TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

IDK40LT

Artjom Tsarajev 050520IABB

MARKDOWN-BASED FLAT-FILE CMS PERFORMANCE COMPARISON ON LOW-POWER COMPUTERS

Bachelor’s thesis

Supervisor: Jekaterina Tšukrejeva
Master’s degree
Assistant

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

IDK40LT

Artjom Tsarajev 050520IABB

MARKDOWN'IL PÕHINEVATE SISUHALDUSSÜSTEEMIDE JÕUDLUSE VÕRDLUS MADALA VOOLUTARBIMISEGA ARVUTITEL

bakalaureusetöö

Juhendaja: Jekaterina Tšukrejeva
Magistrikraad
Assistent

Tallinn 2016

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Artjom Tsarajev

23.05.2016

Abstract

The purpose of this work is to analyse and compare the performance of flat-file content management systems on low-power computers with various hardware specifications. The objective is to determine whether such hardware is a viable option for hosting websites that generate web content from specifically formatted text files and find out the bottlenecks of such solutions as well as an approximate workload that these systems can handle, so that unnecessarily complex server setups can be replaced with lightweight alternatives if workload requirements are not demanding.

This work is aimed at finding out if it is reasonable to dynamically generate web pages or statically generated systems should be preferred, whether the web server software is better optimized for certain hardware architectures, as well as how the CPU speed, the number of CPU cores, RAM size or the network interface speed influence the performance of different CMS-s.

In the course of this work it was determined that both Raspberry Pi systems in question were capable of running an instance of *Apache* web server with a flat-file CMS on top of it. They have managed to provide response times of up to 3 seconds within the local area network for a given set of files. They could handle the load of up to 64 simultaneous requests. This proves these setups can be utilized for small working environments where complex functionality is not required, e.g. a corporate intranet web server, a personal blog or similar.

This thesis is written in English and is 73 pages long, including 7 chapters, 24 figures and 4 tables.

Annotatsioon

Markdown'il põhinevate sisuhaldussüsteemide jõudluse võrdlus madala voolutarbimisega arvutitel

Selle töö eesmärgiks on analüüsida ja võrrelda populaarsete tekstifailipõhiste sisuhaldussüsteemide jõudlust madala voolutarbimisega arvutitel. Lõpptulemuseks on arusaam, kas selliseid arvuteid saab kasutada tekstifailidest genereeritavate veebilehtede majutamiseks, mis on selliste lahenduste pudelikaelad ning mis koormusele nad vastu peavad.

Selle töö tulemuseks on vastused sellistele küsimustele, nagu:

- Kas on mõttekas genereerida veebilehte dünaamiliselt või tuleks kasutada staatilisi lehte?
- Kas veebiserverite tarkvara on paremini optimeeritud teatud riistvaraarhitektuuride jaoks?
- Kuidas mõjutab jõudlust protsessori kiirus?
- Kuidas mõjutab jõudlust operatiivmälu suurus?
- Kuidas mõjutab jõudlust võrguadapter?
- Kuidas mõjutab kiirust protsessori tuumade arv?

Selle töö käigus oli avastatud, et mõlemad testis olevad Raspberry Pi süsteemid olid võimelised jooksutama *Apache* veebiserveri koos tekstifailipõhise sisuhaldussüsteemiga. Nad olid suutelised serveerida lokaalse võrgu kaudu teatud tekstifailidest dünaamiliselt genereeritud veebilehte 3 sekundi piires. Nad said hakkama kuni 64 samaaegsete ühendustega. See tõestab seda, et selliseid süsteeme saab kasutada väikestes töökeskkondades, kus nõudmised funktsionaalsusele ei ole väga kõrged, näiteks firmasisene uudisteportaal, personaalne blogi vms.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 73 leheküljel, 7 peatükki, 24 joonist, 4 tabelit.

List of abbreviations and terms

CMS	Content management system
DBMS	Database management system
CPU	Central processing unit
RAM	Random-access memory

Table of contents

List of figures

List of tables

1 Introduction

According to a research [1], over 25% of all websites available on the Internet are powered by *Wordpress*, *Joomla* or other similar platforms. These traditional CMS-s usually make use of a DBMS for backend storage, all of which require additional management for security and performance reasons. In order to get the new features as they are added in new versions of these CMS-s, the system needs to be updated as well. This additional management includes, but is not limited to, performance monitoring, checking for data corruption and backing up the data itself.

Flat-file CMS-s, on the other hand, generate website content from regular text files, as long as they conform to the required syntax a particular CMS supports. *Markdown* [2] language allows the files to be written in a manner that is easily readable while still supporting basic text formatting. *Markdown* implementations [3] are also available for numerous commonly used programming languages.

Low-power systems are generally considered a niche product when it comes to website hosting. Hardware products like the Raspberry Pi [4] have made ARM-based devices available to the wider audience due to their low cost and low entry barrier. They are available in most of the countries around the world and there are a number of various operating systems that can run on them. These ARM-based products also require as little as 180mA to operate [5]. This makes them a viable computing platform for people who care about the natural environment.

1.1 Background and problem

The problem the author will be trying to solve during the course of this work is optimizing the infrastructure and operational costs of the existing system which consists of several traditional database-backed CMS-s and x86 server architecture. The goal is to find out whether it is worth investing in an ARM based solution to minimize operational costs related to power consumption as well as making sure system resources are being used adequately. It is absolutely clear that running a simple *Apache+PHP+SQL* instance is not needed for a simple page that mostly serves static content. Neither is it reasonable to set up a system that will stay idle most of the time due to low number of page requests.

This work might be of interest to enterprises dealing with hosting websites for small to medium-sized companies as well as independent companies looking to keep their employees up-to-date with the latest internal information or just casual bloggers to privately host their websites at a minimum cost.

1.2 Challenge

This work will answer the following questions for potential users:

- Is it worth switching from an existing CMS to a flat-file CMS? A flat-file CMS does not require a backend database, which minimises the time for software maintenance.
- Is ARM a viable architecture for hosting flat-file CMS-s? Low-power systems can help save money by using less electricity. ARM hardware is relatively cheap, no expensive server hardware needs to be purchased.
- What performance can be expected from such systems?
- What performance bottlenecks can be expected as demands grow?

1.3 Methodology

In order to address the aforementioned questions, we must determine whether and to what extent:

- Statically generated web pages are faster than dynamically generated ones.
- RAM size influences system performance.
- CPU speed influences system performance.
- The network interface influences system performance.

- The number of CPU cores influences system performance.
- Certain software is better optimized for certain system architectures.

1.4 Overview

In chapter 2 we will get an overview of flat-file CMS-s and comment on the ones chosen for testing.

In chapter 3 we will discuss the benefits of ARM architecture and describe the hardware used for testing.

In chapter 4 the author will present the chosen methodologies for measuring system performance.

In chapter 5 the author will present the results of different system performance measurements as well as comment on observations made.

In chapter 6 the author will present graphs showing the performance of various CMS-s.

In chapter 7 the author will draw conclusions based on the results observed.

2 Flat-file content management systems

Content management systems can be basically divided into 2 major categories: the ones that store their content in the form of databases and those that rely on regular text files.

Managing a CMS with a database backend inevitably means having a procedure in place to perform service maintenance, updating, backing up as well as having to deal with constant security vulnerabilities, security assessments and other routine tasks. Newer versions of such content management systems require thorough investigation of possible compatibility issues as well as other potential problems.

Content management systems that rely on text files as their backend storage, on the other hand, require much less attention, being easily backed up using regular scripts, requiring merely to conform to a certain standard file structure to be correctly rendered for the end user. These characteristics make them a great choice for cases where the main goal is to deliver the content itself, rather than to make the experience as functionally and visually appealing as possible. Such systems are a perfect fit for personal or corporate blogs in particular. If one were to post an article or a quick update for system users, it is usually not critical to have all the functionality that regular database-backed CMS-s offer.

