CONTENT

1 INTRODUCTION....................2
2 TESTS....................3
    2.1 Setup....................3
    2.2 Create/Destroy test....................5
    2.3 Screenshot capture test....................6
    2.4 Javascript OOP performance test....................8
    2.5 Web surfing test....................9
CONCLUSIONS....................11
APPENDIX A – Code of javascript OOP performance test....................12

1 INTRODUCTION

This research studied actual selenium WebDrivers, that available on linux platform. Sometimes it is necessary to write automated tests for website or application to communicate with website (what can't be done via API) etc, which can't be done by simply parsing html, since webpages contains complex websites with javascript logic, that has to be processed in the right way.

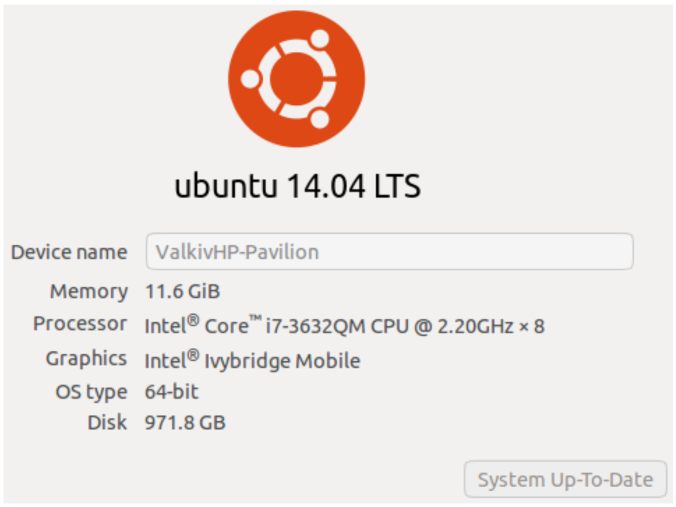
Here selenium can do the stuff by communicating with browser via special driver aka WebDriver. But which one is better, faster, uses less system resources and performing well?

2 TESTS

2.1 Setup

Java platform was used to test selenium WebDrivers.

Picture 2.1.1 shows system on which tests were run. All updates were installed for 2015-09-05. Table 2.1.1 shows WebDrivers participating in tests.


ubuntu 14.04 LTS

Device name ValkivHP-Pavilion
Memory 11.6 GiB
Processor Intel® Core™ i7-3632QM CPU @ 2.20GHz × 8
Graphics Intel® Ivybridge Mobile
OS type 64-bit
Disk 971.8 GB

Picture 2.1.1 – System information

Table 2.1.1 – WebDrivers participating in tests

Browser name	**Browser version**	**Selenium lib version**	**Other**
Phantomjs.	1.9.8 (latest)	2.41.0 (latest for phantomjs WebDriver)	
Firefox	40.0.3+build1-0ubuntu0.14.04.1 (ubuntu repository).	2.47.1 (latest)	
Chromium	44.0.2403.89 Ubuntu 14.04 (ubuntu repository)	2.47.1 (latest)	ChromeDriver: 2.15 (ubuntu repository)
Chrome	44.0.2403.157 (latest)	2.47.1 (latest)	ChromeDriver: 2.18.343837 (latest)

Each WebDriver was run with default settings on virtual display (preset environment variable “**DISPLAY=:1.0**”), which was started by following command:

Xvfb :1 -screen 0 800x600x8

Memory and CPU usage are summed for all subprocesses, that related to WebDriver (if WebDriver instance creates 11 processes then CPU and memory usage were summed).

Memory usage in the tests is the sum of “Private_Dirty” fields from “**/proc/{pid}/smaps**” file, which is equal to “memory” column in **gnome-system-monitor** tool.

CPU usage in the tests is the sum from “**ps -p {pid} -o %cpu**” tool (100% means 1 core fully loaded).

