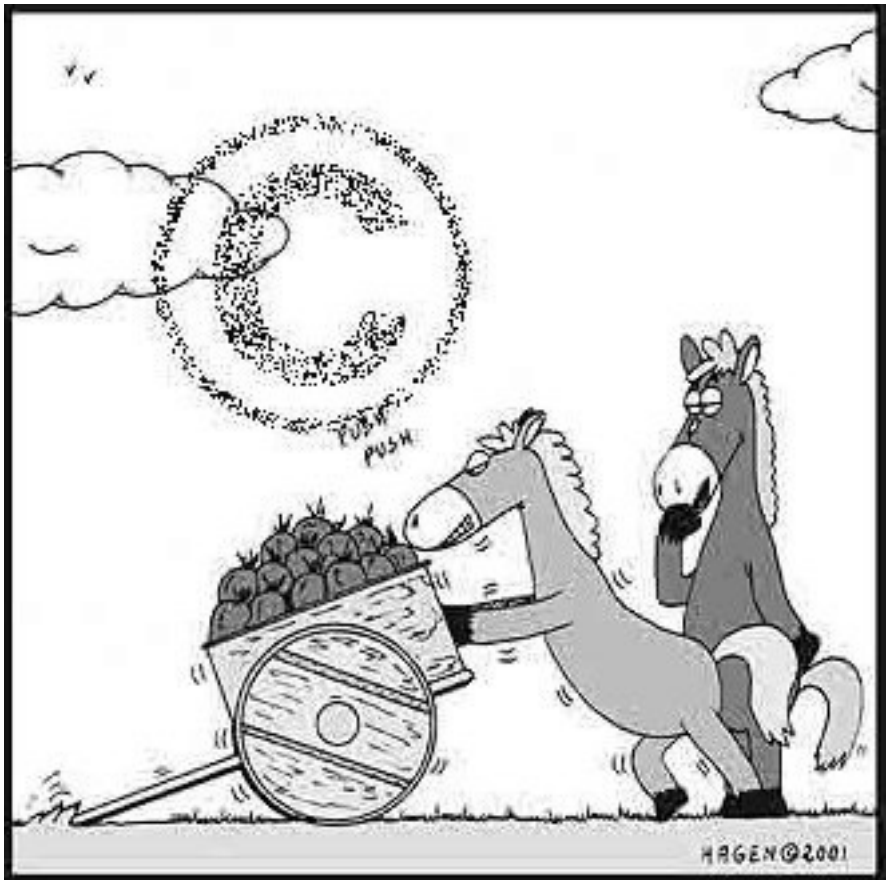
РЕФАКТОРИНГ
И ВТОРОЕ РОЖДЕНИЕ ПРОЕКТА

на примере Zend Framework 2.0

Докладчик: Алексей Пархоменко
alexey@iprojects.com.ua

[Теория]

Что такое рефакторинг?



— изменение исходного кода программы без изменения его внешнего поведения.

Проблемы рефакторинга

- потребность вносить изменения в существующий код;
- необходимость строго придерживаться поставленной задачи;
- покрывать код проверочными тестами;

Нужен ли Вам рефакторинг?

Нужен ли Вам рефакторинг?

- Ваш программный продукт работает, но внесение новой функциональности иногда затягивается на недели;

Нужен ли Вам рефакторинг?

- Ваш программный продукт работает, но внесение новой функциональности иногда затягивается на недели;
- В определенных местах Ваш код работает совершенно не так, как Вы того ожидали;

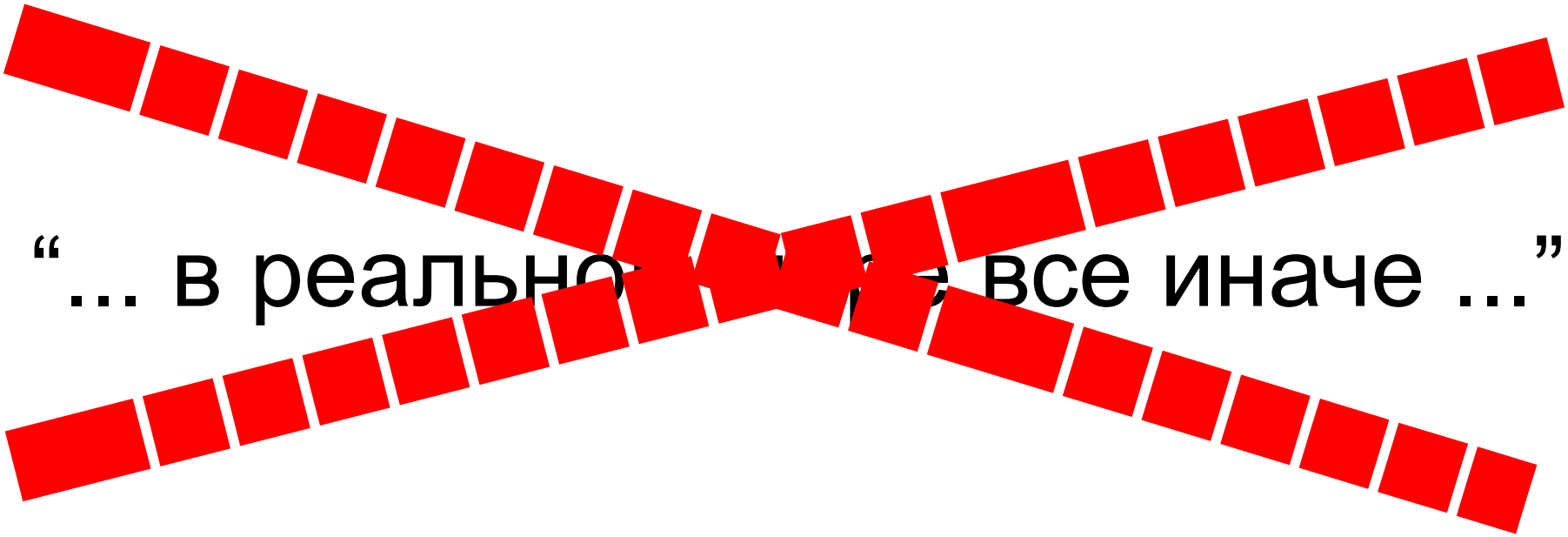
Нужен ли Вам рефакторинг?