2.1 CMS candidates

New flat-file CMS-s can appear at any point in time, so the author has decided to settle on a list of flat-file CMS-s posted on GitHub [6].

To test the performance of flat-file CMS-s, the author has come up with certain criteria that the CMS in question must satisfy. These requirements are specific to this work and further research on the topic can include a wider variety of criteria. This work focuses on CMS-s that satisfy the following requirements:

1. Source code must be available for download to identify the potential security or performance issues. It is also essential if users decide to implement additional

functionality and make contributions to the project. The following CMS-s have been discarded: Pulse CMS, Statamic.

2. *Markdown* support is required in order to compare performance based on the same sample data. The sample set will include text files written in *Markdown* of different size. The following CMS-s have been discarded: Flat Press, Flot, Get Simple CMS, Mozilo, Nanote, PluXML, Pluck CMS, Razor CMS, Vodka, WonderCMS, Nibbleblog.

3. The system must be self-hosted in order to test the performance on a fixed list of hardware. The following CMS-s have been discarded: Dodger CMS.

4. The CMS must be an active project. In the context of this work, the author considers a CMS active, if the latest version was released after 02.03.2015. The following CMS-s have been discarded:

    - Dropplets (release 1.6.2.6 08.09.2013)

    - Feindura (release 2.0.7 30.09.2014)

    - Nibbleblog (release 3.7.1c 06.11.2013) (*Markdown* support version)

5. The project website must clearly list system requirements and software dependencies along with installation instructions. In case all documented requirements have been satisfied, the CMS should work without any additional configuration. The following CMS-s have been discarded:

    - Automad

    Example pages are not working, documentation [7] was followed.

    - Baun

    “index.php” tries to install dependencies via external application “composer”.

    - Metalsmith

The software comes without any bundled plugins or templating engines, all functionality must be implemented from scratch.

- Kirby

  Displays a blank page, no errors observed, documentation [8] was followed.

- Monstra

  “install.php” shows all requirements as satisfied, however, “/install.php?action=install” displays a blank page after entering the setup details, documentation [9] was followed.

- Sphido

  Displays a blank page, no errors observed, documentation [10] was followed.

6. The CMS works with PHP 5.4.3 (the one provided by Tiny Core Linux apache2-mod-php5.tcz package). The following CMS-s have been discarded:

   - Fansoro (requires PHP 5.5)
   - Grav (requires PHP 5.5.9)
   - Parvula (requires PHP 5.5)

7. The underlying technology has to be working on Tiny Core Linux (both ARM and x86 ports). The following CMS-s have been discarded:

   - Herbie

     Works on x86 only, running on Raspberry Pi shows errors in German, no English documentation available.

   - Hugo

     Starting the server results in error “panic: runtime error: invalid memory address or nil pointer dereference”.

- Jekyll

Ruby extension has to be manually recompiled, according to Tiny Core Linux forum [11].

- Middleman

Ruby extension has to be manually recompiled, according to Tiny Core Linux forum [11]

- Nesta

Ruby extension has to be manually recompiled, according to Tiny Core Linux forum [11]

- Hexo

While it is possible to run this CMS on a Raspberry Pi, the installation procedure involves installing the “node.tcz” in “copy to fs” mode, which makes it unpractical, as it only runs until the system is restarted. Once restarted, it is not possible to run “node.tcz” in the same manner as usual, so the Tiny Core Linux distribution image has to be copied to the SD Card once again and manual repartitioning has to occur. The whole procedure is rather time-consuming, so the CMS had to be disqualified. It is important to note, that this was not an issue on x86 platform when booting into a LiveCD environment.

- Urubu

The official installation instructions require “pip” package manager to install. No manual installation instructions are provided. Tiny Core Linux ARM port does not have this package available as of 10.05.2016.

- Wintersmith

While it is possible to run this CMS on a Raspberry Pi, the installation procedure involves installing the “node.tcz” in “copy to fs” mode, which makes it unpractical, as it only runs until the system is restarted. Once restarted, it is not possible to run “node.tcz” in the same manner as usual, so the Tiny Core Linux distribution image has to be copied to the SD Card once again and manual repartitioning has to occur. The whole procedure is rather time-consuming, so the CMS had to be disqualified. It is important to note, that this was not an issue on x86 platform when booting into a LiveCD environment.

2.2 Final contenders

The following flat-file CMS-s have been chosen for the final performance analysis. Table 1 compares various properties of the chosen CMS-s.

Table 1. Properties of the chosen CMS-s.

Name	Release number, date	Notes
Bludit [12]	1.1.2 – 27.02.2016	Administrator account needs to be setup initially.
HTMLy [13]	2.7.4 – 24.01.2016	Administrator account needs to be setup initially.
Mecha [14]	1.2.5 – 22.04.2016	Administrator account needs to be setup initially.
Phile [15]	1.7.1 – 26.04.2016	No control panel available, files need to be uploaded manually.
Pico [16]	1.0.2 – 16.03.2016	No control panel available, files need to be uploaded manually.
PuppyCMS [17]	2.0 – 17.03.2016	No control panel available, files need to be uploaded manually.
Singularity [18]	no release number, using 25.12.2015 GitHub commit bbd9bb0	Singularity is notable for its size (42 lines of code), as well as being contained in a single PHP file. No control panel available, files need to be uploaded

		manually.
Yellow [19]	0.6.3 – 23.02.2016	Yellow is supposed to have an admin panel, however, the author was unable to access it, as the crypto library that Yellow uses was not available on the system. Nevertheless, sample files could be added manually by uploading.

3 ARM architecture

The ARM processor architecture [20] has been around for nearly 30 years, however, it only saw mass adoption with the introduction of smartphones and tablets. Devices built using this architecture are usually preferred when low power usage is needed. ARM is a reduced instruction set family of processors which deliver a moderate power-speed ratio. Certain enterprise-grade server solutions have been introduced to the public, e.g. HP ProLiant Moonshot [21]. However, the x86 architecture still dominates the server market by a large margin [22].

One of the goals of this work is to minimise the cost of operating a traditional server room. The current setup sports a complex air-conditioning system and occupies a relatively large room. Traditional server hardware is costly, and in the case of hardware failure, if failover is not properly implemented, a lot of systems can be rendered unusable until the whole server or the faulty part is replaced. This problem becomes bigger as more physical servers are used as virtual machine hosts. ARM hardware could allow companies to have smaller server rooms with less money spent on cooling, have better redundancy where cheap systems can be upgraded more often, or even have mobile computing units consisting of multiple ARM servers. A similar concept was introduced using traditional x86 hardware [23].

3.1 Hardware choice

While there are many devices built around ARM processors, ranging from smartphones and tablets to enterprise-grade solutions, Raspberry Pi devices are by far the most affordable and well-maintained systems available.

For these reasons, a known list of systems was chosen, including: Raspberry Pi 1 Model B (ARMv6), Raspberry Pi 2 Model B (ARMv7) and an x86 PC with similar hardware specifications. While Raspberry Pi 3 was introduced recently, the author decided it was impractical to test out its performance due to the fact that it lacks full software support at this point in time.

Related hardware specifications [24] for available systems are listed Table 2.

Table 2. Hardware specification of the chosen computer systems.

Device	CPU	RAM	Network interface
Raspberry Pi 1 Model B	Broadcom BCM2835 700 MHz Cores: 1	512 MB	10/100 Mbit/s (8P8C)
Raspberry Pi 2 Model B	Broadcom BCM2836 900 MHz Cores: 4	1 GB	10/100 Mbit/s (8P8C)
X86	Pentium III 933MHz Cores: 1	512 MB	10/100 Mbit/s (Intel PRO/100)

4 Test methodology and lab setup

In order to get the most accurate results, certain decisions had to be made. These included determining sample datasets that would be the same throughout the testing phase, the base operating system that would provide the least amount of overhead while still delivering the proper toolset, reasonably performing network equipment that would not influence the speed tests, as well choosing functional monitoring tools.

