A Method of Optimizing Django Based On Greedy Strategy

Jian Zhou
Department of Computer Science and Technology
State Key Laboratory for Novel Software Technology of NanJing University
NanJing, China
nju.mungco@gmail.com

Lin Chen, Hui Ding, Jingxuan Tu, Baowen Xu
Department of Computer Science and Technology
State Key Laboratory for Novel Software Technology of NanJing University
NanJing, China
lchen@nju.edu.cn

Abstract—**Django on Python does well in agile web development. Python is dynamic programming language, as is known, its runtime efficiency is low. the question is how it will affect Django's efficiency. This Paper answered the question from the angle of memory optimization, by improving the efficiency of Python in memory use; we will see how it will affect django. Python manages non-container objects with the technology of memory pool, after a consequence of allocate and free operations, it will produce memory fragment, and the garbage collection module can hardly handle it. This paper proposed a new non-container object management greedy algorithm; it can minimize the probability of generating fragments, thus the garbage collection module can collect garbage non-container objects in time. We conduct our experiment on the benchmark of an open-source project named Unladen-Swallow. When Django renders a template, it will save about 10% memory, with almost the same time consuming.**

Keywords—web framework; dynamic programming language; optimization; memory pool; garbage collection

I. INTRODUCTION

Web application development has gone through a rapid development with large demands [1]. In order to get customer's feedback and rapidly make adjustment, the programmers should have the ability of agile development[2]. Web technology is sure very mature, but the proliferation of components, frameworks and tools for the development of web applications make the programmers face the challenge of heterogeneity[2]. Web frameworks bases on dynamic programming language brought developers a new developing paradigm: such as Ruby on Rails and Django on Python. This agile web framework can simplify the web application progress.

There are sorts of Web frameworks based on python, such as Django, Grok, PyIons, TurboGears, Web2py, Zope2, which are all full-stack frameworks[9]. These full-stack frameworks provide complete solutions for programmers to develop web applications, but few care about their efficiency. These paper focuses on the question that how Python's runtime efficiency affect its web frameworks. Python is a kind of dynamic programming language with high developing efficiency but low running efficiency. One important reason is the design philosophy that everything is object; the Python VM is obliged to have many memory allocation and type checking operations. We tried to improve Python VM's memory usage efficiency, and see how it affects its web frameworks.

II. PROBLEM DESCRIPTION

Python has many implementations, such as CPython, Jython, IronPython. CPython was written in C and is the original version. Jython runs on Java VM, IronPython runs on .NET. In this paper, we refer Python to the version 2 of CPython, because most projects in pypi are developed by it[4].

A. Motivation example

No matter what programming language you are using, Integer may be the most basic type. During the running of python programs on the Python VM, there will be many integers been allocated. The following is an example:

```
>>> for i in range(344,43332):
>>>     print i
```

The Python VM will first allocate 42989 integers, and then print them. What frustrates us is that the Python VM never frees these memory space. In computing intensive Python projects, we can foresee there exits many similar but more complicate cases.

In fact, this kind of memory management has a big problem. You allocate memory frequently but didn't carefully free them. Python VM solved this problem, in fact, but not quite perfect, which results in low efficiency in memory usage, especially in Web application.

B. Python's memory management system

In Python, everything is object. Basic data type, advanced data types, functions, modules, process's stack frame are all objects. Some objects are very small, e.g. integer objects, some are a little large, e.g. function objects. Frequently allocating and freeing objects will raise the problem of fragment and increase system performance cost.

Python VM's memory management is based on reference count, in order to solve the circular references problem, it classifies all Python objects into two parts, one is container objects, and the other is non-container objects. Integer objects, float objects are all non-container objects; list objects, frame objects are container objects. Python VM has an module named gc that is used to solve the circular reference problem. This module will be triggered in two cases described as follows:

1. The number of circular referenced objects reached a pre-defined threshold;

2. Call it manually.

The management of container objects gets a good balance between fragment and system performance cost. The management of non-container object is much easier. Because the non-container objects are quite common, the Python VM adopts the memory poll technology to handle the allocation and free operations. Take integer objects for example, its main architecture is shown in the Fig. 1:

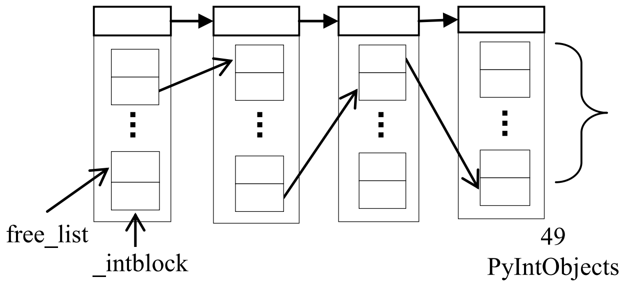

Fig. 1 integer objects model

When an integer object becomes garbage, Python VM only insert it into the head of free list. With the Continuous memory allocation and free operations, Usable integer objects will be scattered in the memory pool. It will be very hard to free one whole block back to the operating system, and it will also cost too much to defrag. So the memory that was occupied by the Python VM satisfies a simple regular:

$$MemofInt_{now} = max(\{ f(x) \mid x \in [t_{begin}, t_{now}]\})$$

In addition, the garbage collection module will scan every block to see if all the integer objects in this block is garbage, if so, it will free this block. In order to implement it, Python's designers give two hypotheses: one is that memory is large enough; the other is that there will not be too many integers. However, with the widely use of Python, especially in science computing, machine learning, Web application, both of these hypotheses are violated.

In summary, Python VM has the following three defects in non-container objects management:

1. Totally depend on garbage collection module. If the number of circular referenced objects didn't reach the threshold, the Python VM will never free the block.

2. Low efficiency. The garbage collection module will scan all the integer objects.

3. Defragment is difficult. Even if you trigger the garbage collection module, only a limited memory can be freeed because the integer objects are scattered in all the blocks.

In this paper, all these problems are solved.

III. METHODS

A. An greedy algorithm for non-container objects managment

In this section, we will introduce a new non-container objects management model for Python. This model can make itself independent from garbage collection module. In this model all the blocks are classified into three categories, they are Empty, Partly, Fully; the blocks in each of the categories are linked by a linked list: L_Empty, L_Partly, L_Fully. The process of allocating and freeing integer objects are controlled by a greedy algorithm. This algorithm is detailed described below:

a. Allocating an object:

When the Python VM allocate an integer object, the block it choose should satisfy the following formula:

$$block \in L\_Partly \wedge block.free\_num = min(\{free \mid free = block.free\_num \wedge b$$

If no usable integer objects remained in a block, the block should be moved into L_Fully. After initialization, L_Partly list should keep at least one block.

b. freeing an object:

There are three cases when freeing an object:

1. If the block the object belongs to is in L_Fully, it will be moved into the head of L_Partly.

2. If the block the object belongs to is in L_Partly; if this block used out, it should be move to L_Empty, or it should compare with the number that the block linked next remained, Swap them if smaller.

3. Collect the blocks in L_Empty.

The pseudo code described as follows:

- algorithm1 allocating an integer

```
1. if L_Empty==NULL then
2.    allocate a new _intblock;
3.    return one PyIntObject from _intblock;
4. else then
5.    if the current_intblock's remaining integer objects==1 then
6.       move L_Partly to L_Fully;
7.    L_Partly = L_Partly ->next;
8.    return one PyIntObject from L_Partly;
```

- algorithm 2 freeing an integer

```
1. find the _intblock this integer belongs to, and assign the _intblock's
      address to current variable
2. label this PyIntObject as free;
3. if current belongs to L_Fully then
4.    move this current to the first of L_Partly;
5. if current belongs to L_Partly then
6.    if current->free_num==N_INTOBJECTS-1 then
7.    move current to L_Empty;
8.    else then
```

```
9.    while current->free_num<current->next->free_num
10.        exchange(current,current->next);
11. free all the blocks in L_Empty;
```

B. *Characteristics of greedy-based model*

- Make the non-container objects independent from the garbage collection module;
- This method is heuristic. As the number of remaining useful integer objects of the blocks in L_Partly keep non-ascending, the most frequently used blocks will be used with priority. The tail blocks of the L_Partly will be collected gradually.