- Ваш программный продукт работает, но внесение новой функциональности иногда затягивается на недели;
- В определенных местах Ваш код работает совершенно не так, как Вы того ожидали;
- Вы часто ошибаетесь в сроках реализации поставленной задачи;

Нужен ли Вам рефакторинг?

- Ваш программный продукт работает, но внесение новой функциональности иногда затягивается на недели;
- В определенных местах Ваш код работает совершенно не так, как Вы того ожидали;
- Вы часто ошибаетесь в сроках реализации поставленной задачи;
- Вам приходится вносить однотипные изменения в разных местах проекта;

“... в реальном мире все иначе ...”


“... в реальности все иначе ...”

Рефакторинг всегда подразумевает,

что вы производите его с какой-то целью и опираетесь на какие-то данные строго преследуя поставленные задачи.

Например

Например:

- Улучшение читаемости кода;

Например:

- Улучшение читаемости кода;
- Уменьшение зависимостей между определенными частями системы;

Например:

- Улучшение читаемости кода;
- Уменьшение зависимостей между определенными частями системы;
- Оптимизация каких-либо компонентов системы вследствие замены архитектурных решений;

Например:

- Улучшение читаемости кода;
- Уменьшение зависимостей между определенными частями системы;
- Оптимизация каких-либо компонентов системы вследствие замены архитектурных решений;
- и так далее;

На каждом этапе вы должны четко знать несколько вещей:

- Зачем это нужно делать?
- Какую проблему вы решаете?
- Что на выходе вы должны получить?

Какие есть методы рефакторинга?

- Инкапсуляция поля (Encapsulate Field);
- Выделение класса (Extract Class);
- Выделение интерфейса (Extract Interface);
- Выделение локальной переменной (Extract Local Variable);
- Выделение метода (Extract Method);
- Генерализация типа (Generalize Type);
- Встраивание (Inline);
- Введение фабрики (Introduce Factory);
- Введение параметра (Introduce Parameter);
- Подъём поля/метода (Pull Up);
- Спуск поля/метода (Push Down);
- Замена условного оператора полиморфизмом (Replace Conditional with Polymorphism);
- и так далее;

Вспомним о тестах и их преимуществах:

Вспомним о тестах и их преимуществах:

- Тесты предотвращают появление ошибок в новом коде

Вспомним о тестах и их преимуществах:

- Тесты предотвращают появление ошибок в новом коде
- Тесты позволяют рефакторить код без риска его сломать

Вспомним о тестах и их преимуществах:

- Тесты предотвращают появление ошибок в новом коде
- Тесты позволяют рефакторить код без риска его сломать
- Тесты могут использоваться в качестве документации

Вспомним о тестах и их преимуществах:

- Тесты предотвращают появление ошибок в новом коде
- Тесты позволяют рефакторить код без риска его сломать
- Тесты могут использоваться в качестве документации
- Тесты улучшают дизайн кода

Вспомним о тестах и их преимуществах:

- Тесты предотвращают появление ошибок в новом коде
- Тесты позволяют рефакторить код без риска его сломать
- Тесты могут использоваться в качестве документации
- Тесты улучшают дизайн кода
- Тесты способствуют повышению квалификации разработчиков

Вспомним о тестах и их преимуществах:

- Тесты предотвращают появление ошибок в новом коде
- Тесты позволяют рефакторить код без риска его сломать
- Тесты могут использоваться в качестве документации
- Тесты улучшают дизайн кода
- Тесты способствуют повышению квалификации разработчиков
- Тесты ускоряют процесс разработки

Вспомним о тестах и их преимуществах:

- Тесты предотвращают появление ошибок в новом коде
- Тесты позволяют рефакторить код без риска его сломать
- Тесты могут использоваться в качестве документации
- Тесты улучшают дизайн кода
- Тесты способствуют повышению квалификации разработчиков
- Тесты ускоряют процесс разработки

Мифы о тестах:

В нашем приложении внедрить тесты невозможно, у нас слишком быстро все меняется;

Мифы о тестах:

В нашем приложении внедрить тесты невозможно, у нас слишком быстро все меняется;

В нашем приложении такая архитектура, что нам требуется много mock объектов;

Мифы о тестах:

В нашем приложении внедрить тесты невозможно, у нас слишком быстро все меняется;

В нашем приложении такая архитектура, что нам требуется много mock объектов;

Написание тестов отнимает слишком много времени;

Мифы о тестах:

В нашем приложении внедрить тесты невозможно, у нас слишком быстро все меняется;

В нашем приложении такая архитектура, что нам требуется много mock объектов;

Написание тестов отнимает слишком много времени;

Мы не делаем код для “академиков”, нам нужны решения проще;

Мифы о тестах:

В нашем приложении внедрить тесты невозможно, у нас слишком быстро все меняется;

В нашем приложении такая архитектура, что нам требуется много mock объектов;

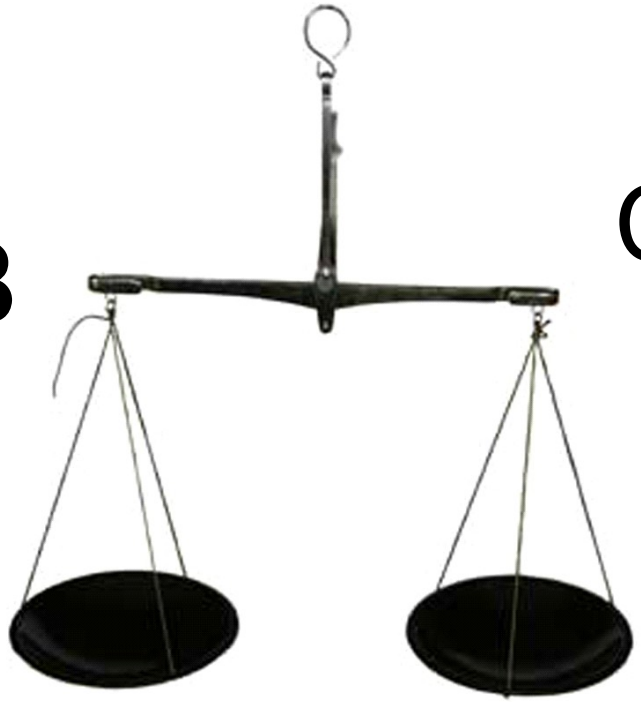
Написание тестов отнимает слишком много времени;

Мы не делаем код для “академиков”, нам нужны решения проще;

Рефакторинг, как методика весьма дорогая для нашей компании. Лучше тратить время программистов на создание новой функциональности;

Рефакторинг не существует без:

Тестов



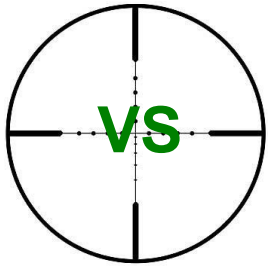
Сопутствующих Инструментов

иначе это не рефакторинг и ваши действия никак не направлены на улучшение существующего кода.

