Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

Тестирование производительности: принципы и практика

Ефремов Михаил Александрович

План лекции

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

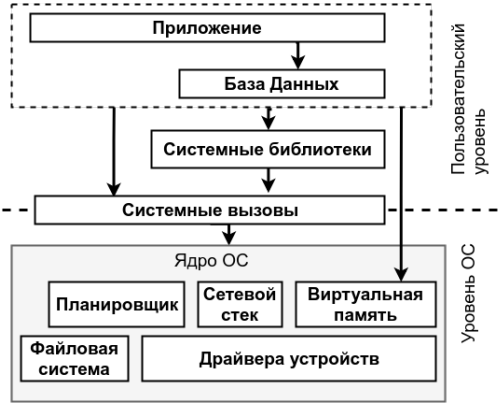
- Определение и характеристика
- Подходы к тестированию
  - Бенчмаркинг
  - Профилирование
- Инструменты и практика
- Тестирование робототехнических фреймворков
- Реализация программного монитора
- Заключение
- Доп. Материалы

Определение

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

Определение

Определение Тестирование производительности – набор мер по изучению и анализу производительности системы, т.е. потребления вычислительных ресурсов.



Что можно измерять?

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

Задержка (latency) – время, которое ожидает операция для того чтобы быть выполненной.

Степень использования (utilization) – мера занятости ресурса, например, загрузка ЦП или объем занимаемой ОП.

Операции ввода-вывода в секунду (IOPS) – скорость работы приложений с периферийными устройствами, например, жесткими дисками.

Пропускная способность (throughput) – количество каких-либо операций в секунду (запросов, обработанных байтов).

Зачем и как измерять?

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

Локализовать и устранить недостатки

- Профилирование
  - gprof – простой во всех смыслах
  - Intel VTune – дорого-богато
  - perf tools – швейцарский нож для профилирования
  - Valgring – слишком тяжелый
- Макробенчмаркинг
  - time – (а почему нет-то?)
  - iostat – IOPS блочных устройств
  - iperf – производительность сети

Сравнение кода между собой

- Микробенчмаркинг (C++)
  - Google Benchmark – большая и функциональная библиотека
  - Hayai – маленький и простой заголовочник
  - x86 rdtsc и chrono – собери бенчмарк сам (сложность зависит от целей)

Методики измерения?

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

Тысячи их:

Methodology	Type
Streetlight anti-method	observational analysis
Random change anti-method	experimental analysis
Blame-someone-else anti-method	hypothetical analysis
Ad hoc checklist method	observational and experimental analysis
Problem statement	information gathering
Scientific method	observational analysis
Diagnosis cycle	analysis life cycle
Tools method	observational analysis
USE method	observational analysis
Workload characterization	observational analysis, capacity planning
Drill-down analysis	observational analysis
Latency analysis	observational analysis
Method R	observational analysis
Event tracing	observational analysis
Baseline statistics	observational analysis
Performance monitoring	observational analysis, capacity planning
Queueing theory	statistical analysis, capacity planning
Static performance tuning	observational analysis, capacity planning
Cache tuning	observational analysis, tuning
Micro-benchmarking	experimental analysis
Capacity planning	capacity planning, tuning

Что важнее: **как делать не надо!**

- Поиск только при свете дня (фонаря) – не надо искать очевидное.
- Тестирование вслепую
- Виноват не я! Не надо искать причину в других людях, а не в коде.

Что мешает?

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

- **СУБЪЕКТИВНОСТЬ И ПОГРЕШНОСТЬ**
- Процессор
  - Кэш (он вообще везде мешает)
  - Конвейер
  - Многоядерность
  - Эвристики (branch prediction)
- Компилятор и его оптимизации

Как это решается?

- Итеративное выполнение (для бенчмаркинга)
- Закон больших чисел и статистика
- Фиксация процесса на ядре, установка одного режима работы ЦП(!)
- Принуждение компилятора (хитростью и напрямую)

Что будем использовать мы?

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

- **Google Benchmark** – библиотека для микробенчмаркинга GitHub

Важно!

В системе должна присутствовать или быть собрана библиотека pthread. Так же убедитесь, что присутствуют утилиты taskset и cpupower.

