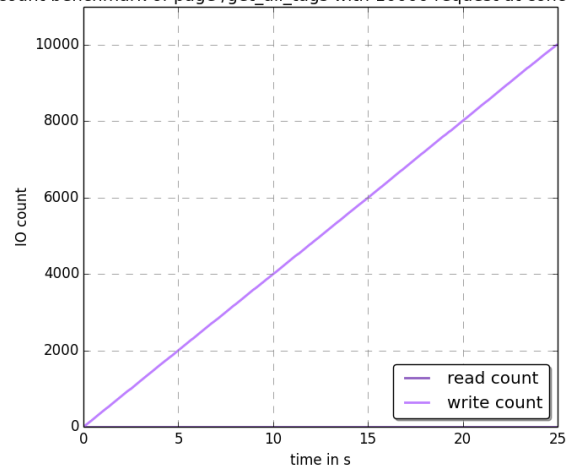
benchmark of page /get_all_tabs with 10000 request at concurrency of 4



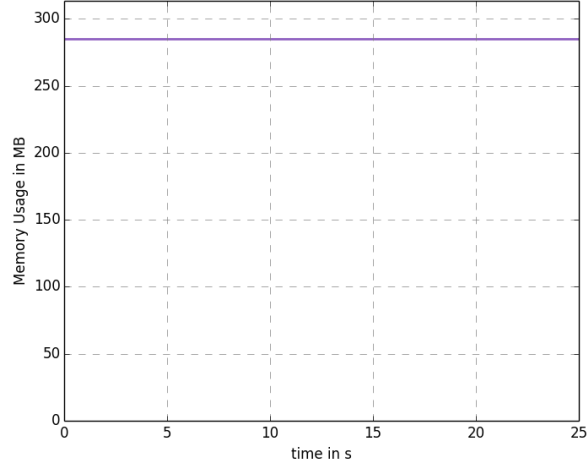
cpu benchmark of page /get_all_tabs with 10000 request at concurrency of 4



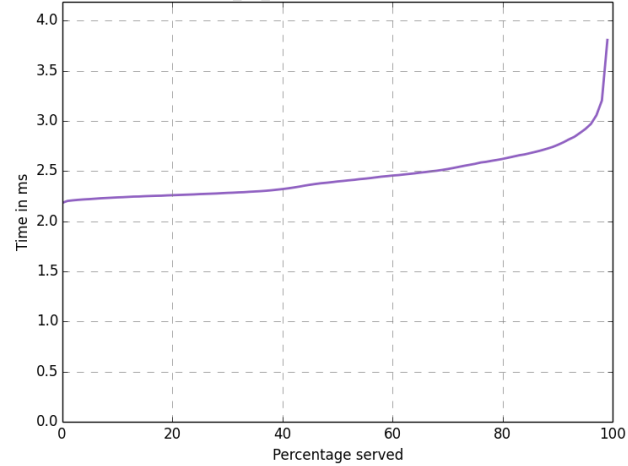
IO count benchmark of page /get_all_tabs with 10000 request at concurrency of 4



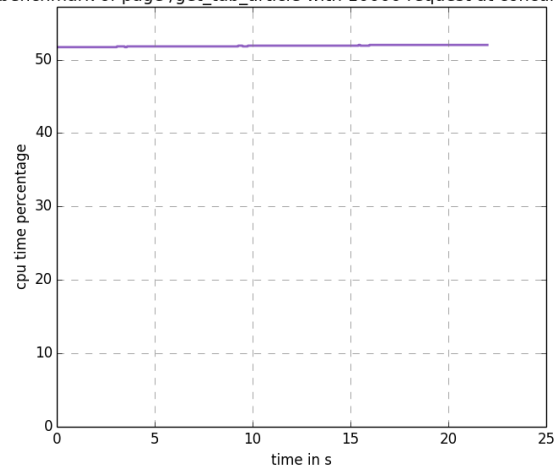
cpu benchmark of page /get_all_tags with 10000 request at concurrency of 4