[История]

Под прицелом:

Zend Framework
1.11.11
2011-09-29 релиз



Zend Framework
2.0.0beta3
релиз 2012-03-02

Для ZF2 были поставлены цели:

“Zend Framework 2.0's focus in on improving consistency and perfomance”

(фреймворк сфокусирован на улучшении согласованности и производительности)

Основные проблемы ZF1:

- Неудовлетворительная производительность;

Основные проблемы ZF1:

- Неудовлетворительная производительность;
- Дублирование некоторой функциональности используемых загрузчиков плагинов (plugin loader);

Основные проблемы ZF1:

- Неудовлетворительная производительность;
- Дублирование некоторой функциональности используемых загрузчиков плагинов (plugin loader);
- Слишком много ответственности выдано View;

Основные проблемы ZF1:

- Неудовлетворительная производительность;
- Дублирование некоторой функциональности используемых загрузчиков плагинов (plugin loader);
- Слишком много ответственности выдано View;
- Один тип исключения (Exception) на компоненту;

Основные проблемы ZF1:

- Неудовлетворительная производительность;
- Дублирование некоторой функциональности используемых загрузчиков плагинов (plugin loader);
- Слишком много ответственности выдано View;
- Один тип исключения (Exception) на компоненту;
- Слишком много путей для реализации одних и тех же задач;

Основные проблемы ZF1:

- Неудовлетворительная производительность;
- Дублирование некоторой функциональности используемых загрузчиков плагинов (plugin loader);
- Слишком много ответственности выдано View;
- Один тип исключения (Exception) на компоненту;
- Слишком много путей для реализации одних и тех же задач;
- Очень тяжелая начальная загрузка (bootstrap) приложения;
- Отдельные компоненты сложно оптимизировать из-за длинных цепочек вызова;

Основные проблемы ZF1:

- Неудовлетворительная производительность;
- Дублирование некоторой функциональности используемых загрузчиков плагинов (plugin loader);
- Слишком много ответственности выдано View;
- Один тип исключения (Exception) на компоненту;
- Слишком много путей для реализации одних и тех же задач;
- Очень тяжелая начальная загрузка (bootstrap) приложения;
- Отдельные компоненты сложно оптимизировать из-за длинных цепочек вызова;
- Ресурсоемкие тесты и необходимость служебных классов «живущих» вместе с ними;

Знаковые нововведения в ZF2:

- Zend\Di — использует популярный паттерн в мире Java Dependency Injection (внедрение зависимостей);

Знаковые нововведения в ZF2:

- Zend\Di — использует популярный паттерн в мире Java Dependency Injection (внедрение зависимостей);
- Zend\EventManager — построен на замыканиях (лямбда-функиях);

Знаковые нововведения в ZF2:

- Zend\Di — использует популярный паттерн в мире Java Dependency Injection (внедрение зависимостей);
- Zend\EventManager — построен на замыканиях (лямбда-функиях);
- Иной взгляд на MVC;

Знаковые нововведения в ZF2:

- Zend\Di — использует популярный паттерн в мире Java Dependency Injection (внедрение зависимостей);
- Zend\EventManager — построен на замыканиях (лямбда-функиях);
- Иной взгляд на MVC;
- Центральное хранилище модулей (http://modules.zendframework.com/) для повторного использования кода, по примеру bundles для symfony;

Но следует понимать, что PHP !== JAVA

JAVA имеет сильные стороны и большой опыт реализации успешных интерфейсов.

Как это применимо для PHP?

[Практика]

Zend\EventManager

Менеджер событий

```
use Zend\EventManager\StaticEventManager,
    Zend\Log\Factory as LogFactory;

$log          = LogFactory($someConfig);
$events       = StaticEventManager::getInstance();
$events->attach('Foo', 'bar', function ($e) use ($log) {
    $event    = $e->getName();
    $target   = get_class($e->getTarget());
    $params = json_encode($e->getParams());

    $log->info(sprintf(
        '%s called on %s, using params %s',
        $event,
        $target,
        $params
    ));
});
```

```
// Later, instantiate Foo:
$foo = new Foo();

// And we can still trigger the above event:
$foo->bar('baz', 'bat');

// results in log message:
// bar called on Foo, using params
// {"baz" : "baz", "bat" : "bat"}"
```

Zend\Di
Паттерн Dependency Injection
Внедрение зависимости

Цель

— выделить доступ к определенному сервису / объекту в отдельный слой, дабы стать менее зависимым от реализации самого сервиса / объекта.

```
class SomeClass
{
}

$di                 = new Zend\Di\Di();
$someClass   = $di->get('SomeClass');
```

- Делегируем создание объекта контейнеру;

```
class SomeClass
{
        protected $_injectionClass;

         public function __construct(InjectionClass $injectionClass) {
                $this->_injectionClass = $injectionClass;
        }
}

$di           = new Zend\Di\Di();
$someClass    = $di->get('SomeClass');
```

- В результате, в переменной $someClass будет новый инстанс объекта с инициализированным свойством injectionClass.

Откуда Di контейнер узнает
как необходимо инициализировать
тот или иной объект?