4.1 Sample dataset

In order to test the performance of various CMS-s, a number of differently sized text files was selected. This is important, because it will allow us to see how fast different hardware solutions scale up, as well as give us insight into how well different content generation algorithms and templating engines perform.

Sample data includes:

- Text file, 35 kB [25]. Beyond this point referred to as “gpl”.
- A dummy text file of size 1kB (beyond this point referred to as “1024”) generated from the abovementioned file using the command:

  ```
  $ head -c 1k < gpl-3.0.txt > 1024.txt
  ```
- Text file, 92 kB [26]. Beyond this point referred to as “communist”.
- Text file, 164 kB [27]. Beyond this point referred to as “alice”.
- Text file, 292 kB [28]. Beyond this point referred to as “jungle”.
- Text file, 581 kB [29]. Beyond this point referred to as “sherlock”.

4.2 Operating system

When dealing with low-power hardware systems it is critical to lower the consequences of hardware bottlenecks to achieve better and more accurate results. To minimise the impact of traditional storage media I/O limitations, Tiny Core Linux was selected as the underlying base OS. The main thing that differentiates it from traditional Linux distributions is that it is configured to execute all software components from RAM once started. This means that in its default configuration, all parts of the operating system are read from a persistent storage (HDD, SD card, etc.) and copied to RAM. This helps achieve remarkable speeds and is extremely helpful during the tests as performance can only be influenced by either the network interface, RAM or CPU speed. Furthermore, the OS boots to the same configuration every time, so in case some form of content caching is performed by any of the CMS-s, a system reboot will restore everything to its original state. The principal diagram describing Tiny Core Linux boot process [30] is available on the project's website.

Another thing that makes Tiny Core Linux a perfect solution for the performance testing is that it has been ported to x86, x86_64 as well as both ARMv6 and ARMv7 architectures, which allows one to compare CMS performance not only on different Raspberry Pi revisions, but also on PC systems to get a rough estimate of performance loss on low-power hardware.

4.3 Web server

Due to RAM limitations, *Nginx* was first considered as the underlying web server component, because of its small memory footprint and good performance. However, Tiny Core Linux's *php5.tcz* package description [31] failed to provide *PHP* configuration instructions for *Nginx*, so *Apache* with *mod-php5* has been chosen instead.

Using *FastCGI* has been excluded as an option, because in order for it to work, all *PHP* scripts had to be edited by adding a line to the top of the file. Although possible, the author has a strong opinion that these manipulations were not required for the scope of this work and can be carried out in later iterations once the best performing CMS-s are determined to further optimise their performance.

4.4 Network equipment

To rule out the network equipment as the source of latency while requesting web pages, all devices in question were connected to a 1 Gbit/s network switch. Raspberry Pi systems both have 100 Mbit/s network adapters, so it is fair to say that the network infrastructure is not a limiting factor during the testing phase.

4.5 CPU load

To examine the CPU load during the tests, *top* [32], a standard *UNIX* utility will be used. It will allow us to observe momentary CPU load as well as the average load during the testing.

4.6 Memory usage

In order to find out memory usage, *free* [33], again, a standard *UNIX* monitoring utility, will be used. It will help us determine how much RAM is available at system boot, and how much resources a particular CMS requires.

4.7 Network monitoring

Monitoring the network performance on the devices will be performed by *IPTraf* [34], an interactive open source tool for monitoring network interface performance. It is supposed to provide us with the statistics of the network interface such as its throughput at any given time.

4.8 CMS performance monitoring and stress testing

To stress-test the systems in question we will be using an open source tool called *ApacheBench* [35]. *Apache JMeter* [36] was also considered as an alternative, however, initial tests showed similar results using both tools, so the author decided to use *ApacheBench* for its easy scripting capabilities and immediate availability as part of the *Apache* software package. It will allow us to specify the exact number of concurrent connections we want to simulate as well as provide us with detailed output about the test results.

5 Performance results

Tests showed that *ApacheBench* performance results were different from real-world page loading times. For this reason, *Google Chrome 50.0.2661.94m* page loading times are also included in the tables. It is important to note that these results show the loading times with 1 concurrent connection only, and serve as just real-world performance results for any given CMS and source file size.

5.1 CPU performance

Several observations have been made throughout the tests regarding CPU load.

While the initial tests to measure canonical performance passed very quickly, the CPU load spiked for such short periods of time that the measurements could not be considered reliable. At higher concurrency rates, the CPU load was reported between 80-90%.

During later tests of the actual CMS-s, the CPU load on Raspberry Pi 1 was immediately above 90% for even the smallest sample file, so further testing was not performed for stability and accuracy reasons.

The Raspberry Pi 2, on the other hand, was clearly taking advantage of its 4 CPU cores. At concurrency rates of 1 and 2, the CPU load was reported as being 25% and 50% respectively. CPU load went over 90% only in case of 4 concurrent connections. This leads the author to believe that the Raspberry Pi 2 could be better suited for small business environments, where there are several concurrent connections to a given web resource.

5.2 Network performance

Conducting the network monitoring had to be cancelled, as running such resource-intensive software on the Raspberry Pi systems resulted in instability and software crashes. As a workaround, to rule out the network interface speed limits as a factor that could influence test results, the x86 system was also equipped with a 10/100 Mbit/s network adapter.

Based on the CPU performance results, the author considers it unlikely that the network interface could have been the bottleneck during the CMS testing process.

5.3 Memory usage

Table 3 shows memory consumption (MB) at different stages of testing. All data was acquired by running the command “free -m” after certain steps.

Table 3. Memory usage at different stages of testing.

Stage	Raspberry Pi 1	Raspberry Pi 2	X86
On boot	Used 53	Used 59	Used 21
Apache started	Used 74	Used 78	Used 65
Apache + data	Used 75	Used 81	Used 67
Bludit + data	Used 144	Used 168	Used 104
Htmly + data (1-4)	Used 126	Used 145	Used 94
Mecha + data (1-5)	Used 158	Used 180	Used 114
Phile + data	Used 79	Used 84	Used 76
Pico + data	Used 82	Used 87	Used 74
PuppyCMS + data	Used 78	Used 83	Used 69
Singularity + data	Used 77	Used 82	Used 66
Yellow + data	Used 90	Used 95	Used 70

5.4 Canonical performance

The initial *Apache* sample page named *index.html* of size 55 B with the contents “<html><body><h1>It works!</h1></body></html>” was chosen as a model for measuring the loading times. Its loading time is considered the standard for *Apache* performance on the given hardware in the course of this work. It is to show the network and web server performance limitations of the lab setup.

After determining the maximum performance of the test setups, the sample files were copied to the web server root folder and also tested to see the ideal performance for the source files without any conversion or file generation taking place.

Canonical performance results for the Raspberry Pi 1 are listed in Appendix 1. Table 1.

Canonical performance results for the Raspberry Pi 2 are listed in Appendix 1. Table 2.

Canonical performance results for the x86 system are listed in Appendix 1. Table 3.

5.5 Bludit

One important observation is that the loading times of “gpl” were slower than the one of a bigger file “communist”. This result was tested multiple times, however, each time this peculiar behaviour was observed. The nature of this phenomenon is unknown to the author and can be further investigated in later iterations of this work.

Bludit performance results for the Raspberry Pi 1 are listed in Appendix 1. Table 4.

Bludit performance results for the Raspberry Pi 2 are listed in Appendix 1. Table 5.

Bludit performance results for the x86 system are listed in Appendix 1. Table 6.

5.6 HTMLy

Upon initial page load for file “1024” in HTMLy, the loading times were very high. Subsequent page loading times were much smaller, with caching disabled on the browser side. This suggests that HTMLy performs internal optimisation which greatly improves overall performance. Table 4 shows the initial loading times (ms) for file “1024”.