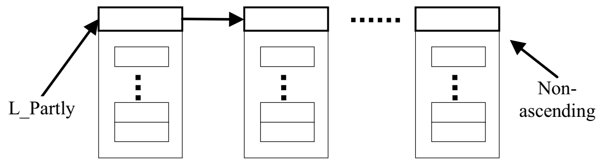

Fig. 2 L_Partly

IV. EXPERIMENT

Through experiment we want to answer the following questions:

1. Does greedy algorithm work?

2. Will Python VM runs slower after this modification?

3. How will this optimization affect web applications written with Django?

A. *Experimental subject*

Because Django is most widely used, enjoy the biggest market share, we choose Django as our experimental subject. Table 1 shows a comparison of project numbers between several widely used Python web frameworks

TABLE 1. A comparison between several widely used Python web framework open source project in PyPi.

Web Framework	Number of Projects
Django	*3547*
Pylons	*190*
TurboGears	*144*
Zope2	*935*

Django's excellent characteristics also make itself competitive with Ruby on Rails. Ben Askins and Alan Green gave a detail comparison between Django and Ruby on Rails[12]. Two skilled programmers implement the same functions with this two different web framework. Their study showed that Django uses fewer lines of code, takes less time. Django also has a more concise template system, the ability to automatically generate admin pages[6]. So Django is competitive in agile web development. In China, the most successfully case is douban, now it has almost 100 million users.

B. *Experiment review*

We used a PC with CPU 1.86GHz, Ubuntu Kernel version 2.6.28 platform. We improve the open source project Unladen-Swallow's benchmark[7]. Then We compare two different page rendering ways: render with HttpResponse(), and render with Template.render(). HttpResponse() is used to render a 4M page and Template.render() is used to render a 150*150 table written with the Django's template language.

In order to obtain Python VM's memory usage comparison of the same time, we wrote an extensive module used to obtain the memory usage. We make instrument before and after the rending expression and then iterate 100 times.

C. *Experiment result*

Physical memory usage can be obtained by reading smaps file which is a new character in 2.6 version kernel. Each time we run benchmark, an accurate memory usage data will be recorded. The Unladen-Swallow monitors the memory through a different way. It will sample the memory usage randomly from the beginning to the end. The following figures show a detailed data comparison. The red line represents initial, the blue one represents the modified one.