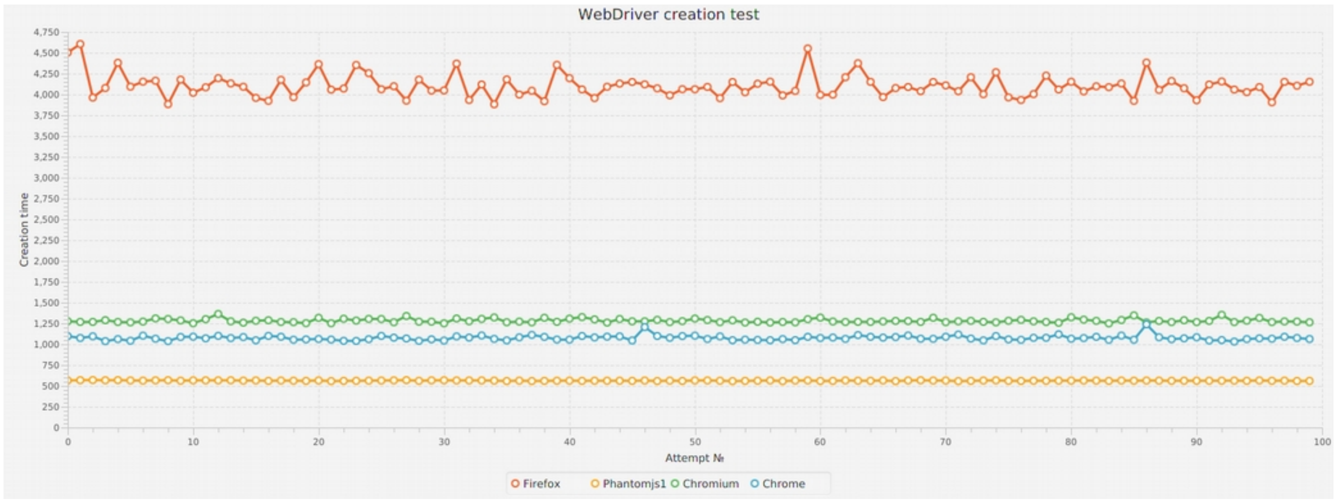
Swap file was disabled (there always were free ~4GB of RAM).

JVM was started with preallocated heap size of 2gb to not use garbage collector and heap allocation during tests.

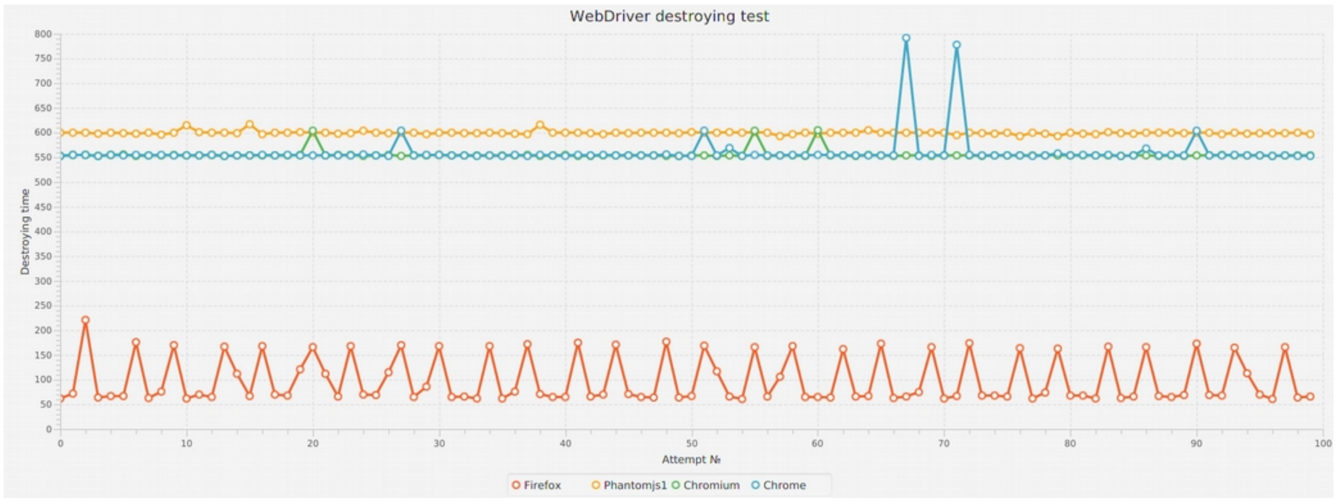
For each WebDriver window resolution were set to 1336x766.

Tests were run at least 3 times to ensure, that results are stable. WebDrivers in the tests were tested sequentially (only one WebDriver were run at the same time).

2.2 Create/Destroy test



Picture 2.2.1 – Time to create WebDriver instance



Picture 2.2.2 – Time to destroy WebDriver instance

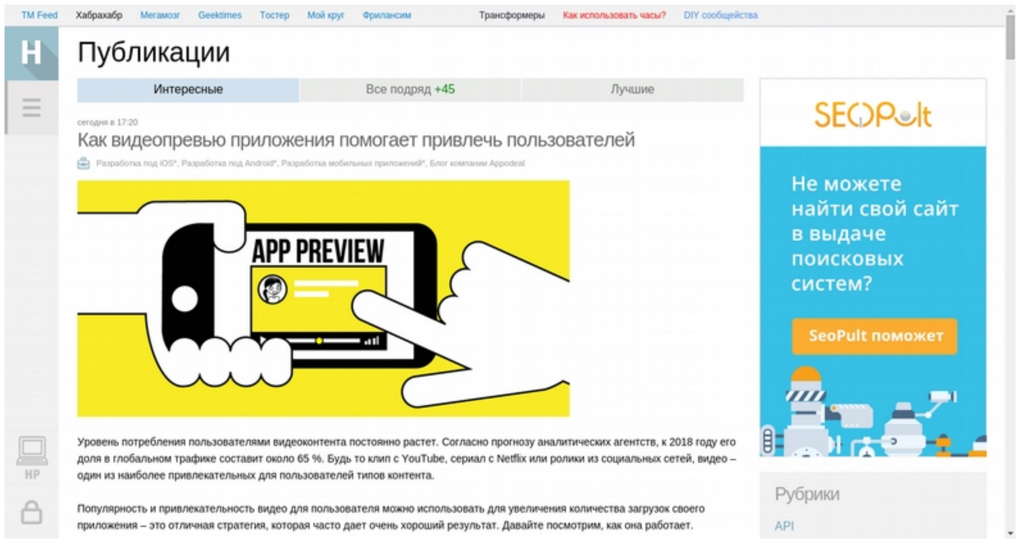
Table 2.2.1 – Summary info of create/destroy instance test