Table 4. Initial loading times for file “1024” in Bludit.

	1024
Raspberry Pi 1	11330.000
Raspberry Pi 2	21440.000
x86	7240.000

When trying to add the last 2 sample files, the system was rendered completely unusable, so the tests had to be conducted only for the first 4 sample files.

HTMLy performance results for the Raspberry Pi 1 are listed in Appendix 1. Table 7.

HTMLy performance results for the Raspberry Pi 2 are listed in Appendix 1. Table 8.

HTMLy performance results for the x86 system are listed in Appendix 1. Table 9.

5.7 Mecha

When trying to add the last sample file, the system was rendered completely unusable, so the tests had to be conducted only for the first 5 sample files.

Mecha performance results for the Raspberry Pi 1 are listed in Appendix 1. Table 10.

Mecha performance results for the Raspberry Pi 2 are listed in Appendix 1. Table 11.

Mecha performance results for the x86 system are listed in Appendix 1. Table 12.

5.8 Phile

No particularly interesting observations were made while testing Phile. The results were predictable, no anomalies found.

Phile performance results for the Raspberry Pi 1 are listed in Appendix 1. Table 13.

Phile performance results for the Raspberry Pi 2 are listed in Appendix 1. Table 14.

Phile performance results for the x86 system are listed in Appendix 1. Table 15.

5.9 Pico

One important observation is that the loading times of “gpl” were slower than the one of a bigger file “communist”. This result was tested multiple times, however, each time this peculiar behaviour was observed. The nature of this phenomenon is unknown to the author and can be further investigated in later iterations of this work.

Pico performance results for the Raspberry Pi 1 are listed in Appendix 1. Table 16.

Pico performance results for the Raspberry Pi 2 are listed in Appendix 1. Table 17.

Pico performance results for the x86 system are listed in Appendix 1. Table 18.

5.10 PuppyCMS

As with Singularity, *ApacheBench* tool results were different from the actual complete page load times. This is due to the fact that the pages were being loaded asynchronously.

PuppyCMS performance results for the Raspberry Pi 1 are listed in Appendix 1. Table 19.

PuppyCMS performance results for the Raspberry Pi 2 are listed in Appendix 1. Table 20.

PuppyCMS performance results for the x86 system are listed in Appendix 1. Table 21.

5.11 Singularity

As with PuppyCMS, *ApacheBench* tool results were different from the actual complete page load times. This is due to the fact that the pages were being loaded asynchronously.

Singularity performance results for the Raspberry Pi 1 are listed in Appendix 1. Table 22.

Singularity performance results for the Raspberry Pi 2 are listed in Appendix 1. Table 23.

Singularity performance results for the x86 system are listed in Appendix 1. Table 24.

5.12 Yellow

No particularly interesting observations were made while testing Yellow. The results were predictable, no anomalies found.

Yellow performance results for the Raspberry Pi 1 are listed in Appendix 1. Table 25.

Yellow performance results for the Raspberry Pi 2 are listed in Appendix 1. Table 26.

Yellow performance results for the x86 system are listed in Appendix 1. Table 27.

6 Performance graphs

The following graphs present the loading times of different CMS-s as well as canonical performance for each individual sample data file. Results exceeding 10s have a negative value to make the differences between relative results clearly visible. In cases where tests have not been performed (predictable results exceeding 10s, software crashes due to performance issues) the results are omitted. Due to the fact that the loading times were growing at a steady and predictable pace beyond 4 simultaneous connections on all tested platforms, the graphs only show the loading times in *Chrome* and *ApacheBench* results for concurrency rates of 1, 2 and 4.

6.1 1024

Figure 1 shows page loading times in *Chrome* for all tested platforms.

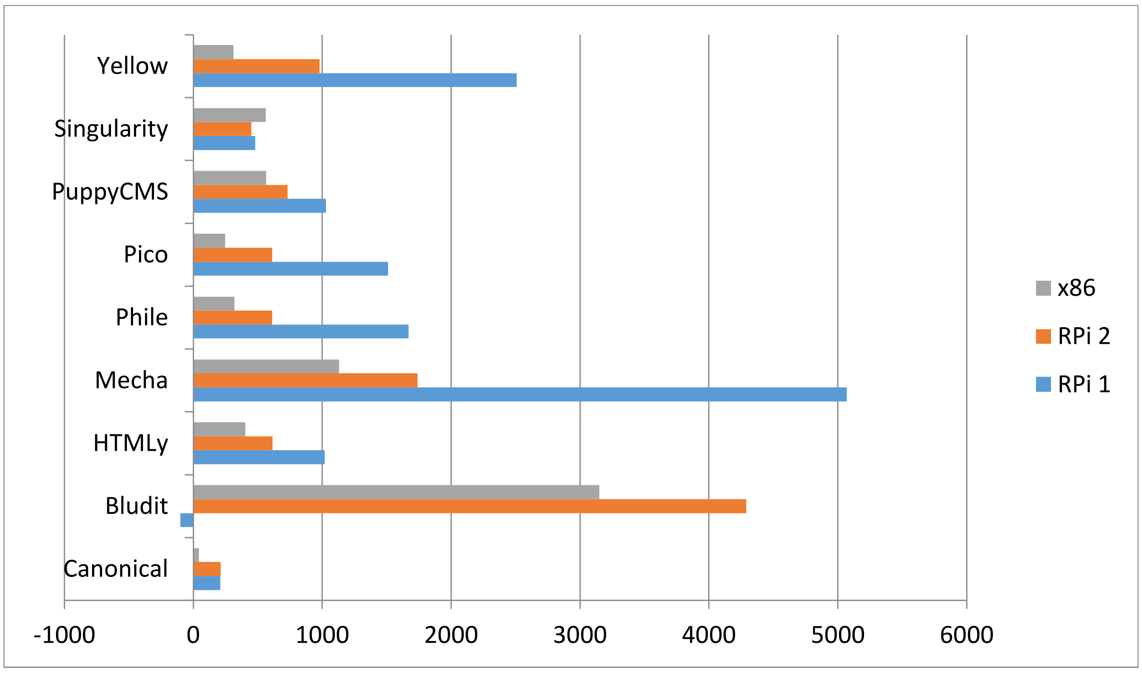

Figure 1. Page loading times in *Chrome* for sample file “1024”.

Singularity managed to perform better on both Raspberry Pi models compared to the x86 system. Canonical performance on the x86 system was noticeably better. Raspberry Pi 1 results are noticeably worse in almost all cases, except Singularity. Singularity is a clear winner on both Raspberry Pi systems. HTMLy is the second fastest for the Raspberry Pi systems. Pico and Phile outperformed PuppyCMS on the Raspberry Pi 2, although they were slower on the Raspberry Pi 1.

Figure 2 shows average loading times (time per request, mean; ms) with concurrency of 1 for all tested platforms.