ZfcUser/config/module.config.php

```
'di' => array(
    'instance' => array(
        'alias' => array(
            'zfcuser'                       => 'ZfcUser\Controller\UserController',
            'zfcuser_user_service'          => 'ZfcUser\Service\User',
            'zfcuser_auth_service'          => 'Zend\Authentication\AuthenticationService',
            'zfcuser_uemail_validator'      => 'ZfcUser\Validator\NoRecordExists',
            'zfcuser_uusername_validator'   => 'ZfcUser\Validator\NoRecordExists',
            'zfcuser_captcha_element'       => 'Zend\Form\Element\Captcha',

            // Default Zend\Db
            'zfcuser_zend_db_adapter' => 'Zend\Db\Adapter\Adapter',
            'zfcuser_user_mapper'     => 'ZfcUser\Model\UserMapper',
            'zfcuser_usermeta_mapper' => 'ZfcUser\Model\UserMetaMapper',
            'zfcuser_user_tg'         => 'Zend\Db\TableGateway\TableGateway',
            'zfcuser_usermeta_tg'     => 'Zend\Db\TableGateway\TableGateway',
        ),

        /* …. */
    )
);
```

- **модуль обеспечивает очень простую и настраиваемую систему аутентификации и регистрации пользователя**

Конфигурационный файл только объектов маленького модуля занимает 159 строк.

Любая неточность == частично инициализированный объект

Для реализации Di паттерна ZF2 использует рефлексию (reflection)

Проблемы:

- Сложность отладки программы;

Проблемы:

- Сложность отладки программы;
- Потеря производительности;

Проблемы:

- Сложность отладки программы;
- Потеря производительности;
- Приложение трудно читать из-за неочевидности какой объект отдает Di контейнер в данный момент;

Di контейнер внутри ZF2

- Используется в основе нового взгляда на парадигму MVC;
- “Тяжелый Bootstrap” ZF1 переместился в Zend/Mvc/Bootstrap.php

Zend/Mvc/Bootstrap.php

```
$di = new Di;
$di->instanceManager()->addTypePreference('Zend\Di\Locator', $di);

// Default configuration for common MVC classes
$diConfig = new DiConfiguration(array('definition' => array('class' => array(
    'Zend\Mvc\Router\RouteStack' => array(
        'instantiator' => array(
            'Zend\Mvc\Router\Http\TreeRouteStack',
            'factory'
        ),
    ),
    'Zend\Mvc\Router\Http\TreeRouteStack' => array(
        'instantiator' => array(
            'Zend\Mvc\Router\Http\TreeRouteStack',
            'factory'
        ),
    ),
    'Zend\Mvc\View\DefaultRenderingStrategy' => array(
        'setLayoutTemplate' => array(
            'layoutTemplate' => array(
                'required' => false,
                'type'     => false,
            ),
        ),
    )
```

Ваше приложение зависимо от Di

- С ростом проекта вызовы Di расползаются по всему приложению и в какой-то степени он становиться схож с Singleton'ом;
- Вы получаете узкое место и потенциально жесткую архитектуру, хоть и с возможностью подмены одного объекта другим.

Собрав все за и против – произведем нагрузочное тестирование

[Тестирование]

Команда ZF2 ставила для себя задачи улучшения согласованности и повышения производительности.

А также заявляла о некоторых решениях:

- Переход на Namespace (пространство имен);
- Рефакторинг исключений;
- Переход ZF исключительно на автозагрузку без использования require_once;
- Совершенствование и стандартизация системы плагинов;

Замеры производительности производились на железе следующей конфигурации:

Процессор:

CPU cores: 2
CPU family: 6
CPU MHz: 2999.748
Model Name: Intel Core 2 Duo CPU E8400
Cache size: 6144 KB

Оперативная память:
RAM: 2011 MB

Операционная система:
Debian: 6.0.5
Linux core: 2.6.32-5-amd64

ZF2 Приложение: ZendSkeletonApplication
(https://github.com/zendframework/ZendSkeletonApplication)

ZF1 Приложение: Default Welcome Application

Окружение:

Nginx: 1.2.0
Php: 5.4.0

Php modules: Core, ctype, curl, date, dom, ereg, exif, fileinfo, filter, ftp, gd, hash, iconv, json, libxml, mbstring, mcrypt, mysql, mysqli, mysqlnd, pcre, PDO, pdo_mysql, pdo_sqlite, Phar, posix, Reflection, session, SimpleXML, soap, sockets, SPL, sqlite3, standard, tokenizer, xml, xmlreader, xmlwriter, zip, zlib

