ORACLE

Performance 101
метод. опт. прозв.

Aleksey Shipilëv
aleksey.shipilev@oracle.com, @shipilev

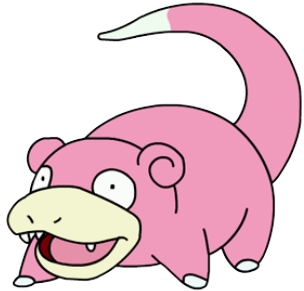
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Intro

Intro: стандартный дисклеймер

1. Computer Science → Software Engineering
   - Строим приложения по функциональным требованиям
   - В большой степени абстрактно, в «идеальном мире»
   - Рассуждения при помощи формальных методов

2. Performance Engineering
   - «Real world strikes back!»
   - Исследуем взаимодействия софта с железом на типичных данных
   - Эффективно предсказывается уже мало что
   - Рассуждения при помощи формальных методов



У меня всё медленно!

Что делаем в первую очередь?

Intro: Классические ошибки первого шага

Неправильно:

- «Я вижу, что метод `foo()` реализован неэффективно»
- «По профилю видно, что метод `bar()` – горячий и занимает 5%»
- «По-моему, у нас тормозит БД, и нужно перейти с DBMS_x на DBMS_y»

Intro: Классические ошибки первого шага

Неправильно:

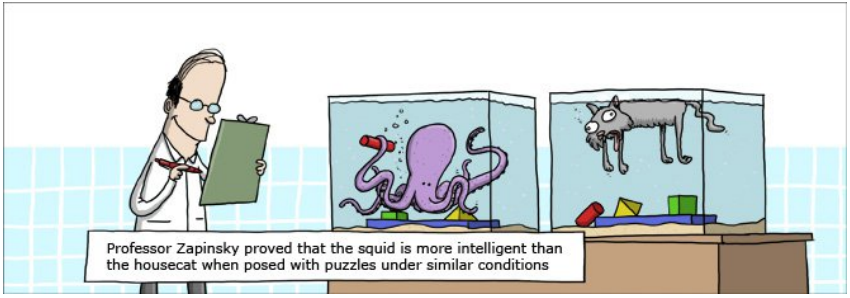
- «Я вижу, что метод `foo()` реализован неэффективно»
- «По профилю видно, что метод `bar()` – горячий и занимает 5%»
- «По-моему, у нас тормозит БД, и нужно перейти с DBMS_x на DBMS_y»

Правильно:

1. Выбрать метрику: ops/sec, txs/sec, время исполнения, время отклика, ...
2. Убедиться в корректности метрики: релевантность, повторяемость, ...
3. Поставить целью улучшение метрики

Определения

Определения: Эксперименты



Никуда не деться от хорошего эксперимента, который должен быть:

- **Релевантным**: должен воспроизводить нужное нам поведение
- **Изолированным**: должен исключить посторонние эффекты

Определения: Эксперименты

Никуда не деться от хорошего эксперимента, который должен быть:

- **Измеримым**: должен давать исчислимые метрики
- **Воспроизводимым**: должен давать постоянные результаты

Определения: Метрики

Две главные группы метрик:

Bandwidth (λ)

- Сравнительно легко измерить
- Легко вводится в steady state
- Скрывает большие задержки
- Упускает active-idle

Latency (τ)

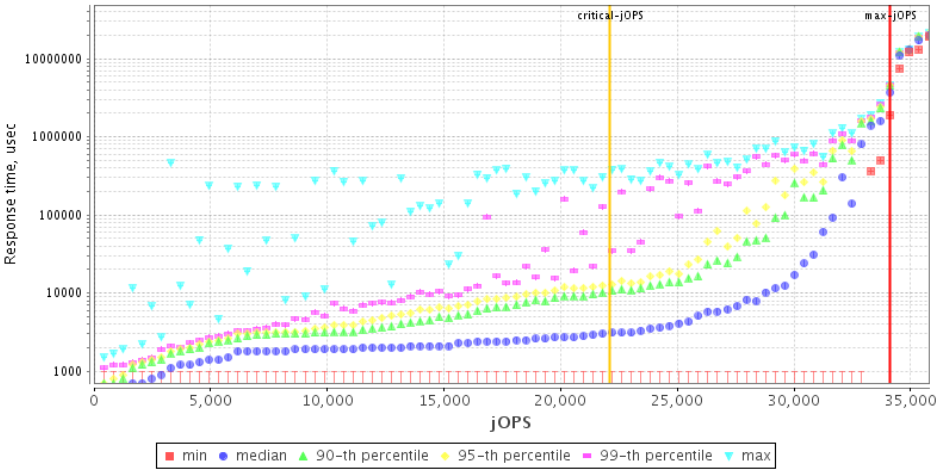
- Трудно измерить корректно
- Средняя latency как правило бессмыслена
- Нужно считать квантили
- Семплы меньше 1мкс получить не практично