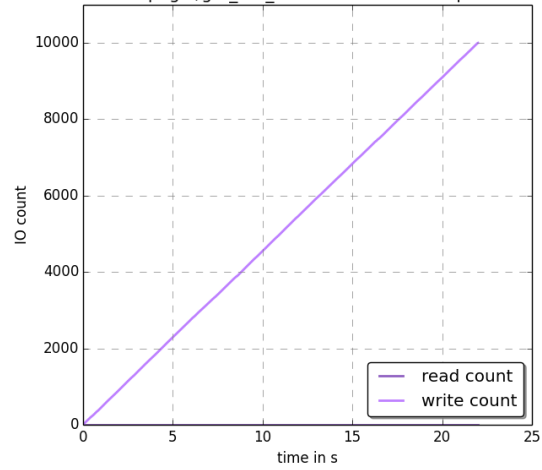
benchmark of page /get_all_tags with 10000 request at concurrency of 4



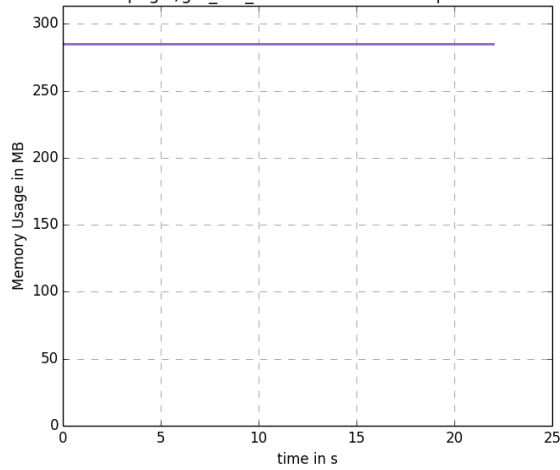
cpu benchmark of page /get_tab_article with 10000 request at concurrency of 4



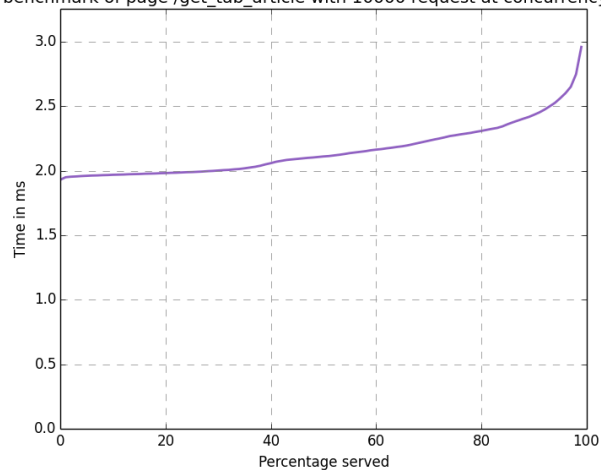
count benchmark of page /get_tab_article with 10000 request at concurrency o

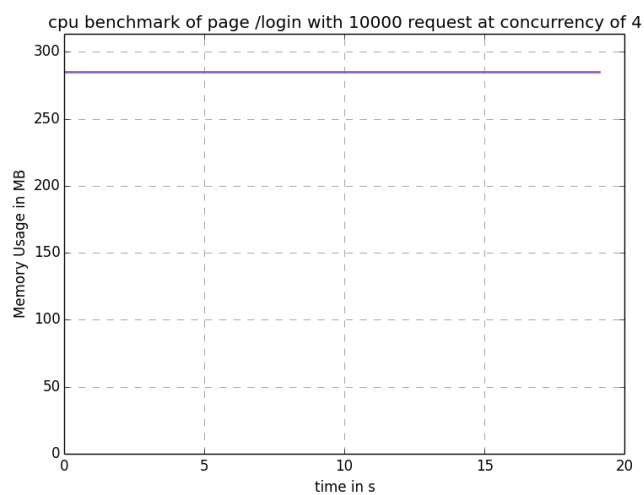
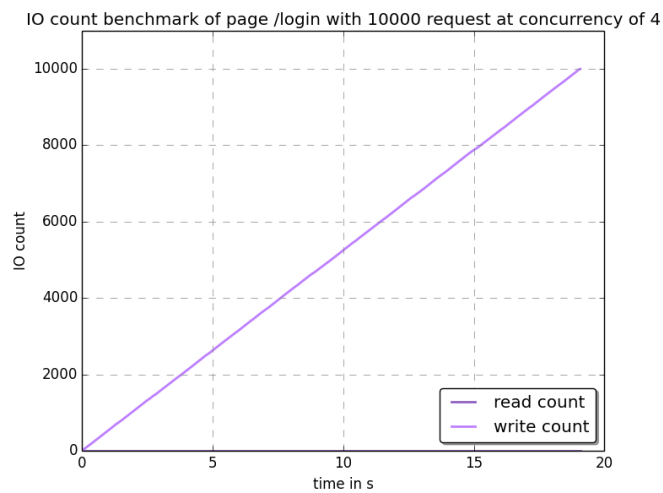
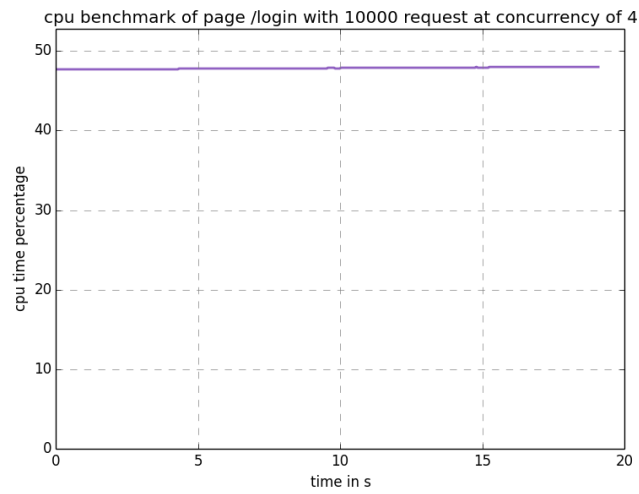