- **Perf tools** – профилирование

Пакеты для Ubuntu	Пакет для Archlinux и Fedora
linux-tools-generic linux-tools	perf

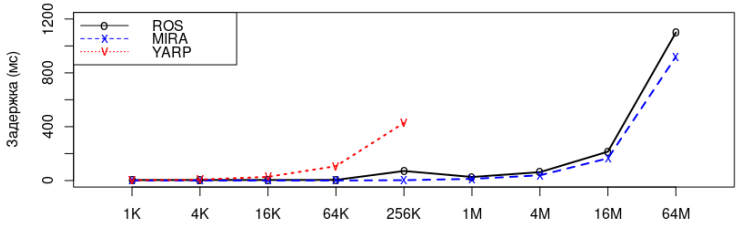
- **Heaptrack** – учет расхода памяти

Пакеты для Ubuntu, Archlinux, Fedora
heaptrack heaptrack_gui

Тестирование производительности робототехнических фреймворков

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

- Постановка задачи
- Выбор инструментов
- Реализация
- Проблемы
- Продолжение реализации
- Результаты



Программный монитор для КМПО

- Предыстория
- Постановка задачи
- Проблемы и решения
- WIP

Определение и характеристика

Инструменты и практика

На опыте

Материалы

Доп. Материалы

Микробенчмаркинг

Память

FlameGraph


```
#define STORE_STATE()   \
__asm__ __volatile__("mov %0, rsp;\n\t" : "=m"(VenusTestLib::MacrosOnly::directInstance._rsp));          \
__asm__ __volatile__("mov %0, rbp;\n\t" : "=m"(VenusTestLib::MacrosOnly::directInstance._rbp));          \
__asm__ __volatile__("mov %0, rbx;\n\t" : "=m"(VenusTestLib::MacrosOnly::directInstance._rbx));          \
__asm__ __volatile__("mov %0, r12;\n\t" : "=m"(VenusTestLib::MacrosOnly::directInstance._r12));          \
__asm__ __volatile__("mov %0, r13;\n\t" : "=m"(VenusTestLib::MacrosOnly::directInstance._r13));          \
__asm__ __volatile__("mov %0, r14;\n\t" : "=m"(VenusTestLib::MacrosOnly::directInstance._r14));          \
__asm__ __volatile__("mov %0, r15;\n\t" : "=m"(VenusTestLib::MacrosOnly::directInstance._r15));          \
__asm__ __volatile__("lea %0, [rip - 0x7];\n\t" : "=r"(VenusTestLib::MacrosOnly::directInstance._rip));  \
__asm__ __volatile__("mov r12, %0;\n\t" : : "m"(VenusTestLib::MacrosOnly::directInstance._r12));         \
__asm__ __volatile__("mov r13, %0;\n\t" : : "m"(VenusTestLib::MacrosOnly::directInstance._r13));         \
__asm__ __volatile__("mov r14, %0;\n\t" : : "m"(VenusTestLib::MacrosOnly::directInstance._r14));         \
__asm__ __volatile__("mov r15, %0;\n\t" : : "m"(VenusTestLib::MacrosOnly::directInstance._r15));         \
__asm__ __volatile__("mov rbx, %0;\n\t" : : "m"(VenusTestLib::MacrosOnly::directInstance._rbx));         \
__asm__ __volatile__("mov rsp, %0;\n\t" : : "m"(VenusTestLib::MacrosOnly::directInstance._rsp));         \
__asm__ __volatile__("mov rbp, %0;\n\t" : : "m"(VenusTestLib::MacrosOnly::directInstance._rbp));         \
#define RESTORE_STATE()                                                                                  \
__asm__ __volatile__("jmp %0;\n\t" : : "m"(VenusTestLib::MacrosOnly::directInstance._rip));  \
```

Книги и доклады

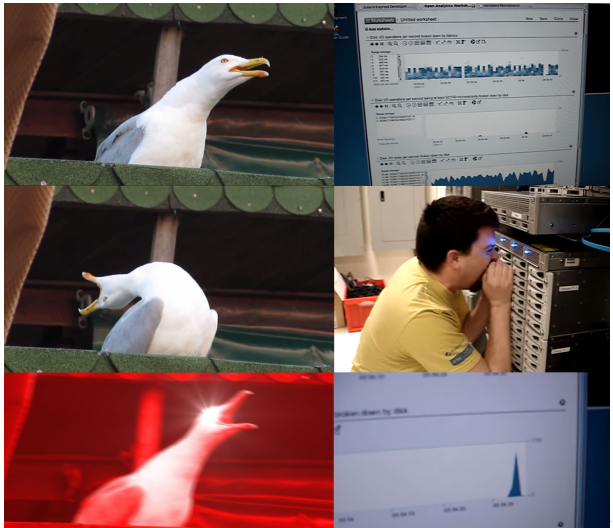
Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

- Главное действующее лицо: Brendan Gregg (Netflix) с сайтом и блогом и книгой Systems Performance Enterprise and the Cloud (2014)
- Chandler Carruth (Google) ▸ Tuning C++: Benchmarks, and CPUs, and Compilers! Oh My!
- Александр Алексеев (Postgres Professional) ▸ Профилирование кода на C/C++ в *nix-системах
- Кирилл Борисов (Яндекс) ▸ Flame graph новый взгляд на привычное профилирование

Заключение

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

Тестирование производительности ресурсозатратно, дорого, требует много знаний и сил... но порой и очень интересно



Функции, спасающие от слишком умного компилятора

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

1. Помогает сохранить какой-то объект, не дать компилятору в ходе оптимизации избавиться от объекта:

```
static void escape(void *p) {
    asm volatile ("" : : "g"(p) : "memory");
}
```