Browser	**Create instance**	**Destroy instance**	**Create+Destroy**
phantomjs	100%	545%	100%
Firefox	745%	100%	366%
Chromium	227%	500%	156%
Chrome	190%	500%	139%

Firefox definitely lost in instance create/destroy test. if application where WebDriver will be used has frequent, but short living jobs then best solution will be phantomjs or chrome/chromium WebDrivers.

2.3 Screenshot capture test

Phantomjs and firefox WebDrivers capture screenshots of whole page without scroll when Chrome/Chromium capture only area which is visible to user together with scroll, can be fixed with workaround by increasing window height to document height before capturing screenshot. See pictures 2.4.1-2.4.2.



Публикации

сегодня в 17:20

Как видеопревью приложения помогает привлечь пользователей

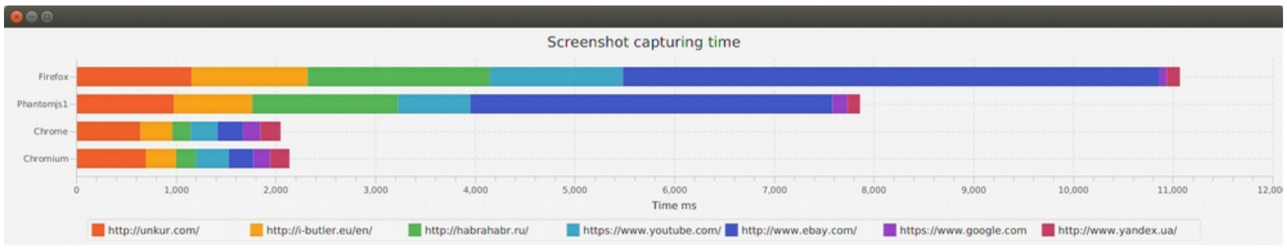
Разработка под iOS*, Разработка под Android*, Разработка мобильных приложений*, Блог компании Appodeal

Уровень потребления пользователями видеоконтента постоянно растет. Согласно прогнозу аналитических агентств, к 2018 году его доля в глобальном трафике составит около 65 %. Будь то клип с YouTube, сериал с Netflix или ролики из социальных сетей, видео – один из наиболее привлекательных для пользователей типов контента.

Популярность и привлекательность видео для пользователя можно использовать для увеличения количества загрузок своего приложения – это отличная стратегия, которая часто дает очень хороший результат. Давайте посмотрим, как она работает.

Picture 2.3.1 – Screenshot by chrome driver (visible to user area)

Picture 2.3.2 – Screenshot by phantomjs/firefox driver (whole page)

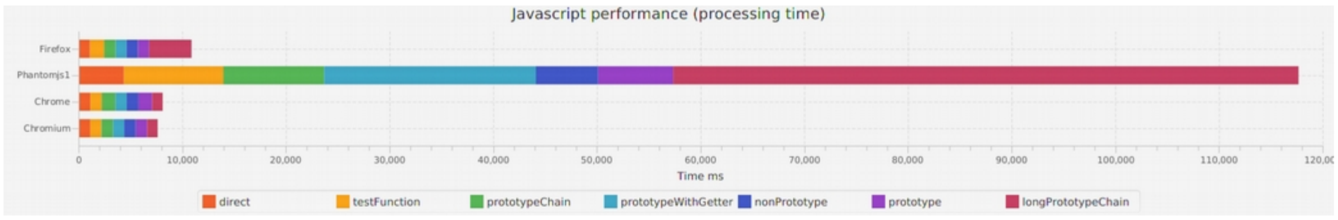


Picture 2.3.1 – Time to get screenshot (chrome/chromium driver takes less time, but they don't take screenshot of full page)

