ТЕХНОТРЕК

Урок четвёртый

Оптимизация производительности

Антон Кухтичев

Посещаемость

Не забудьте отметиться!!!!

Программа на сегодня

1. Мониторинг потребления ресурсов;
2. Профилирование backend;
3. Приёмы оптимизации: кеширование, денормализация.

ТЕХНОТРЕК

Мониторинг потребления ресурсов

top

top - консольная команда, которая выводит список работающих в системе процессов и информацию о них.

`PID` — идентификатор процесса
`USERNAME` — пользователь, от которого запущен процесс
`SIZE` — размер процесса (данные, стек и т. д.) в килобайтах
`RES` — текущее использование оперативной памяти
`VIRT` — полный объем виртуальной памяти, которую занимает процесс
`%CPU` — процент доступного времени процессора

atop

atop — продвинутый интерактивный полноэкранный монитор производительности, написанный для Linux.

Чтение файла:

```
atop -r /var/log/atop/atop_<date>
```

Горячие клавиши:

`t` - переход к следующему временному интервалу;
`Shift + t` - переход к предыдущему временному интервалу;
`Shift + m` - сортировка процессов по занимаемой резидентной памяти;
`Shift + c` - сортировка процессов по потреблению CPU (по умолчанию);
`Shift + d` - сортировка процессов по использованию диска;
`Shift + n` - сортировка процессов по использованию сети;

iotop

iotop — утилита, выводящая данные по использованию жесткого диска.

```
# Посмотрим наиболее активные процессы.
iotop -o
# Собрать статистику за определённое время
iotop -o -a
```

iostat

Утилита, предназначенная для мониторинга использования дисковых разделов.

```
iostat -d -t -p sda -x
```

`-c` — вывести отчёт по CPU;
`-d` — вывести отчёт по использования диска;
`-t` — интервал, за который усредняются значения и вычисляются «средние» значения в секундах;

ТЕХНОТРЕК

Профилирование **backend**

Для чего нужно профилирование?

- Позволяет найти «узкие места» в вашем коде;
- Чем быстрее код, тем больше работы за единицу времени;

Для чего нужно профилирование?

Основные способы - замеры

- CPU
- Память
- Частота/продолжительность вызовов функций

Методы

- Статистический метод (сэмплирование)
- Инструментирование

Python profiler

- ***cProfile*** — относительно новый (с версии 2.5) модуль, написанный на C и оттого быстрый;
- ***profile*** — нативная реализация профайлера (написан на чистом питоне), медленный, и поэтому не рекомендуется к использованию;
- ***hotshot*** — экспериментальный модуль на си, очень быстрый, но больше не поддерживается и в любой момент может быть удалён из стандартных библиотек;

cProfile. Способ I.

```
python -m cProfile -o output.txt ptest.py
```

```
import pstats
p = pstats.Stats("output.txt")

p.strip_dirs().sort_stats(-1).print_stats()
```

cProfile. Способ II.

```
import cProfile, pstats

pr = cProfile.Profile()
pr.enable()
# ... do something ...
pr.disable()

pr.print_stats()
pr.dump_stats(“output.prof”)

apt-get install graphviz

gprof2dot -f pstats output.prof | dot -Tpng -o
output.png
```

Flame graph

- Метод визуализации собранных фреймов стека;
- Введены в обиход Бренданом Греггом (Brendan Gregg);
- Помогают понять общую картину выполнения приложения;
- Работает с разными формата результатов (perf, DTrace и т.д.).

Flame graph. Установка

```
# Установить библиотеку Python flamegraph
pip3 install flamegraph
# Получить логи профилирования для .
python3 -m flamegraph -o perf.log your_script.py
<args>

# Получить svg.
./flamegraph.pl --title "MyScript CPU" perf.log >
perf.svg
```

Кеширование

Виды кеширования

Кэширование означает сохранение результатов дорогостоящего вычисления, чтобы избежать его повторного вычисления в следующий раз.

1. Memcached;
2. Кэширование в базу данных;
3. Кэширование на файловую систему;
4. Кэширование в оперативной памяти;

Memcached

- Все данные хранятся прямо в оперативной памяти;
- Работает как демон и захватывает определённый объём оперативной памяти;
- Нет никакой дополнительной нагрузки на базу данных или файловую систему.

Установка Memcached

```
# Установить библиотеку Python
# для работы с memcached
pip3 install python-memcached
# Установить пакет memcached.
sudo apt-get install memcached

# Запустить демона memcached
# По умолчанию порт 11211
# /etc/memcached.conf
systemctl start memcached
```

Подключение мемкеша

```
CACHES = {
    'default': {
        'BACKEND':
'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

Использование кеша I

```
from django.core.cache import cache

# cache.set(key, value, timeout=DEFAULT_TIMEOUT,
version=None)
>>> cache.set('my_key', 'hello, world!', 30)

# cache.get(key, default=None, version=None)
>>> cache.get('my_key')
'hello, world!'
```

Использование кеша II

```
from django.views.decorators.cache import cache_page

@cache_page(60 * 15)
def my_view(request):
    ...
```

ТЕХНОТРЕК

Денормализация

Когда нужна денормализация?

В запросах к полностью нормализованной базе нередко приходится соединять до десятка, а то и больше, таблиц. А каждое соединение — операция весьма ресурсоемкая.

- Денормализация путем сокращения количества таблиц;
- Денормализация путём ввода дополнительного поля в одну из таблиц.

Домашнее задание №4

1. Прикрутить memcached (4 балла);
2. Наполнить базу искусственными синтетическими данными (доп. 2 балла);
3. Использование профайлера (4 балла);

Срок сдачи

Сроков нет, но вы держитесь

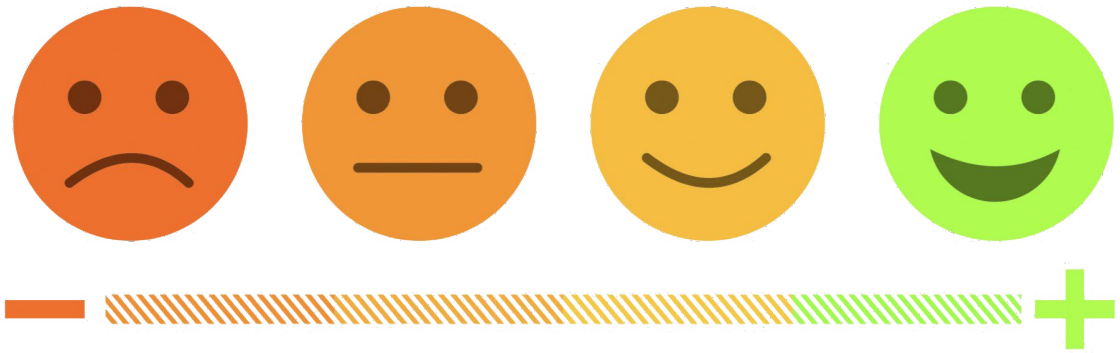
Полезные ссылки

Виды кеширования в Django

Для саморазвития (опционально)
Чтобы не набирать двумя пальчиками

Обратная связь

Не забудьте поставить оценки!!!!



ТЕХНОТРЕК

Спасибо за внимание!

Антон Кухтичев

a.kukhtichev@corp.mail.ru