Определения: путаем bandwidth и latency

- Отличия в нагрузке:
  - λ обычно измеряют под пиковой нагрузкой
  - τ как правило измеряют под регулируемой нагрузкой
  - Нагрузка – это новая степень свободы!

- Отличия в подходах к измерению:
  - Средняя задержка – бессмысленная метрика, ибо $\tau_{avg} = \frac{1}{\lambda}$
  - Нужно мерить отдельные события и считать квантили!
  - События короче 1 мкс? Мухаха.

Определения: λ и τ – братья навек



Java

Определения: Закон Литтла

$$L = \lambda\tau$$

L – количество обслуживаемых клиентов,
λ – пропускная способность,
τ – время обслуживания

Следствия:

- при $L = const$, $\lambda \sim \frac{1}{\tau}$ или $\tau \sim \frac{1}{\lambda}$
- при известных λ и τ можно оценить L.

Определения: Утилизация

Утилизация = насколько ресурс занят?

$$Utilization = \frac{ResourceBusyTime}{TotalTime}$$

Простой = насколько ресурс свободен?

$$Idle = 1 - Utilization$$

Определения: Эффективность

Эффективность = часть времени, потраченная на полезную работу

- Оценка КПД для времени, её невозможно строго вычислить
- **Высокая утилизация $\neq$ высокая эффективность**

High Utilization **does not imply** High Efficiency

Java

Определения: SpeedUp

«A в N раз быстрее B» означает:

$$SpeedUp = \frac{time_B}{time_A} = \frac{throughput_A}{throughput_B}$$

Следствия:

- «Один раз – не спидап»
- Спидап всегда неотрицательный
- Разные масштабы по разные стороны от единицы:
  «A быстрее B в 50 раз» $\sim$ «A быстрее B в 0.02 раза»

Определения: Boost%

«A на n% быстрее B» означает:

$$Boost_{\%} = \frac{time_B - time_A}{time_B} * 100\%$$

$$Boost_{\%} = \frac{thr_A - thr_B}{thr_B} * 100\%$$

$$Boost_{\%} = (SpeedUp - 1) * 100\%$$

Следствия:

- Нулевой буст = нулевые улучшения/деградации
- Та же проблема с масштабом по обе стороны нуля

Определения: Про масштабы

Test	Speedup	Boost
текущее состояние	1.0x	0.0%

Определения: Про масштабы

Test	Speedup	Boost
текущее состояние	1.0x	0.0%
ковыряние палочкой X	1.5x	+50.0%

Определения: Про масштабы

Test	Speedup	Boost
текущее состояние	1.0x	0.0%
ковыряние палочкой X	1.5x	+50.0%
багфикс Y	0.82x	-18.0%

Определения: Про масштабы

Test	Speedup	Boost
текущее состояние	1.0x	0.0%
ковыряние палочкой X	1.5x	+50.0%
багфикс Y	0.82x	-18.0%
улучшение N	1.2x	+20.0%

Не, ну а чего: накатим багфикс Y, а потом починим перформанс с улучшением N. Потеряли 18%, а вернули целых 20%, на эти 2% и живём.

Определения: Про масштабы

Test	Speedup	Boost
текущее состояние	1.0x	0.0%
ковыряние палочкой X	1.5x	+50.0%
багфикс Y	0.82x	-18.0%
улучшение N	1.2x	+20.0%

Не, ну а чего: накатим багфикс Y, а потом починим перформанс с улучшением N. Потеряли 18%, а вернули целых 20%, на эти 2% и живём.

На самом деле, $0.82 * 1.2 = 0.984$,
т.е. мы остались с 1.5% замедлением.