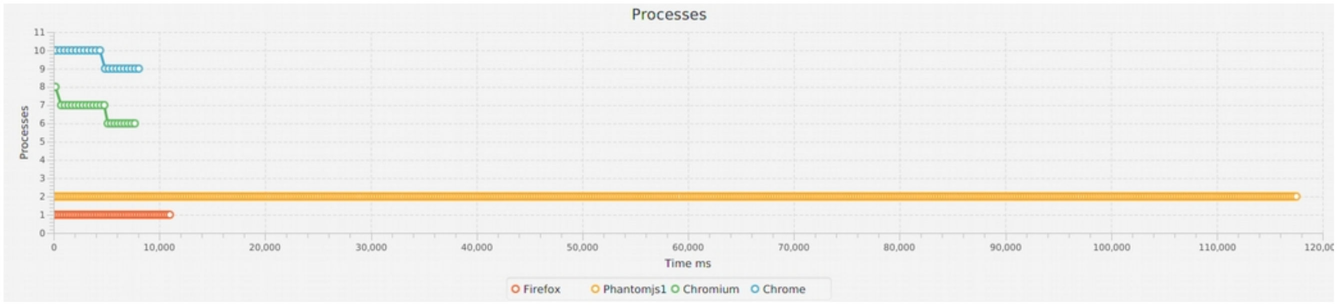
Depending on what application required (screenshot of whole page or screenshot of area, that visible to user) both solutions has place to live. But when it is possible to do a trick (increase window height by calculating document height) with chrome/chromium driver to get screenshot of full page, it is not possible to get screenshot of area, that is visible to user with phantomjs and firefox driver.

Size and color depth of virtual display doesn't matter, since screenshot resolution and color depth is bigger than **800x600x8**.