2. Похожая функция, указывающая, что в момент исполнения возможна запись в любую область памяти, что помогает не дать компилятору избавиться от «бесполезных» операторов:

```
static void clobber() {
    asm volatile ("" : : : "memory");
}
```

Пример кода теста example.cpp

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

```
#include "benchmark.h"
#include <vector>

static void escape(void *p) { asm volatile ("" : : "g"(p) : "memory"); }
static void clobber() { asm volatile ("" : : : "memory"); }

static void bm_create(benchmark::State &state) {
   while (state.KeepRunning()) {
      std::vector<int> v;
      escape(&v);
      (void) v;   } } BENCHMARK(bm_create);

static void bm_reserve(benchmark::State &state) {
   while (state.KeepRunning()) {
      std::vector<int> v;
      v.reserve(1);
      escape(v.data());   } } BENCHMARK(bm_reserve);

static void bm_push_back(benchmark::State &state) {
   while (state.KeepRunning()) {
      std::vector<int> v;
      escape(v.data());
      v.push_back(42);
      clobber();   } } BENCHMARK(bm_push_back);

BENCHMARK_MAIN();
```

Как скомпилировать для Google Benchmark правильно?

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

- -O2 или -O3 флаг, ибо в реальности ваше приложение будет компилироваться с применением оптимизаций, а значит тестировать надо уже с ними (и с ними же бороться).
- -fno-omit-frame-pointer – позволяет сохранять границы кадров стека, что требуется для адекватного отображения результата профилирования в perf report

Итоговый минимальный makefile:

```
example: example.cpp
	g++ example.cpp -L. -lbenchmark -lpthread -O3 -fno-omit-frame-pointer -o example
```

На всякий случай: в системе должна быть установлена, либо находиться рядом библиотека pthread, собственно библиотека benchmark.a, а так же ее заголовочный файл benchmark.h.

Запуск теста и просмотр результата

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

1. Используем perf record для записи результата:

```
perf record -g ./example
```

2. Используем perf report для удобного просмотра результирующего дерева вызовов функций (пробелы после запятых не нужны!):

```
perf report -g "graph,0.5,caller"
```

С интерфейсом предлагается разобраться самостоятельно, основываясь на предложенном видео выступления Чендлера. Стоит отметить, что при повторном просмотре ассемблерного кода perf может отматываться в самый низ ассемблера, т.о. требуется просто прокрутить экран вверх.

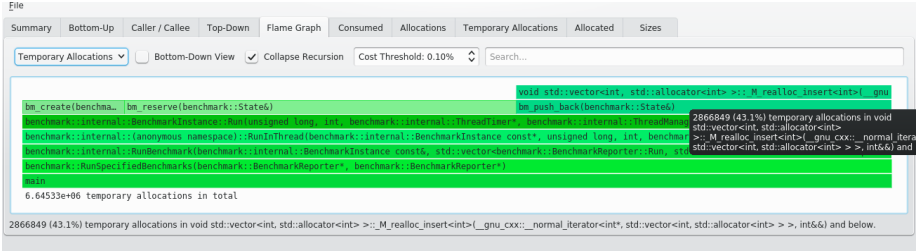
Профилирование по памяти

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

Для записи выделения и освобождения памяти, а так же проблем связанных с данными процессами используется программа heaptrack. Чтобы явным образом ее опробовать требуется, соответственно, использовать операторы new и delete в коде. Запустить профилирование можно следующей командой:

```
heaptrack ./example
```

Для просмотра результатов рекомендуется использовать утилиту heapsort_gui.



FlameGraph – наглядное отображение результатов профилирования

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

- Основное:
  - снизу вверх располагаются кирпичики-блоки функции в порядке уровня вложенности
  - блоки вызванных функций располагаются строго на блоках их вызвавших
  - на одном уровне блоки расположены по ширине соответственно использованию рассматриваемого ресурса
- Нужные perl-скрипты располагаются в репозитории GitHub
  - stackcollapse-perf.pl – для работы со срезами стека из perf
  - flamegraph.pl – для преобразования полученной из предыдущего скрипта структуры данных интерактивного svg-изображения с искомым флеймграфом.

Построение FlameGraph-а

Определение и характеристика
Инструменты и практика
На опыте
Материалы
Доп. Материалы
Микробенчмаркинг
Память
FlameGraph

1. Сохраняем в директорию с результатами профилирования perf.data вышеупомянутые perl-скрипты. Не забываем дать им возможность исполняться, выполнив chmod +x на файлах скриптов.
2. Получаем срезы стека из perf:

```
perf script > perf.script
```

3. Обрабатываем первым скриптом:

```
./stackcollapse-perf.pl perf.script > perf.folded
```

4. Получаем svg-файл вторым скриптом:

```
./flamegraph.pl perf.folded > example.svg
```