Закон Амдала: Pop quiz

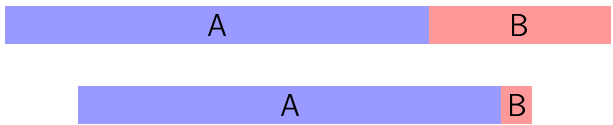
Представим себе приложение с двумя отдельными частями:

- Часть A занимает 70% времени, разгоняема в 2 раза
- Часть B занимает 30% времени, разгоняема в 6 раз
- Какую часть будем разгонять?

Закон Амдала: Pop quiz

Представим себе приложение с двумя отдельными частями:

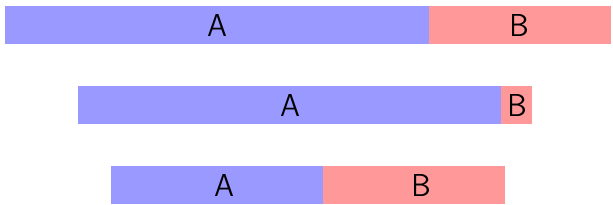
- Часть A занимает 70% времени, разгоняема в 2 раза
- Часть B занимает 30% времени, разгоняема в 6 раз
- Какую часть будем разгонять?



Закон Амдала: Pop quiz

Представим себе приложение с двумя отдельными частями:

- Часть A занимает 70% времени, разгоняема в 2 раза
- Часть B занимает 30% времени, разгоняема в 6 раз
- Какую часть будем разгонять?



Закон Амдала: вывод

$$Part_A = \frac{A}{A+B}$$
$$Part_B = \frac{B}{A+B}$$

Закон Амдала:

$$S = \frac{A+B}{\frac{A}{S_A}+B} = \frac{1}{\frac{Part_A}{S_A}+Part_B}$$

Следствия:

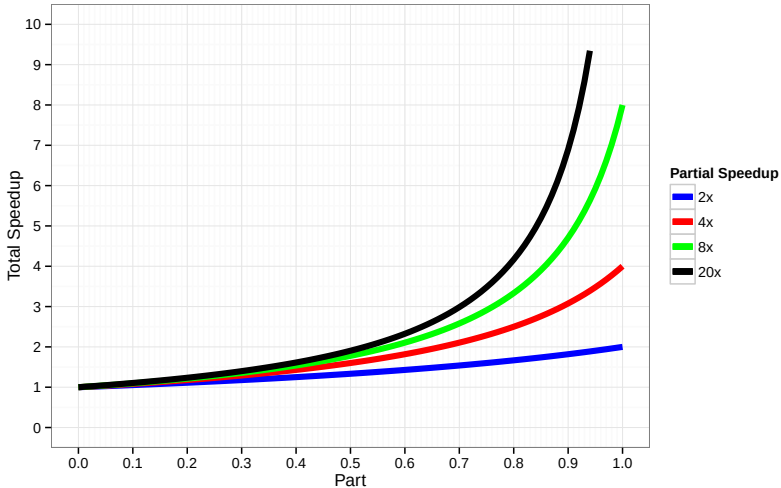
$$\lim_{Part_A \to 0} S = 1$$

$$\lim_{Part_A \to 1} S = S_A$$

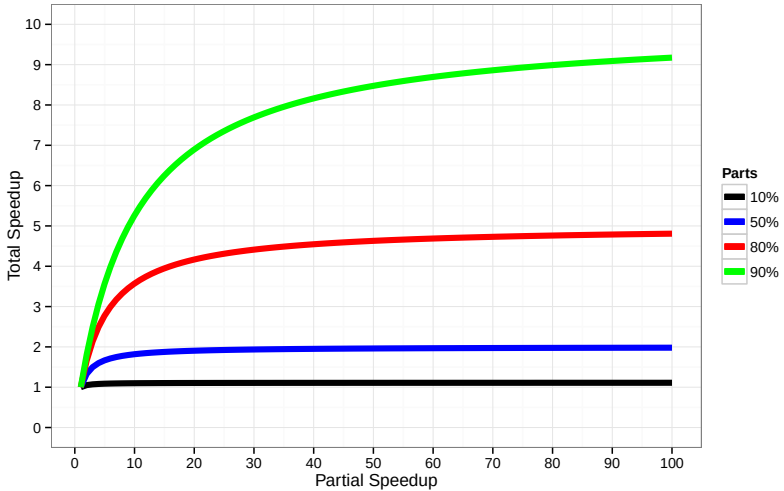
$$\lim_{S_A \to 0} S = 0$$

$$\lim_{S_A \to \infty} S = \frac{1}{Part_B}$$

Закон Амдала: поведение



Закон Амдала: поведение



Закон Амдала: обобщение

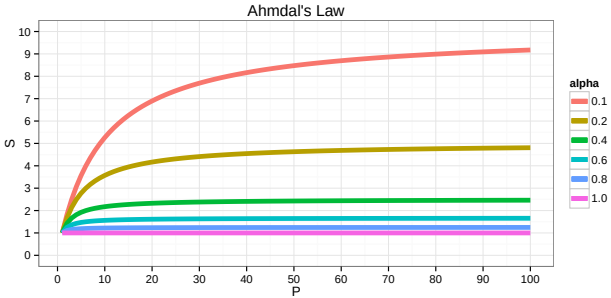
Немножко поиграем членами:

$$S = \frac{1}{\frac{P_A}{S_A} + P_B} = \frac{1}{\frac{1-P_B}{S_A} + P_B} = \frac{S_A}{1 - P_B + P_B S_A} = \frac{S_A}{1 + P_B(S_A - 1)}$$

или, после подстановки
$p = S_A$ (во сколько раз добавили ресурса),
$\alpha = P_B$ (сколько весит всё остальное):

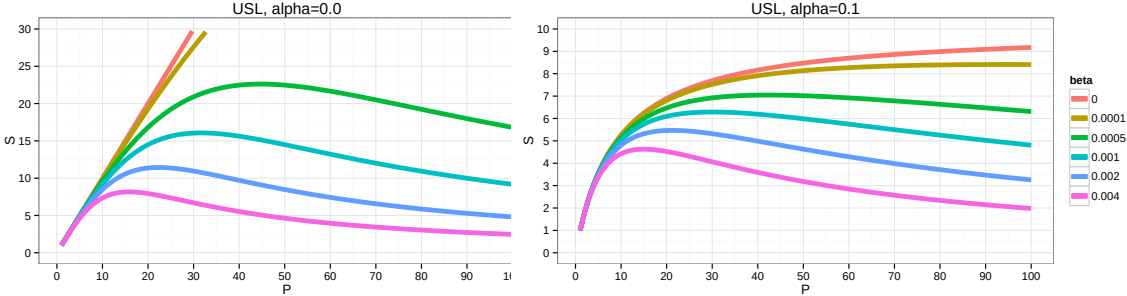
$$S = \frac{p}{\underbrace{1}_{concurrency} + \underbrace{\alpha(p-1)}_{contention}}$$

Закон Амдала: поведение



$$S = \frac{p}{\underbrace{1}_{concurrency} + \underbrace{\alpha(p-1)}_{contention}}$$

USL: Universal Scalability Law



$$S = \frac{p}{\underbrace{1}_{concurrency} + \underbrace{\alpha(p-1)}_{contention} + \underbrace{\beta p(p-1)}_{coherence}}$$

USL: Universal Scalability Law

$$S = \frac{p}{\underbrace{1}_{concurrency} + \underbrace{\alpha(p-1)}_{contention} + \underbrace{\beta p(p-1)}_{coherence}}$$

Наблюдения:

- USL хорошо натягивается на эмпирические данные
- При $\beta > 0$, с добавлением ресурсов не то, что может не становиться лучше, может становиться **хуже**
- Систем с $\alpha = 0$ и $\beta = 0$ практически не существует

USL: $\alpha, \beta > 0$

Предположим, что есть функциональные блоки A и B. Есть ли разница при параллельном и последовательном исполнении?

- С точки зрения функциональности, абстракция «чёрного ящика» – результат зависит только от входов и внутреннего состояния:

$$Result(A||B) \sim Result(A \rightarrow B)$$

- С точки зрения производительности, никто точно не знает!

$$Perf(A||B) \ ??? \ Perf(A \rightarrow B)$$

Подходы

Подходы: второй шаг, ошибки

«Я вижу, что метод `foo()` реализован неэффективно»

Подходы: второй шаг, ошибки

«Я вижу, что метод `foo()` реализован неэффективно»

- ...а метод не используется вообще
- ...или используется, но занимает совсем чуть общего времени
- ...или используется активно, *но дело не в этом*

Подходы: второй шаг, ошибки

«По профилю видно, что метод `bar()` – горячий и занимает 5%»

- ...а оказывается, что это всего 5% от всего приложения, которое занимает 6.25% CPU 16-процессорной системы
- ...или это метод, помогающий всем остальным быть быстрее
- ...или он действительно проблемный, *но дело не в этом*