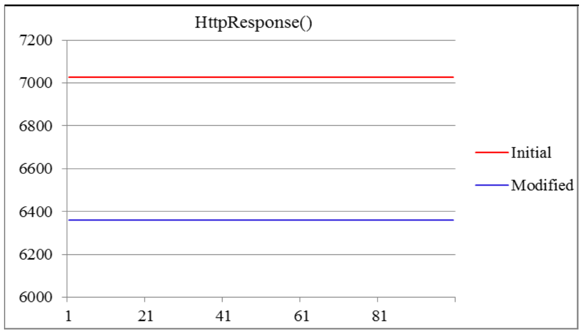

Fig. 3 memory use of HttpResponse() by make instruments

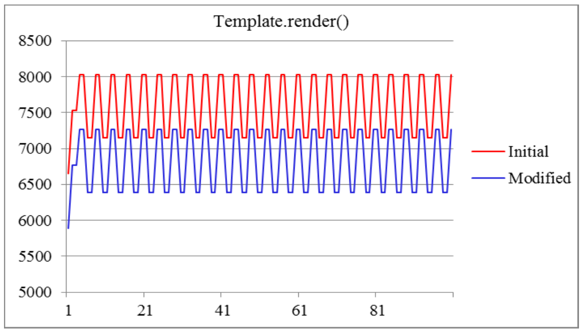

Fig. 4 memory use of Template.render() by make instruments

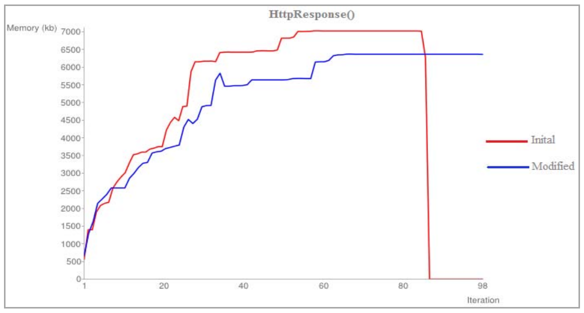

Fig. 5 memory use of HttpResponse() by randomly sampling

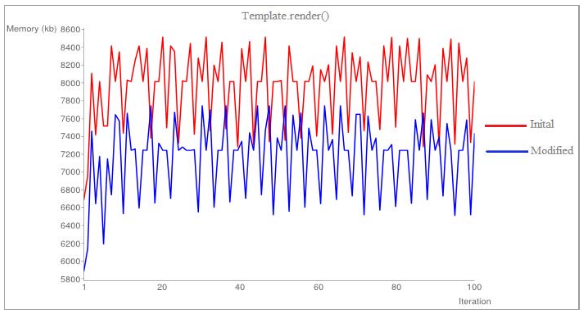

Fig. 6 memory use of Template.render() by randomly sampling

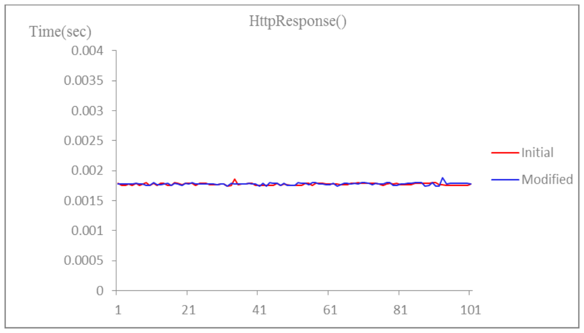

Fig. 7 time cost of HttpResponse()

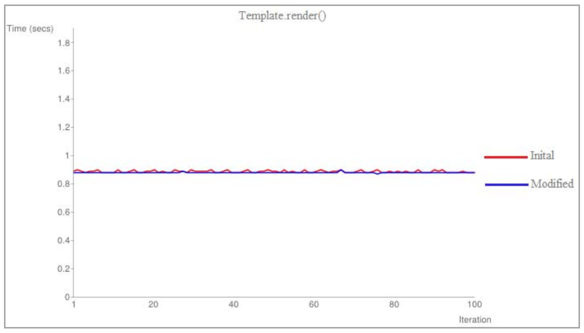

Fig. 8 time cost of Template.render()

D. *Experiment result analysis*

Because physical memory's management is tied with operating system, so figure 4 shows a fluctuation in memory use. Figure 8 saved 10% of memory compared with optimized before. Figure 9 and figure 10 proved that there is barely time cost.

The experimental result has successfully answered the questions answered before.

1. Greedy strategy does work.
2. The Python VM's time efficiency is not influenced.
3. Our methods have improved Django by 10% in memory use.

V. CONCLUTION

In this paper, we propose a greedy based memory management strategy and refined three defects of Python VM's integer management, and then improve the memory usage efficiency of Python VM. We did our experiments on Python web framework Django and save about 10% physical memory. This method can be compatible with many other optimization ways such as the one mentioned in [5].

REFERENCES

[1] C. Barry, M.Lang. A Survey of Multimedia and Web Development Techniques and Methodology Usage, IEEE MultiMedia, 2001, 8(2):52-61.

[2] A. McDonald & R. Welland, 2001, 'A Survey of Web Engineering in Practice', Department of Computing Science Technical Report R-2001-79, University of Glasgow, Scotland, 2001

[3] http://www.python.org/

[4] https://pypi.python.org/pypi

[5] Kazuaki Ishizaki, Priya Nagpurkar, Takeshi Ogasawara, David Edelsohn, Jose Castanos, Toshio Nakatani, Adding Dynamically-Typed Language Support to a Statically-Typed Language Compiler: Performance Evaluation, Analysis, and Tradeoffs. Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments. London, England, UK, 2012. 169-180.

[6] https://www.djangoproject.com/

[7] https://code.google.com/p/unladen-swallow/

[8] José Ignacio Fer nández-Villamor, Laura Díaz-Casillas, Car los Á. Iglesias, A Comparison Model for Agile Web Frameworks. Proceedings of the 2008 Euro American Conference on Telematics and Information System. Aracaju, Brazil, 2008. 1-8.

[9] http://wiki.python.org/moin/WebFrameworks

[10] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. Scikit-learn: Machine Learning in Python. The Journal of Machine Learning Research. 12, 2011:2825-2830.