Тестирование через Apache jMeter

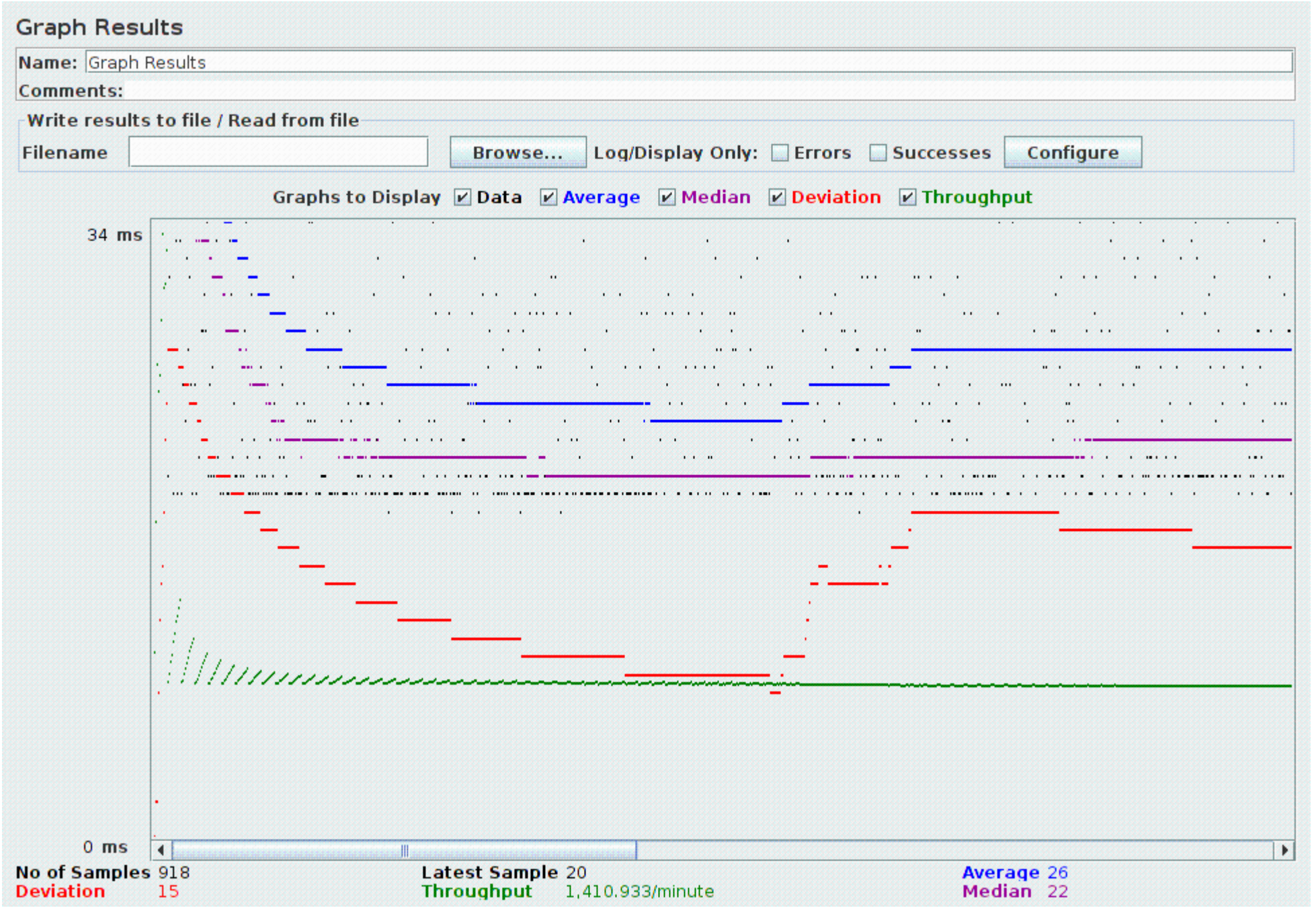
Задача

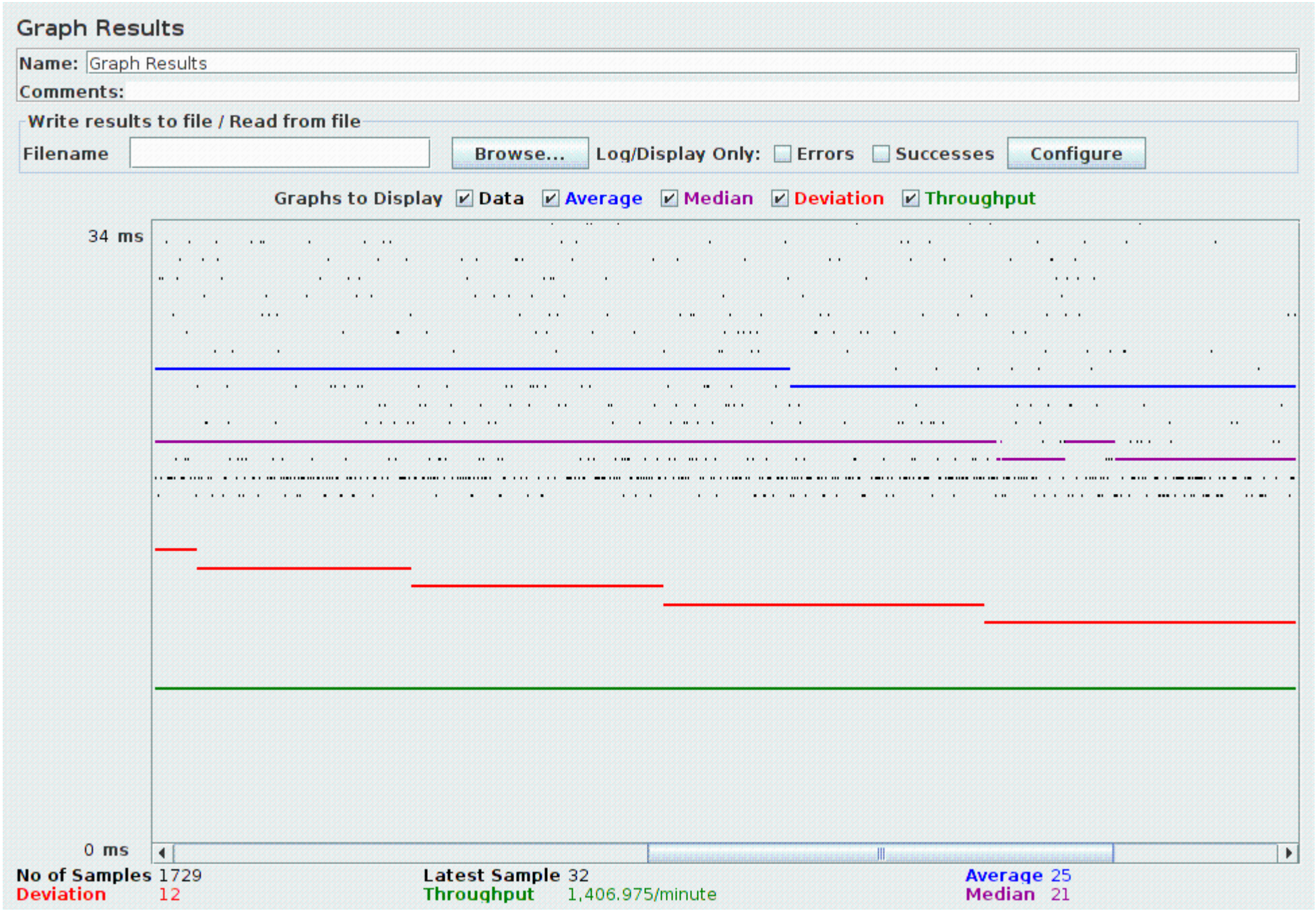
Протестировать отказоустойчивость фреймворка
при нагрузке 3000 пользователей в минуту на протяжении
10-ти минут в локальной сети.