Подходы: второй шаг, ошибки

«Полносистемная профилировка показывает, что у нас тормозит БД, и нужно срочно перейти с DBMS_x на DBMS_y»

Подходы: второй шаг, ошибки

«Полносистемная профилировка показывает, что у нас тормозит БД, и нужно срочно перейти с DBMS_x на DBMS_y»

- ...а вдруг оказывается, что диски слабоваты
- ...или оказывается, что злые админы зашейпили сеть
- ...или просто БД уже давно никто не «пылесосил»

Подходы: анти-метод уличного фонаря

«Ищу ключи под фонарём, а не там где их потерял – тут светлее»

1. Выбираем любимый профайлер
2. Смотрим в любимый профиль в любимом профайлере
3. Чиним проблемы, которые мы умеем решать

Подходы: как ускорить приложение?

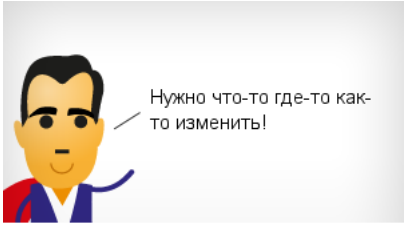
Подходы: как ускорить приложение?

Что?

Где?

Как?

Подходы: как ускорить приложение?



Что мешает работать быстрее?

Где это находится?

Как это исправить?

Java

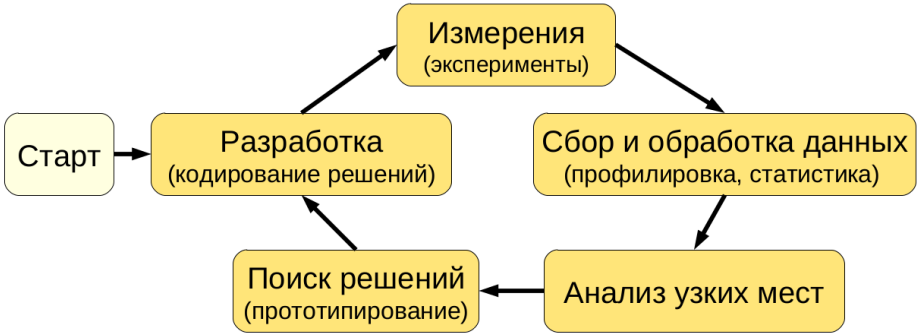
Подходы: как ускорить приложение?

Что мешает работать быстрее?
Используем голову и monitoring tools

Где это находится?
Используем голову и profiling tools

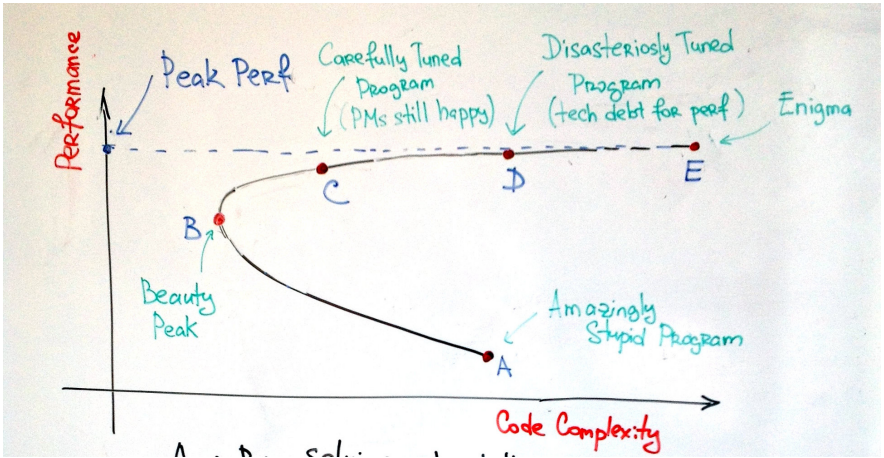
Как это исправить?
Используем голову и холодные руки

Подходы: Итерации



- Новая фаза только после пройденных функциональных тестов
- Одно изменение за цикл
- Каждая итерация должна быть задокументирована

Подходы: Performance Work Phase Diagram



https://twitter.com/shipilev/status/578193813946134529

Java

Методологии

Методологии: супержадные методы

Прямой обход всего пространства ресурсов:

Пространство ресурсов в общем случае N-мерное:

$$K^n$$

Производительность – это скалярное поле над K^n:

$$P : K^n \to R$$

1. Если нет комбинаторного взрыва, то и думать не надо
2. Попробовали все конфигурации, выбрали лучшую, деплойнули
3. Всем радоваться полчаса

Методологии: жадные методы

Локальный поиск лучшего телодвижения:

Масштабируемость – это градиент производительности:

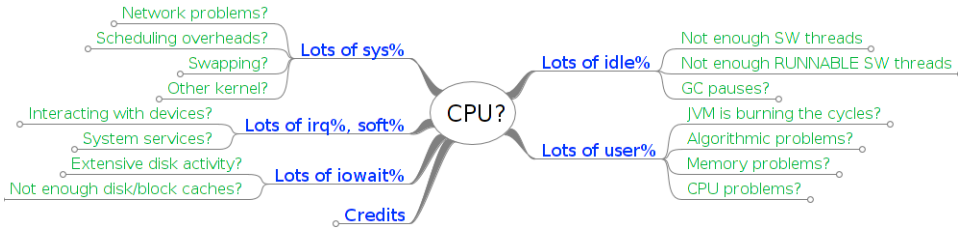
$$S = \nabla P$$

Масштабируемость по ресурсу – это рост P в конкретном направлении:

$$S_i = \frac{\partial P}{\partial R_i}$$

1. Оценим в текущей точке, вырастет ли P, если добавить ресурса
2. Добавляем ресурса, радуемся, обратно на п.1.
3. Пришли в локальный минимум? Не радуемся.

Методологии: workload characterization



- Выбрали сценарий, метрику, запустили нагрузочный тест
- Смотрим на CPU и копаем в нужную сторону

```
# mpstat
CPU    %usr   %sys  %iowait  %irq  %idle
all   44.06  21.42    10.71  0.00  23.76
```


Методологии: USE[1]

Для каждого ресурса проверить утилизацию, насыщение, ошибки

- **Utilization**: среднее время загруженности ресурса
- **Saturation**: количество работы, которую ресурс не может обработать
- **Errors**: ошибки, зарепорченные ресурсом

Задача:
найти утилизированный/насыщенный/погрязший
в ошибках ресурс и помочь ему

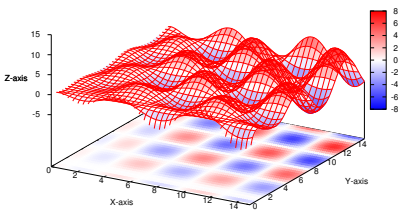
[1] `http://www.brendangregg.com/usemethod.html`

Методологии: USE, ресурсы

1. Уровень системы
   - Сеть
   - Диск
   - Операционная система
   - Процессор/память
2. Уровень приложения
   - Алгоритмы и структуры данных
   - Блокировки, синхронизация
   - Потоки и процессы
3. Микроархитектурный уровень
   - Cache optimizations
   - Branch prediction
   - Other pipeline events

Методологии: нежадные методы

Построение перформансной модели:



1. Предположить вид пространства
2. Построить проверяющие эксперименты
3. Не получилось? Обратно на п.1
4. Получилось? Прыгаем в локальный максимум

Java

Ресурсы

Ресурсы: основные инструменты

- Мозг
  - Плагин «данунеможетбыть» для проверок и перепроверок фактов
  - Плагин «щапридумаем» для построения гипотез и способов их проверки
  - Плагин «чётоянепонял» для проверки консистентности гипотез
  - Плагин «ядурак» для лёгкого отвержения ложных гипотез
- Руки
  - Прямого профиля, для постановки аккуратных экспериментов
  - Сильного профиля, для обработки тонн экспериментальных данных
- Язык, уши, глаза и прочее I/O
  - Для обмена результатами и peer review
  - Для доступа к предыдущим экспериментам

Ресурсы: дополнительные эксперименты

- Профили приложений:
  VisualVM, Java Mission Control, Sun Studio Performance Analyzer
- Профили системы:
  top, vmstat, mpstat, iostat, dtrace, strace
- Профили JRE:
  `-XX:+PrintCompilation`, `-verbose:gc`, `-verbose:class`
- Системные счётчики:
  JMH perfasm/perfnorm, Sun Studio Performance Analyzer, oprofile

http://shipilev.net/#performance-101