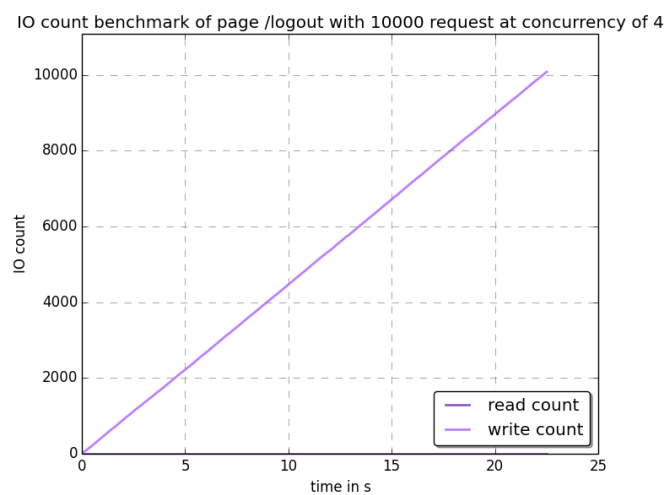
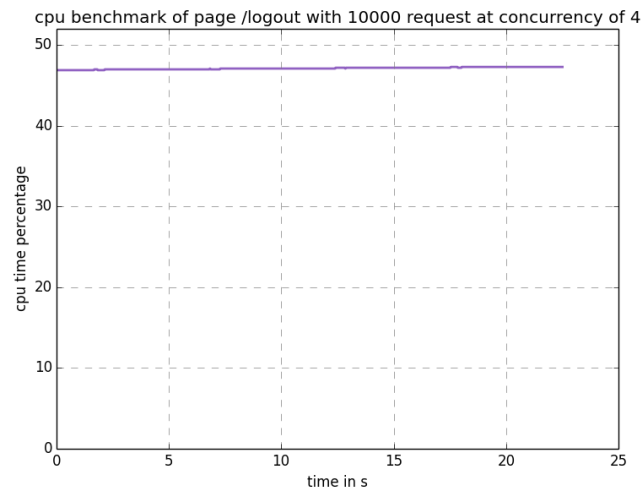
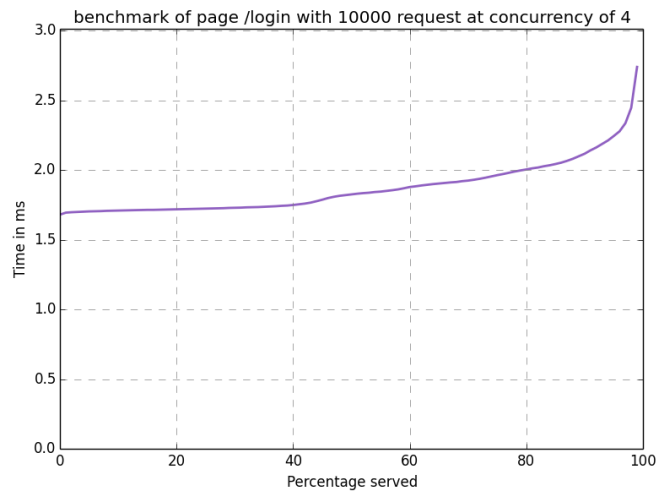
cpu benchmark of page /get_tab_article with 10000 request at concurrency of 4