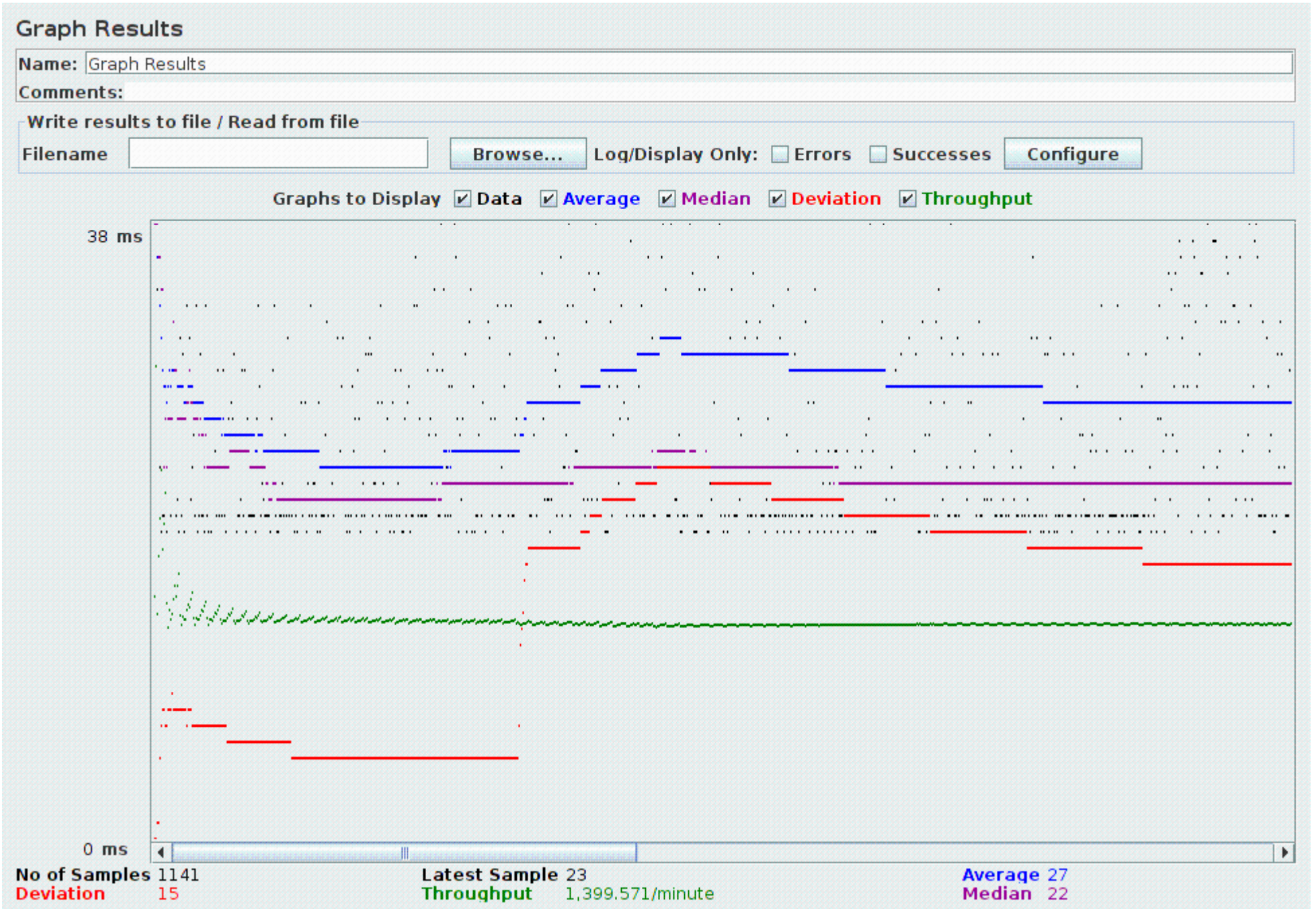
Решение

2 ноутбука через wifi каждые 400 миллисекунд осуществляют
10 одновременных подключений к серверу;

Тесты ZF1


Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
34 ms
0 ms
No of Samples 918
Deviation 15
Latest Sample 20
Throughput 1,410.933/minute
Average 26
Median 22


Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
34 ms
0 ms
No of Samples 1729
Deviation 12
Latest Sample 32
Throughput 1,406.975/minute
Average 25
Median 21


Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
38 ms
0 ms
No of Samples 1141
Deviation 15
Latest Sample 23
Throughput 1,399.571/minute
Average 27
Median 22

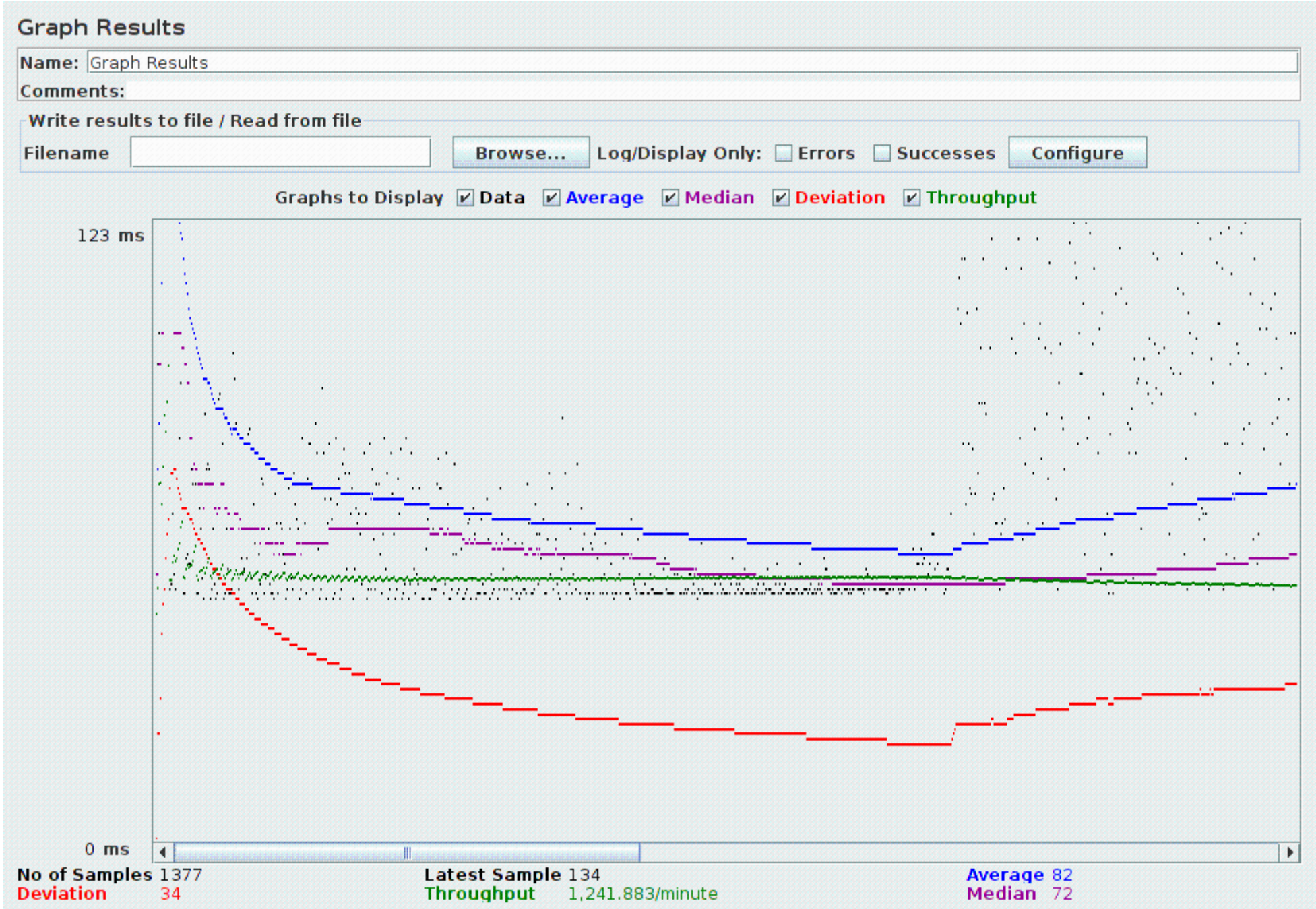
Результаты ZF1:

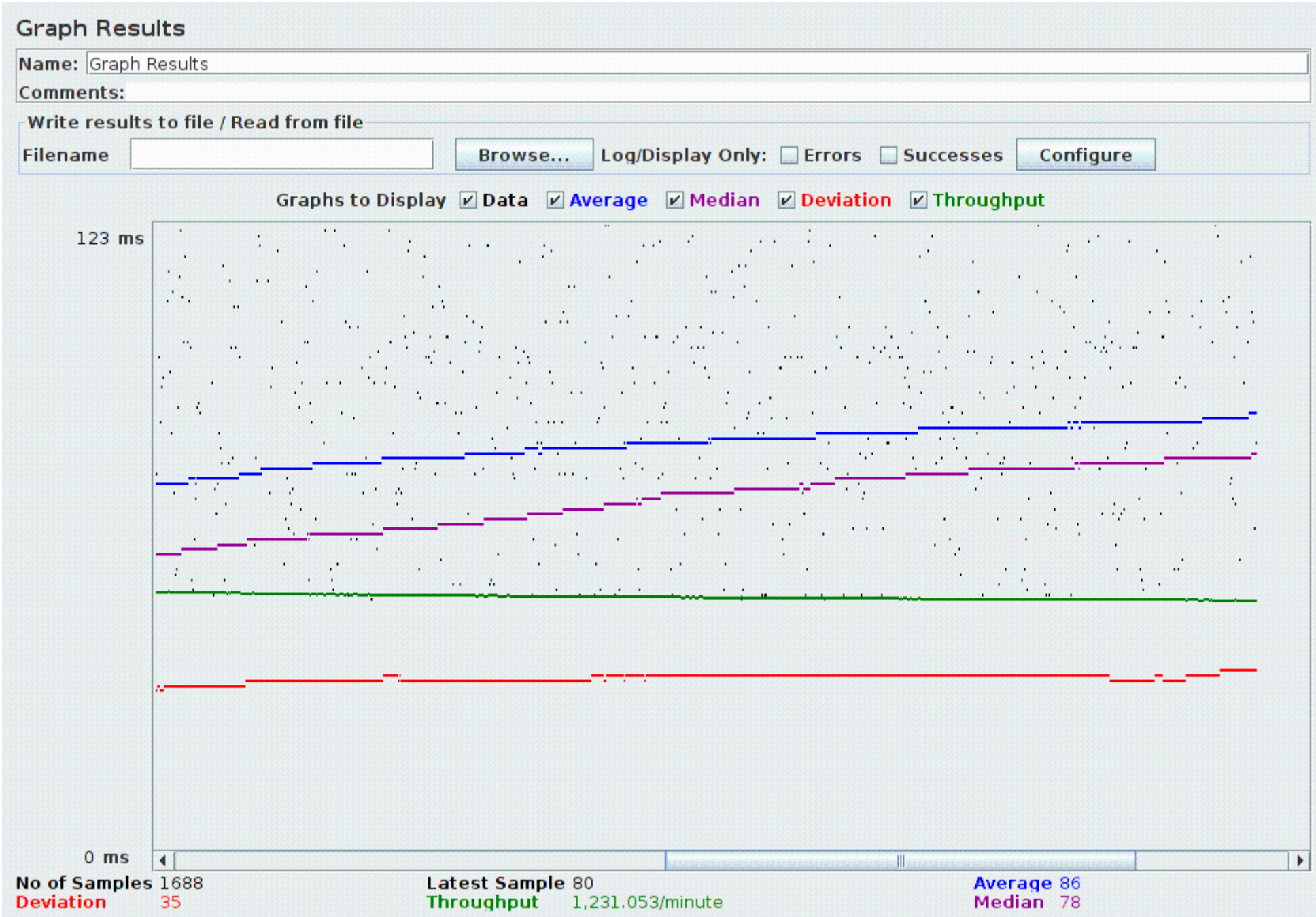
В среднем ZF1 выдавал страницу за	26ms;
Может обслуживать клиентов в минуту	~1200-1400;
Минимально клиент ждал	19ms;
Максимально клиент ждал	250ms;