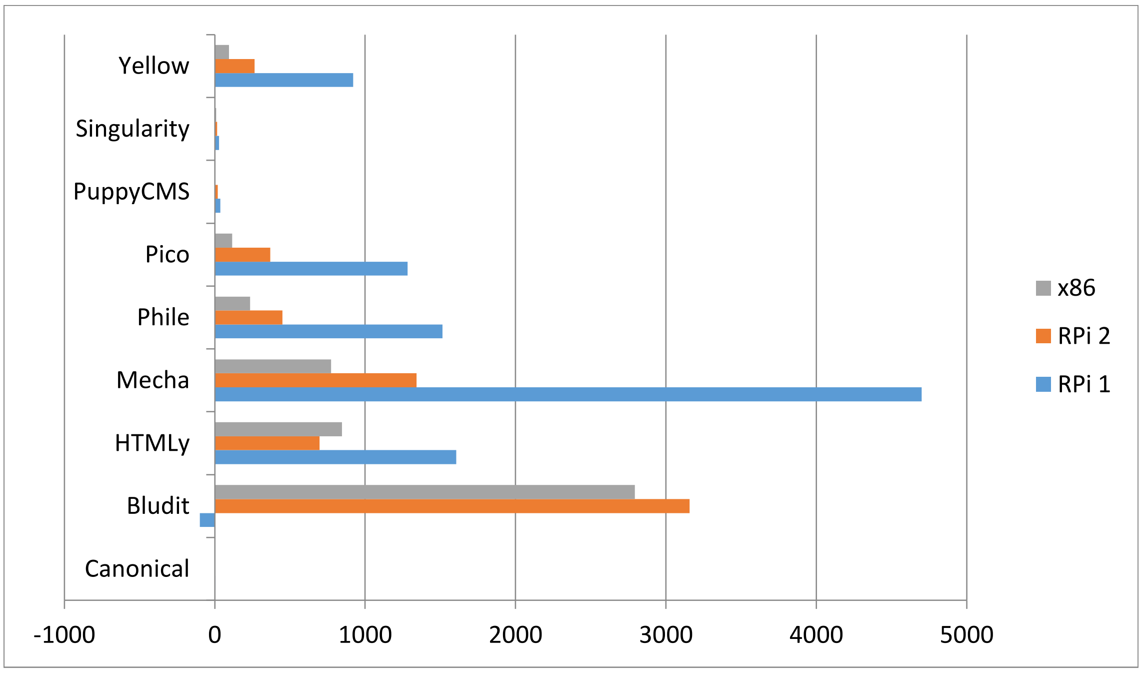

Figure 2. Average loading times for sample file "1024" with concurrency of 1.

Yellow was the best-performing CMS after Singularity and PuppyCMS, for which the results cannot be considered trustworthy for the reason described in 5.10 and 5.11. HTMLy performed better on the Raspberry Pi 2 compared to the x86 system. Bludit performed nearly the same on the Raspberry Pi 2 and the x86 system. Raspberry Pi 1 results were noticeably worse in almost all cases.

Figure 3 shows average loading times (time per request, mean; ms) with concurrency of 2 for all tested platforms.

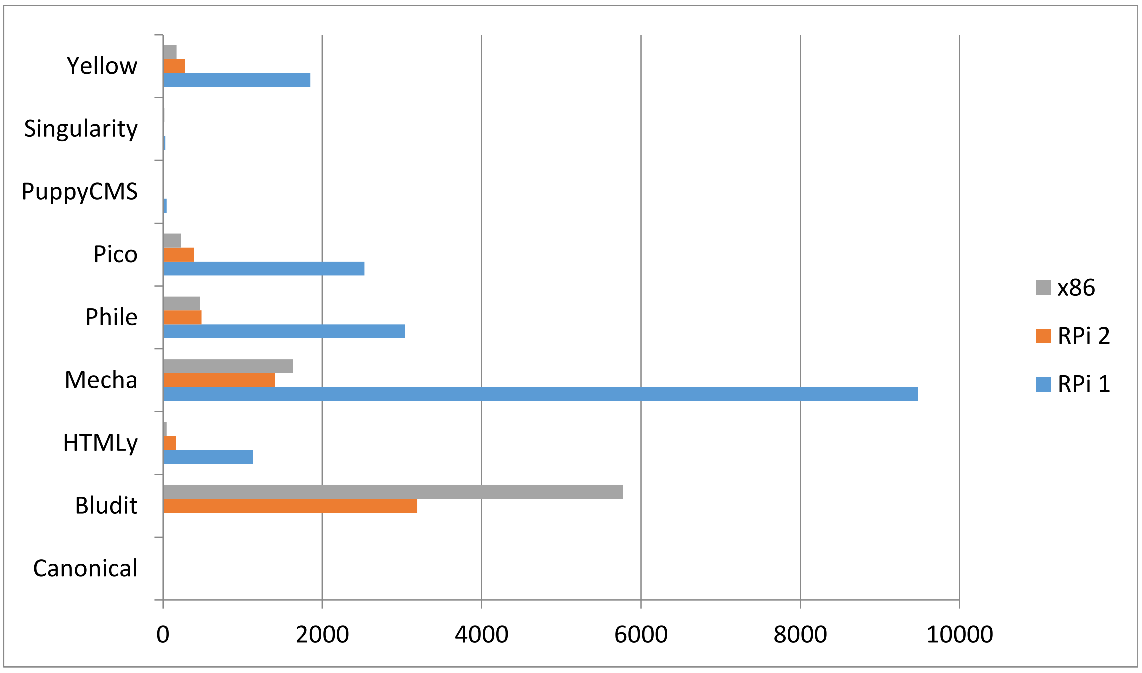

Figure 3. Average loading times for sample file "1024" with concurrency of 2.

Bludit performed almost twice as fast on the Raspberry Pi 2 compared to the x86 system. Mecha was also slower on the x86 system compared to the Raspberry Pi 2. Phile performance was almost the same on the x86 system and the Raspberry Pi 2. Raspberry Pi 1 results were noticeably worse in almost all cases.

Figure 4 shows average loading times (time per request, mean; ms) with concurrency of 4 for all tested platforms.

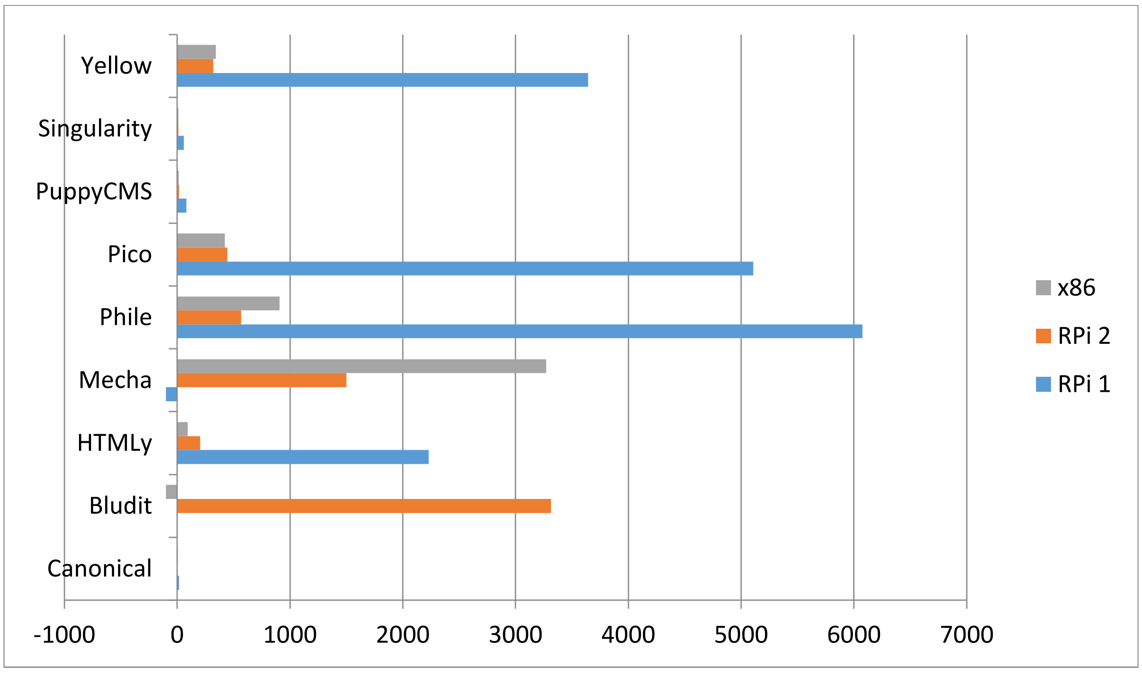

Figure 4. Average loading times for sample file “1024” with concurrency of 4.