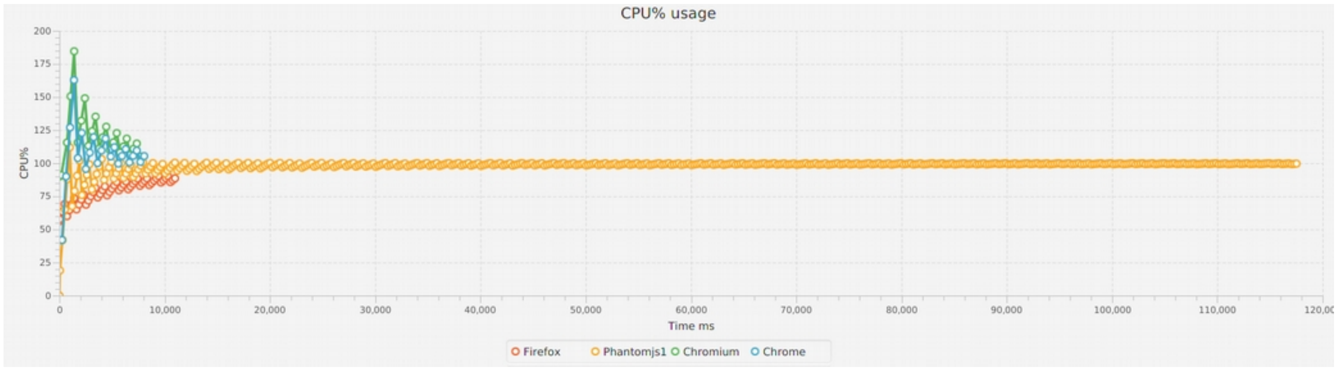
2.4 Javascript OOP performance test



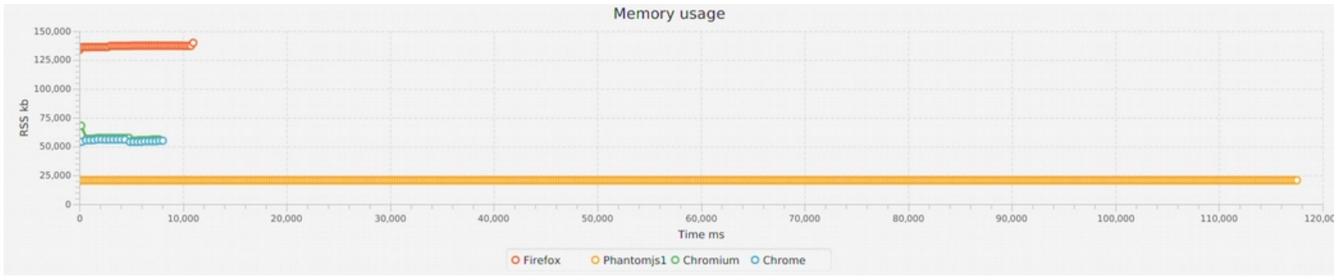
Picture 2.4.1 – Time to process test



Picture 2.4.2 – Total number of processes



Picture 2.4.3 – CPU usage



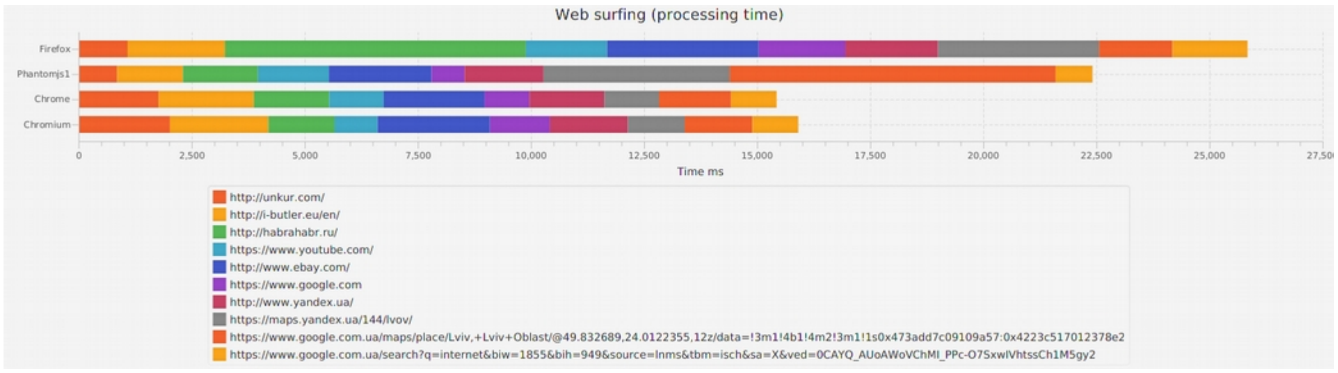
Picture 2.4.4 – Memory usage

Phantomjs definitely lost javascript performance test, it took less memory, but time to process javascript logic is more valuable.

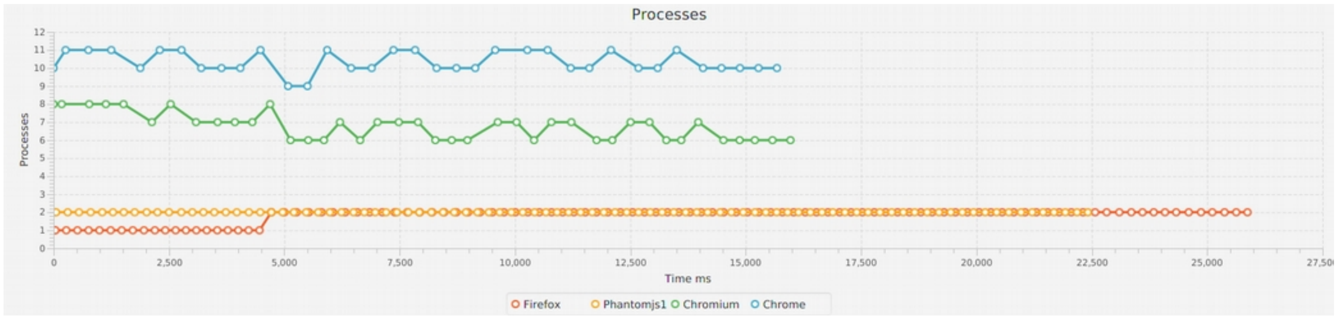
Firefox javascript performance is near to chrome, but lost in test with long OOP inheritance hierarchy to Chrome/Chromium. Also firefox took much more memory.

Looks like at the moment Chrome/Chromium has fastest javascript engine. Chrome/Chromium will be best choice to process fast websites with complex javascript logic.