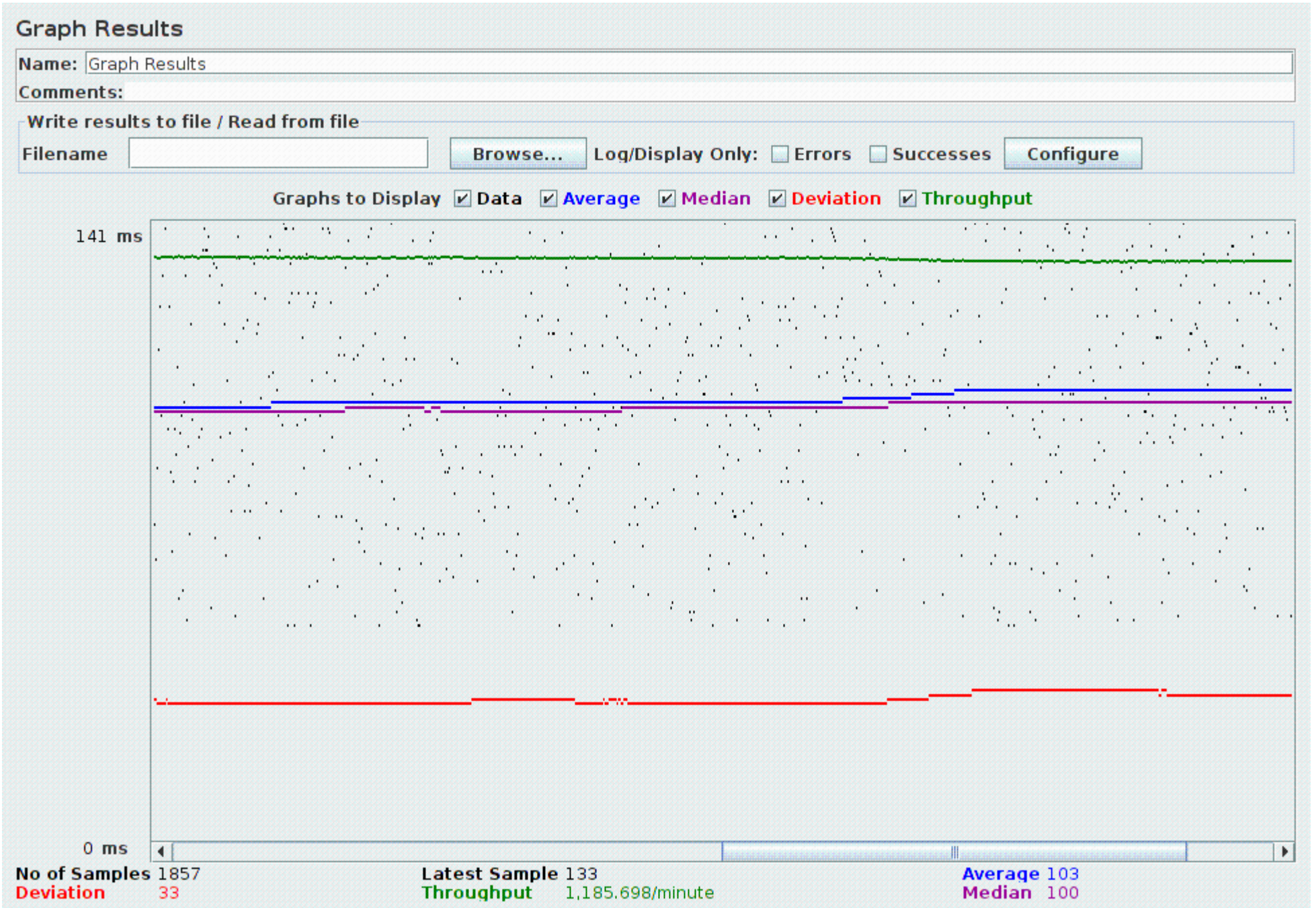
Пиковая нагрузка на сервер составила:

RAM	~10 MB;
CPU	~38%-50%;
Load Average	0.39;

Тесты ZF2


Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
123 ms
0 ms
No of Samples 1377
Deviation 34
Latest Sample 134
Throughput 1,241.883/minute
Average 82
Median 72


Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
123 ms
0 ms
No of Samples 1688
Deviation 35
Latest Sample 80
Throughput 1,231.053/minute
Average 86
Median 78


Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
141 ms
0 ms
No of Samples 1857
Deviation 33
Latest Sample 133
Throughput 1,185.698/minute
Average 103
Median 100

Результаты ZF2:

В среднем ZF2 выдавал страницу за	102ms;
Может обслуживать клиентов в минуту	~1000-1200;
Минимально клиент ждал	47ms;
Максимально клиент ждал	341ms;

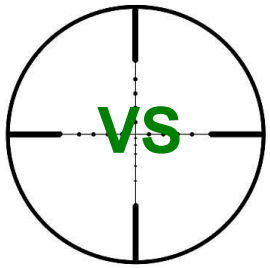
Пиковая нагрузка на сервер составила:

RAM	~20MB;
CPU	~92%-100%;
Load Average	3.98;

Под прицелом:

Zend Framework 1.11.11

2011-09-29 релиз



Zend Framework 2.0.0beta3

релиз 2012-03-02

Из приведенных графиков видно,
что ZF2 не только медленнее ZF1,
но и весьма прожорлив по ресурсам сервера.

Профилирование

Профайлер Xdebug

Архитектура ZF2 изнутри

Основная проблема производительности Zend\Di

Основная проблема производительности Zend\Di

- слишком много обращений за один сеанс скрипта;

Компромиссы, оптимизация?

В связи с текущей архитектурой, единственный выход, который мне видится — выпустить Di контейнер в виде Си расширения для PHP;

Компромиссы, оптимизация?

В связи с текущей архитектурой единственный выход, который мне видится — выпустить Di контейнер в виде Си расширения для PHP;

Я реализовал пробный вариант решения, в котором учтены еще далеко не все возможные инъекции, задуманные разработчиками, и благодаря которому быстродействие фреймворка уже увеличилось на 20%;

[Выводы]

- ZF2 существенно медленнее ZF1;

- ZF2 существенно медленнее ZF1;
- Dependency injection в PHP сложно организуемый паттерн.

- ZF2 существенно медленнее ZF1;
- Dependency injection в PHP сложно организуемый паттерн.
- PHP !== Java — эти границы следует очень четко понимать;

- ZF2 существенно медленнее ZF1;
- Dependency injection в PHP сложно организуемый паттерн.
- PHP !== Java — эти границы следует очень четко понимать;
- Тестирование решает многие проблемы выбора инструмента или методик;

- ZF2 существенно медленнее ZF1;
- Dependency injection в PHP сложно организуемый паттерн.
- PHP !== Java — эти границы следует очень четко понимать;
- Тестирование решает многие проблемы выбора инструмента или методик;
- Любое заявление среди разработчиков, и мое в частности, должно быть перепроверено вами лично и сделаны собственные выводы;

[Спасибо ;)]

Докладчик: Алексей Пархоменко
alexey@iprojects.com.ua