Here we can already see the advantages of the 4 cores of the Raspberry Pi 2. Phile, Yellow and Mecha perform better on the Raspberry Pi 2 compared to the x86 system, whereas Mecha is almost twice as fast. Raspberry Pi 1 results were noticeably worse in almost all cases.

6.2 Gpl

Figure 5 shows page loading times in *Chrome* for all tested platforms.

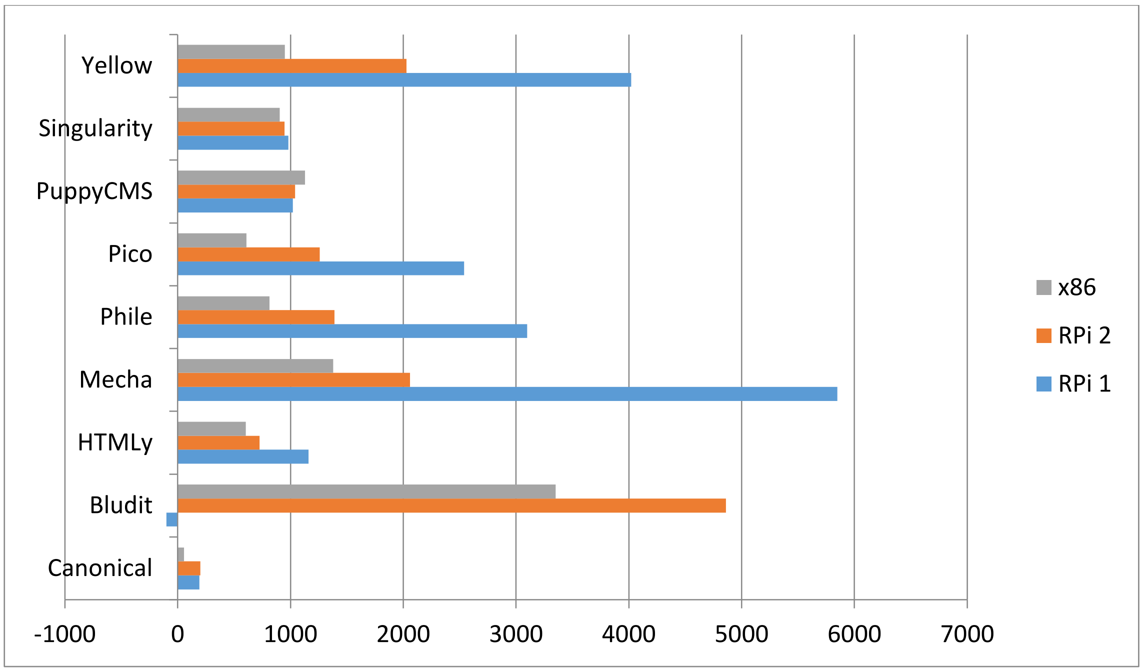

Figure 5. Page loading times in *Chrome* for sample file “gpl”.

HTMLy was the fastest performing CMS on the Raspberry Pi 2, while Singularity and PuppyCMS were clear winners on the Raspberry Pi 1 with HTMLy not far behind. The author has reasons to believe this is due to HTMLy’s internal optimisation mechanism mentioned in 5.6. Singularity and PuppyCMS showed almost the same results on all platforms. In the case of other CMS-s, the Raspberry Pi 1 performed noticeably worse.

Figure 6 shows average loading times (time per request, mean; ms) with concurrency of 1 for all tested platforms.

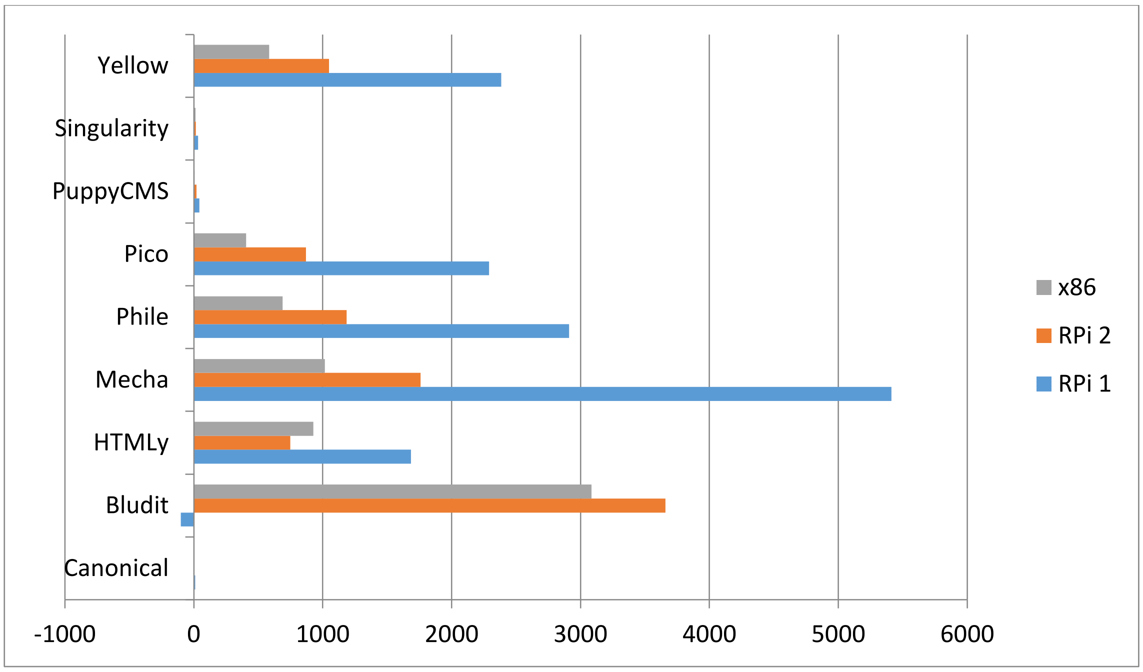

Figure 6. Average loading times for sample file "gpl" with concurrency of 1.

HTMLy performed better on the Raspberry Pi 2 compared to the x86 system. In all other cases, the x86 system was faster than the Raspberry Pi 2. Raspberry Pi 1 results were noticeably worse in almost all cases.

Figure 7 shows average loading times (time per request, mean; ms) with concurrency of 2 for all tested platforms.

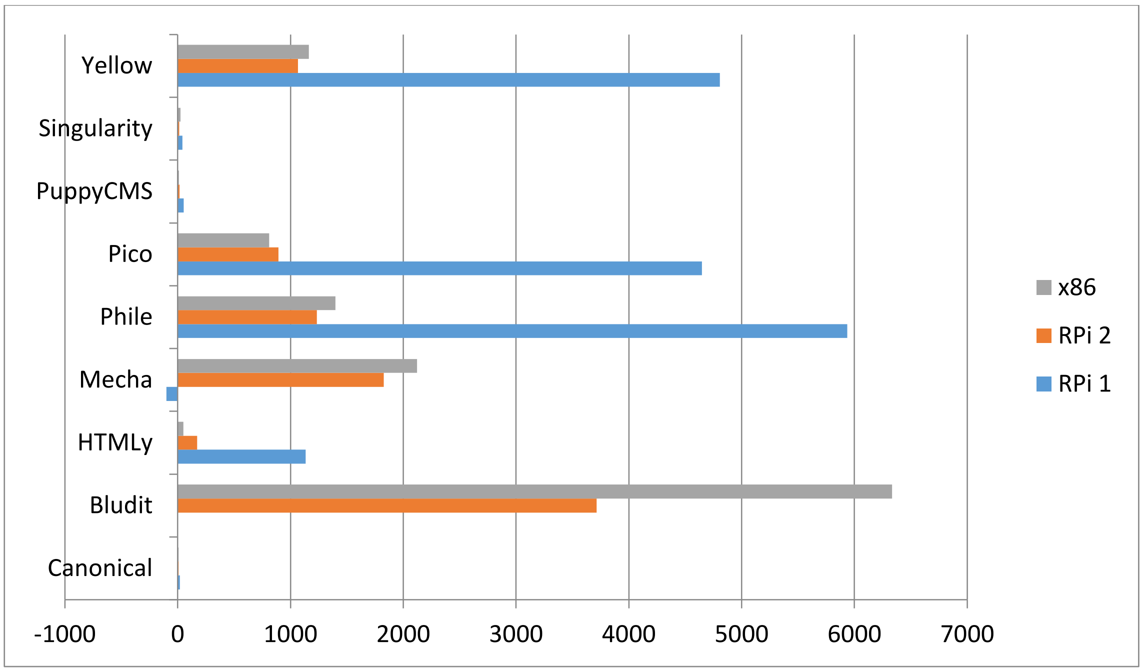

Figure 7. Average loading times for sample file "gpl" with concurrency of 2.