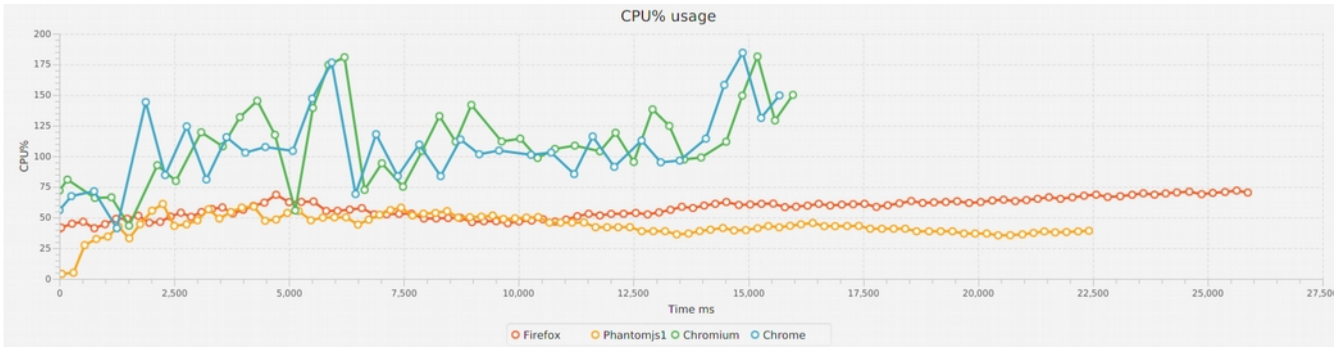
2.5 Web surfing test



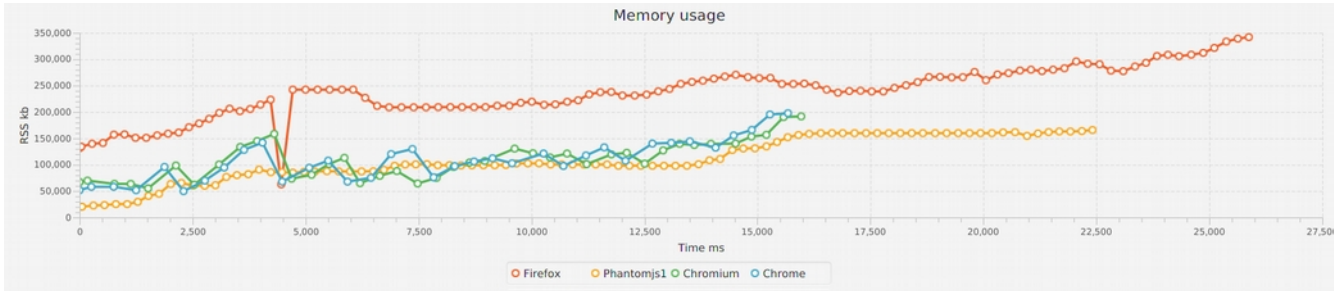
Picture 2.5.1 – Time to load web page



Picture 2.5.2 – Total number of processes



Picture 2.5.3 – CPU usage



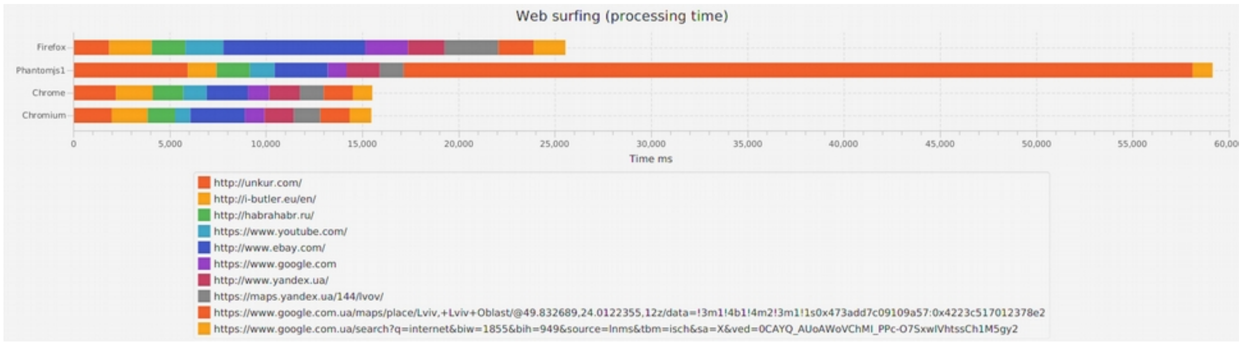
Picture 2.5.4 – Memory usage

Results of processing time can be completely different for each relaunch, for example one of web drivers could load webpage during 30 sec when with another try in 2 sec. They can differs because of websites on high load during requests, network etc. But in most cases CPU and memory usage near to above and Chrome/Chromium load webpage faster (maybe due to faster javascript engine).

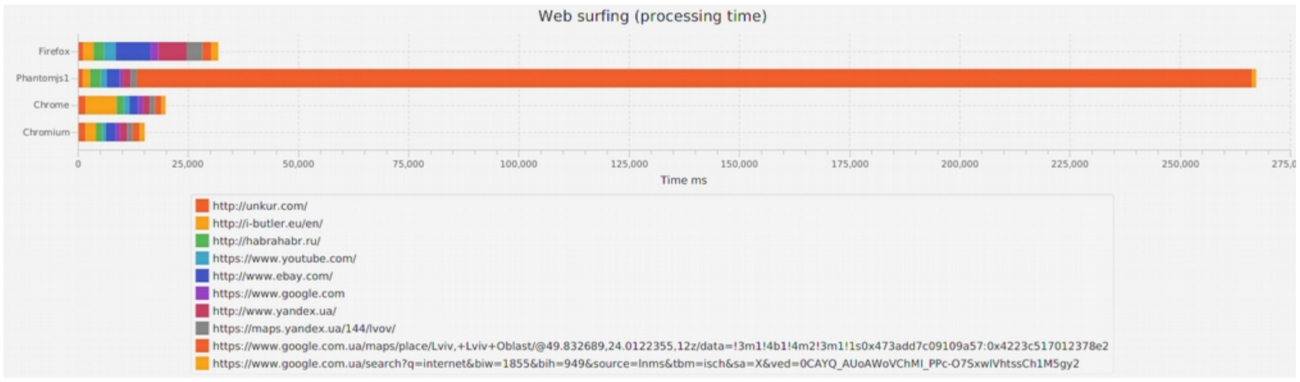
Firefox uses more memory than phantomjs and Chrome/Chromium WebDrivers, while CPU usage the same as for phantomjs.

Memory usage in Chrome/Chromium during loading pages the same as for phantomjs WebDriver, but Chrome/Chromium driver use more CPU resources which can be omitted since they load pages faster.