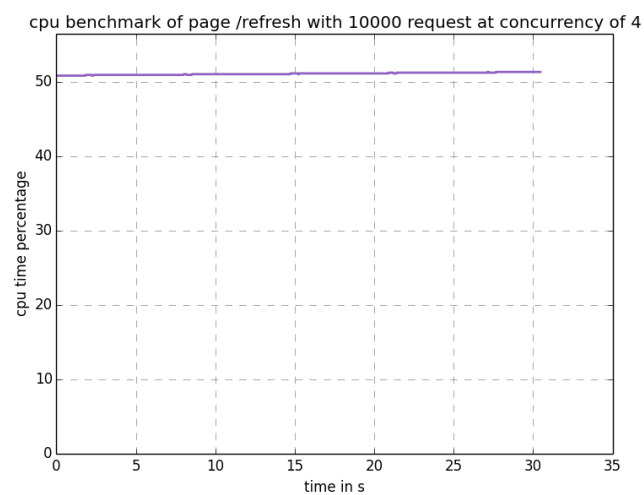
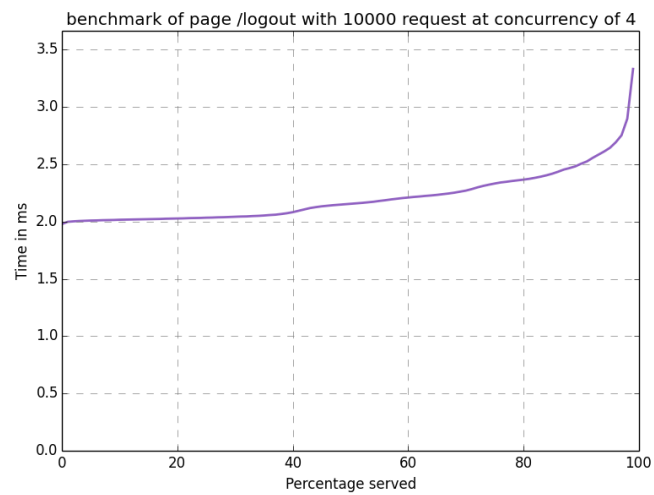
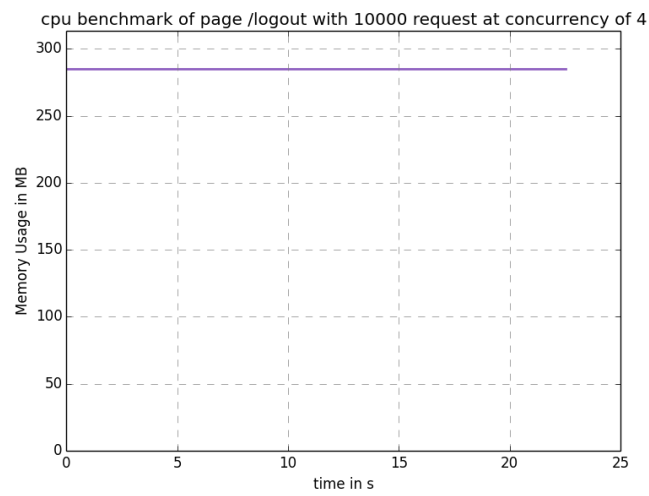