Yellow, Phile, Mecha and Bludit performed better on the Raspberry Pi 2 compared to the x86 system, Bludit results being noticeably better. Raspberry Pi 1 results were noticeably worse in almost all cases.

Figure 8 shows average loading times (time per request, mean; ms) with concurrency of 4 for all tested platforms.

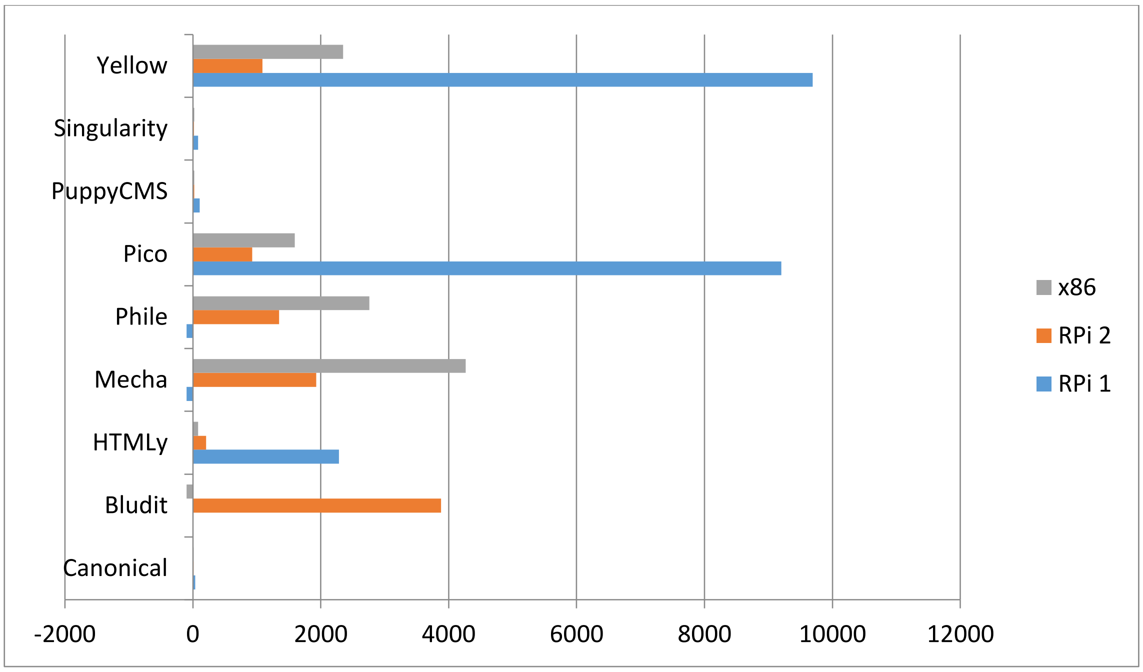

Figure 8. Average loading times for sample file "gpl" with concurrency of 4.

Again, we can see the advantage of the 4 CPU cores of the Raspberry Pi 2. Yellow, Pico, Phile, Mecha and Bludit perform noticeably (almost 2 times, except Pico) better on the Raspberry Pi 2 compared to the x86 system. Raspberry Pi 1 results were noticeably worse in almost all cases.

6.3 Communist

Figure 9 shows page loading times in *Chrome* for all tested platforms.

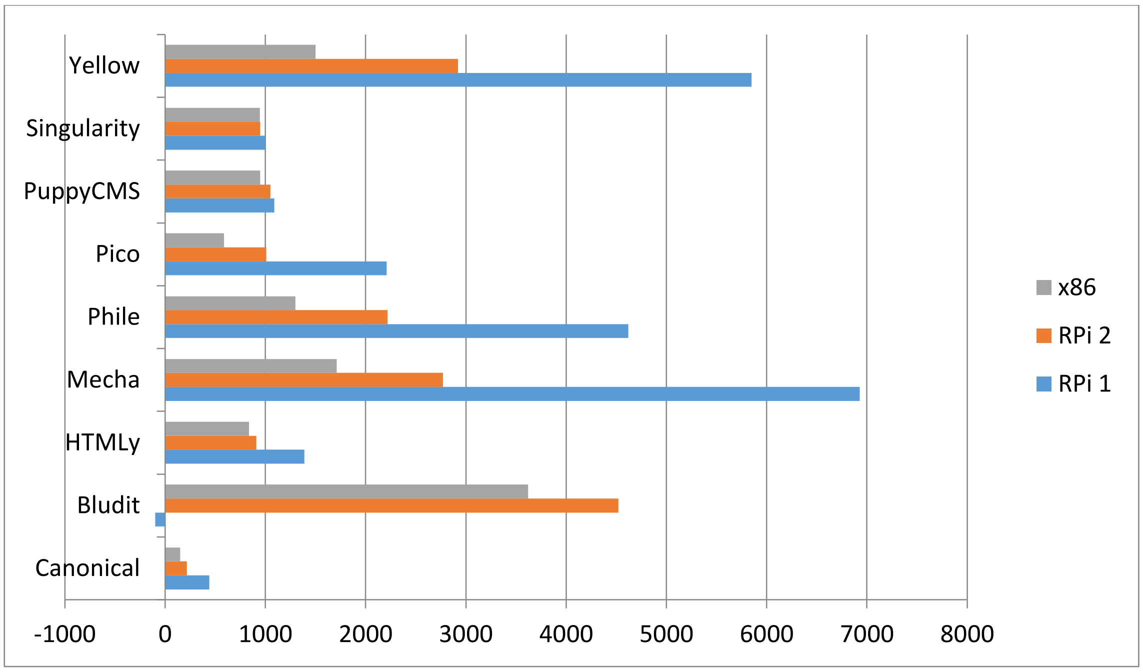

Figure 9. Page loading times in *Chrome* for sample file "communist".

HTMLy was the fastest performing CMS on the Raspberry Pi 2, with Singularity, Pico and PuppyCMS not far behind. Singularity was the fastest performing CMS on the Raspberry Pi 1, followed closely by PuppyCMS and HTMLy. Singularity and PuppyCMS showed almost the same results on all platforms. In the case of other CMS-s, the Raspberry Pi 1 performed noticeably worse.

Figure 10 shows average loading times (time per request, mean; ms) with concurrency of 1 for all tested platforms.