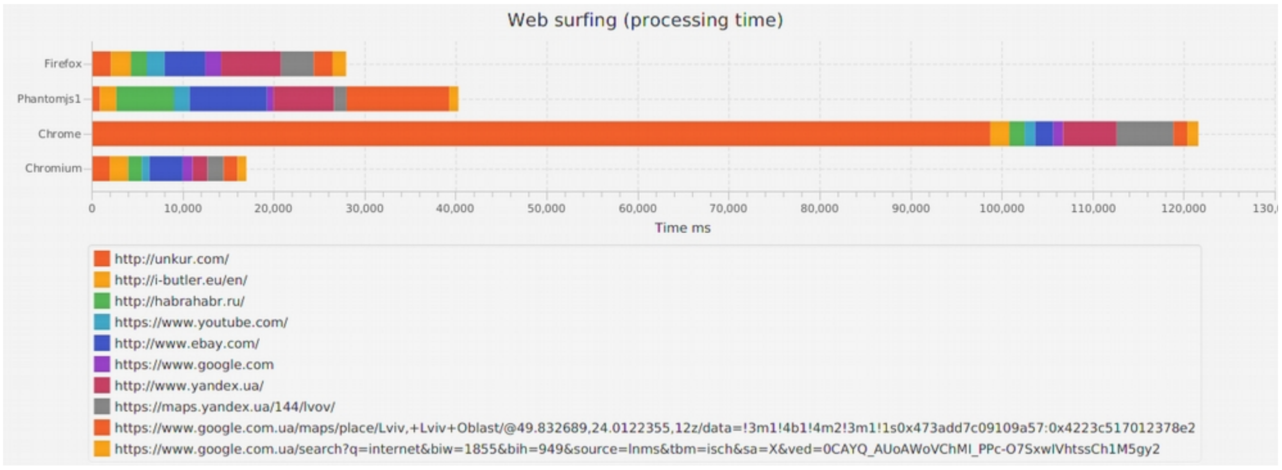
See few more relaunches on 2.5.5 - 2.5.7 pictures.



Picture 2.5.5 – Time to load web page. Attempt №2



Picture 2.5.6 – Time to load web page. Attempt №3



Picture 2.5.7 – Time to load web page. Attempt №4

CONCLUSIONS

Phantomjs is GUI-less and can be good solution in order to quickly start to work. It doesn't required special environment on server and will work out of the box. It will use less memory since it GUI-less and doesn't require overhead for GUI. Issue with phantomjs, that on some websites it could crash, stuck etc when Firefox and Chrome will work fine.

In case when faster javascript engine required or phantomjs crashes on specific websites, Chrome/Chromium will be nice choice since it use less memory, has better javascript engine, load pages faster than Firefox.

One more situation when Chrome/Chromium and Firefox have to be used is extensions/plugins, which can enhance web-content (adblock, highlighter, content enhancers etc) and of course is not available for phantomjs.

Firefox/Chrome will require you to setup virtual display on server, but it is easy to do.

One distinct feature of Chrome/Chromium is number of processes for one instance, which is near to ~10 processes per WebDriver instance, but this should not be an issue. Anyway some kind of cleanup mechanism should exist to destroy uncontrolled Phantomjs/Firefox/Chrome processes, which may not be destroyed successfully (for example deamon process, which destroys such processes by names if their lifetime is more than 10 minutes, which can be determined by standard linux utils).