benchmark of page /get_tab_article with 10000 request at concurrency of 4

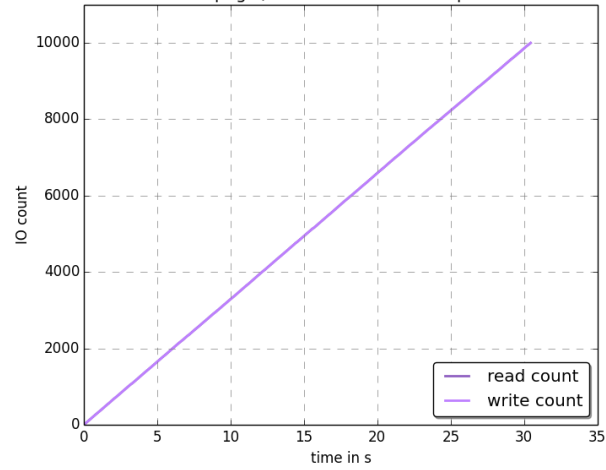




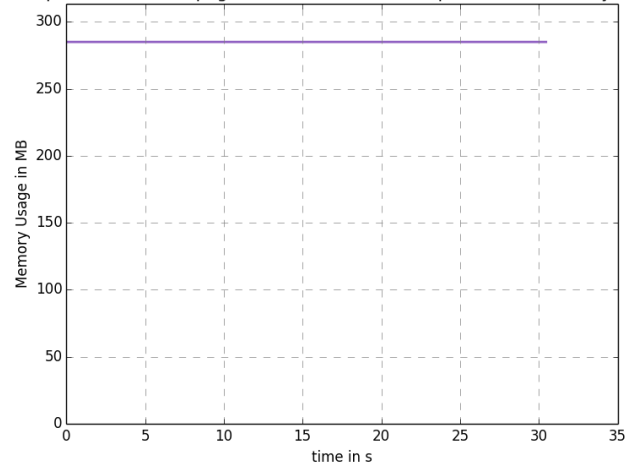




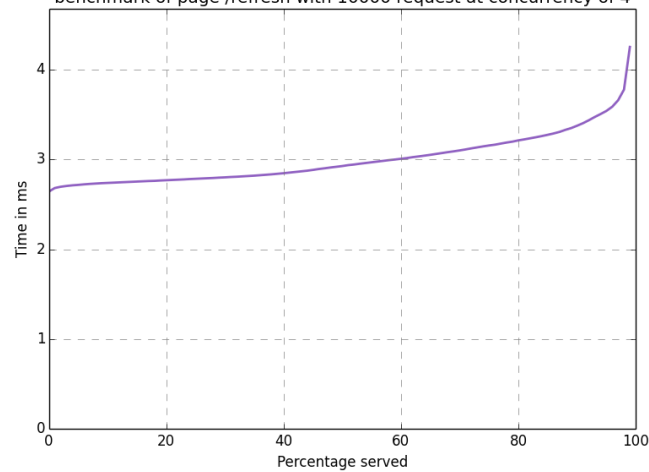
IO count benchmark of page /refresh with 10000 request at concurrency of 4



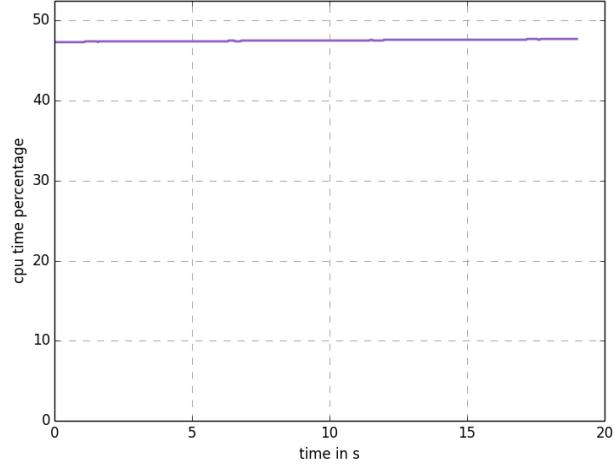
cpu benchmark of page /refresh with 10000 request at concurrency of 4



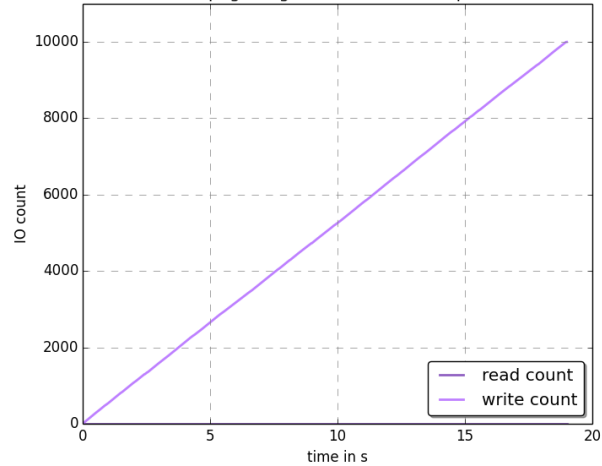
benchmark of page /refresh with 10000 request at concurrency of 4



cpu benchmark of page /register with 10000 request at concurrency of 4



IO count benchmark of page /register with 10000 request at concurrency of 4



cpu benchmark of page /register with 10000 request at concurrency of 4