APPENDIX A – Code of javascript OOP performance test

```
function testPerformance() {
    return {
        direct: getExecutionTime(testDirect),
        testFunction: getExecutionTime(testFunction),
        nonPrototype: getExecutionTime(testNonPrototype),
        prototypeChain: getExecutionTime(testPrototypeChain),
        prototype: getExecutionTime(testPrototype),
        longPrototypeChain: getExecutionTime(testLongPrototypeChain),
        prototypeWithGetter: getExecutionTime(testPrototypeWithGetter)

    }
}
var DELTA = 10;
var ITERATIONS = 1000;

function getExecutionTime(f) {
    var start = new Date().getTime();
    f();
    var end = new Date().getTime();
    var diff = end - start;
    return diff;
}

function testPrototypeChain() {

    function Parent() {
        this.delta = DELTA;
    };

    function ChildA() {};
    ChildA.prototype = new Parent();

    function ChildB() {}
    ChildB.prototype = new ChildA();

    function ChildC() {}
    ChildC.prototype = new ChildB();

    function ChildD() {};
    ChildD.prototype = new ChildC();

    function ChildE() {};
    ChildE.prototype = new ChildD();

    function nestedFn() {
        var child = new ChildE();
        var counter = 0;
        for (var i = 0; i < ITERATIONS; i++) {
            for (var j = 0; j < ITERATIONS; j++) {
                for (var k = 0; k < ITERATIONS; k++) {
                    counter += child.delta;
                }
            }
        }
        console.log('Final result: ' + counter);
    }
    nestedFn();
}


function testPrototype() {
    function Parent() {};
    Parent.prototype.delta = DELTA;

    function ChildA() {};
    ChildA.prototype = new Parent();

    function nestedFn() {
        var child = new Parent();
        var counter = 0;
        for (var i = 0; i < ITERATIONS; i++) {
            for (var j = 0; j < ITERATIONS; j++) {
                for (var k = 0; k < ITERATIONS; k++) {
                    counter += child.delta;
                }
            }
        }
        console.log('Final result: ' + counter);
    }

    nestedFn();
}

function testPrototypeWithGetter() {
    function Parent() {};
    Parent.prototype.delta = DELTA;
    Parent.prototype.getDelta = function() {
        return this.delta;
    };

    function ChildA() {};
    ChildA.prototype = new Parent();

    function ChildB() {}
    ChildB.prototype = new ChildA();

    function ChildC() {}
    ChildC.prototype = new ChildB();

    function ChildD() {};
    ChildD.prototype = new ChildC();

    function ChildE() {};
    ChildE.prototype = new ChildD();

    function nestedFn() {
        var child = new ChildE();
        var counter = 0;
        for (var i = 0; i < ITERATIONS; i++) {
            for (var j = 0; j < ITERATIONS; j++) {
                for (var k = 0; k < ITERATIONS; k++) {
                    counter += child.getDelta();
                }
            }
        }
        console.log('Final result: ' + counter);
    }

    nestedFn();
}

function testNonPrototype() {
    function Parent() {
        this.delta = DELTA;
    };

    function nestedFn() {
        var child = new Parent();
        var counter = 0;
        for (var i = 0; i < ITERATIONS; i++) {
            for (var j = 0; j < ITERATIONS; j++) {
                for (var k = 0; k < ITERATIONS; k++) {
                    counter += child.delta;
                }
            }
        }
        console.log('Final result: ' + counter);
    }

    nestedFn();
}
```

```
function testFunction() {
    function getDelta(object) {
        return object.delta;
    }

    function nestedFn() {
        var child = {
            delta: DELTA
        };
        var counter = 0;
        for (var i = 0; i < ITERATIONS; i++) {
            for (var j = 0; j < ITERATIONS; j++) {
                for (var k = 0; k < ITERATIONS; k++) {
                    counter += getDelta(child);
                }
            }
        }
        console.log('Final result: ' + counter);
    }
    nestedFn();
}

function testLongPrototypeChain() {
    function Parent() {};
    Parent.prototype.sub = new SubChildE();

    function ChildA() {};
    ChildA.prototype = new Parent();

    function ChildB() {}
    ChildB.prototype = new ChildA();

    function ChildC() {}
    ChildC.prototype = new ChildB();

    function ChildD() {};
    ChildD.prototype = new ChildC();

    function ChildE() {};
    ChildE.prototype = new ChildD();

    function SubParent() {
        this.delta = DELTA;
    };

    function SubChildA() {};
    SubChildA.prototype = new SubParent();

    function SubChildB() {}
    SubChildB.prototype = new SubChildA();

    function SubChildC() {}
    SubChildC.prototype = new SubChildB();

    function SubChildD() {};
    SubChildD.prototype = new SubChildC();

    function SubChildE() {};
    SubChildE.prototype = new SubChildD();

    function nestedFn() {
        var child = new ChildE();
        var counter = 0;
        for (var i = 0; i < ITERATIONS; i++) {
            for (var j = 0; j < ITERATIONS; j++) {
                for (var k = 0; k < ITERATIONS; k++) {
                    counter += child.sub.delta;
                }
            }
        }
        console.log('Final result: ' + counter);
    }
    nestedFn();
}
```

```
function testPrototypeChain() {
    function Parent() {
        this.delta = DELTA;
    };

    function ChildA() {};
    ChildA.prototype = new Parent();

    function ChildB() {}
    ChildB.prototype = new ChildA();

    function ChildC() {}
    ChildC.prototype = new ChildB();

    function ChildD() {};
    ChildD.prototype = new ChildC();

    function ChildE() {};
    ChildE.prototype = new ChildD();

    function nestedFn() {
        var child = new ChildE();
        var counter = 0;
        for (var i = 0; i < ITERATIONS; i++) {
            for (var j = 0; j < ITERATIONS; j++) {
                for (var k = 0; k < ITERATIONS; k++) {
                    counter += child.delta;
                }
            }
        }
        console.log('Final result: ' + counter);
    }

    nestedFn();
}

function testDirect() {
    function nestedFn() {
        var delta = DELTA;
        var counter = 0;
        for (var i = 0; i < ITERATIONS; i++) {
            for (var j = 0; j < ITERATIONS; j++) {
                for (var k = 0; k < ITERATIONS; k++) {
                    counter += delta;
                }
            }
        }
        console.log('Final result: ' + counter);
    }
    nestedFn();
}
```