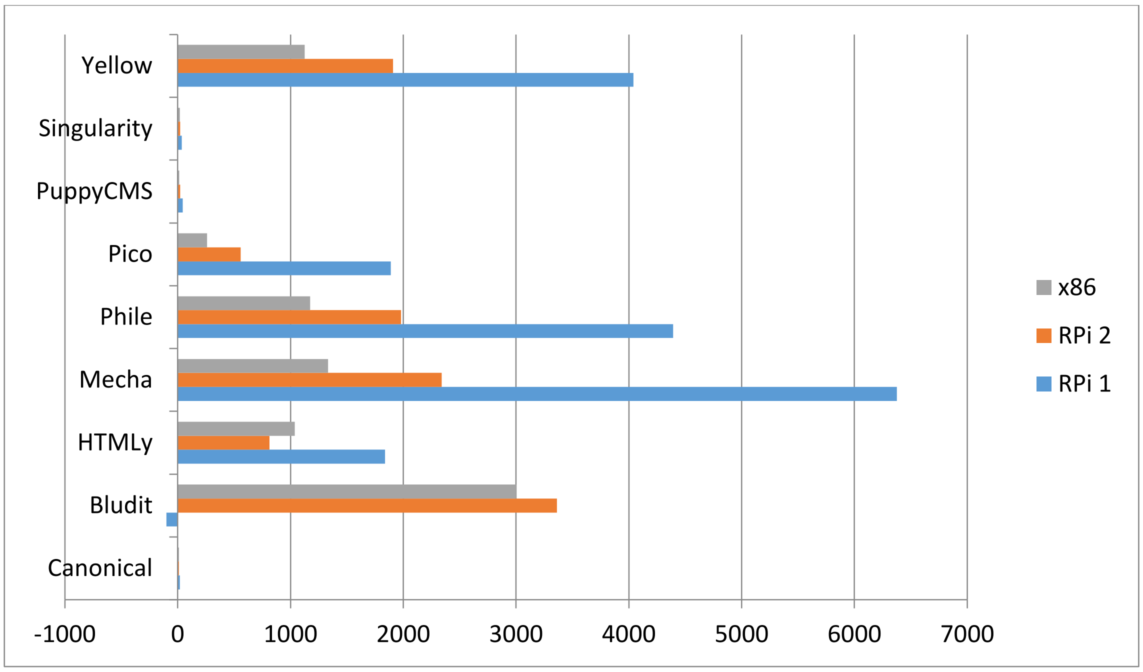

Figure 10. Average loading times for sample file “communist” with concurrency of 1.

HTMLy performed better on the Raspberry Pi 2 compared to the x86 system. In all other cases, the x86 system was faster than the Raspberry Pi 2. Raspberry Pi 1 results were noticeably worse in almost all cases.

Figure 11 shows average loading times (time per request, mean; ms) with concurrency of 2 for all tested platforms.

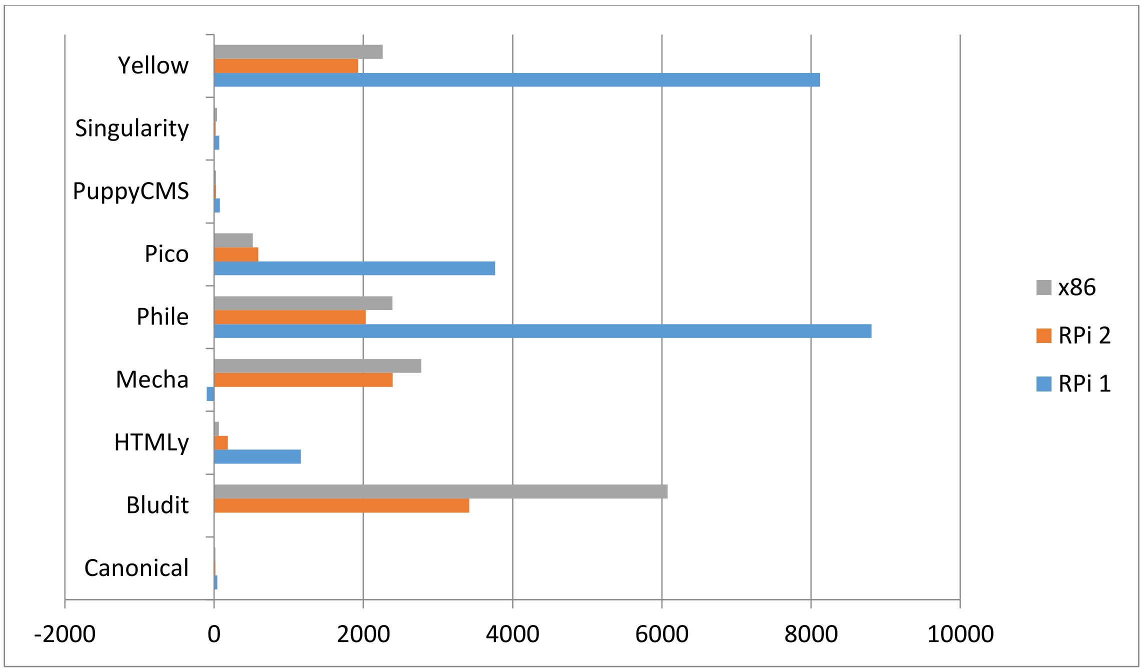

Figure 11. Average loading times for sample file “communist” with concurrency of 2.

Yellow, Phile, Mecha and Bludit performed better on the Raspberry Pi 2 compared to the x86 system, Bludit results being noticeably better. Raspberry Pi 1 results were noticeably worse in almost all cases.

Figure 12 shows average loading times (time per request, mean; ms) with concurrency of 4 for all tested platforms.

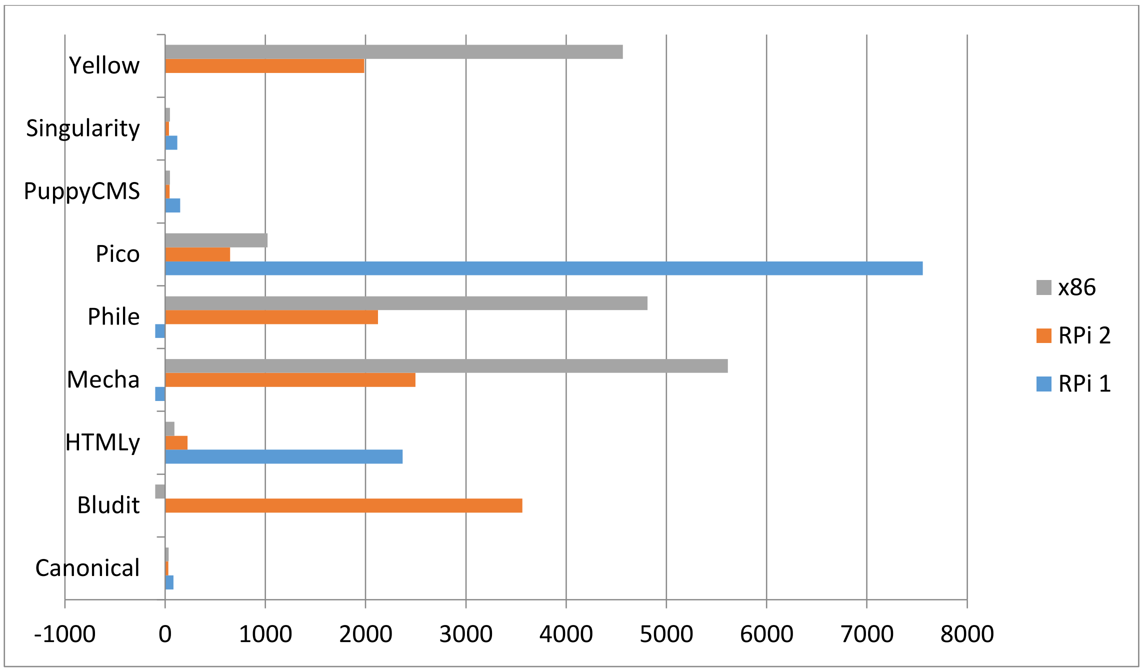

Figure 12. Average loading times for sample file “communist” with concurrency of 4.

Again, we can see the advantage of the 4 CPU cores of the Raspberry Pi 2. Yellow, Pico, Phile, Mecha and Bludit performed noticeably (over 2 times, except Pico) better on the Raspberry Pi 2 compared to the x86 system. Raspberry Pi 1 results were noticeably worse in all cases.

6.4 Alice

Figure 13 shows page loading times in *Chrome* for all tested platforms.

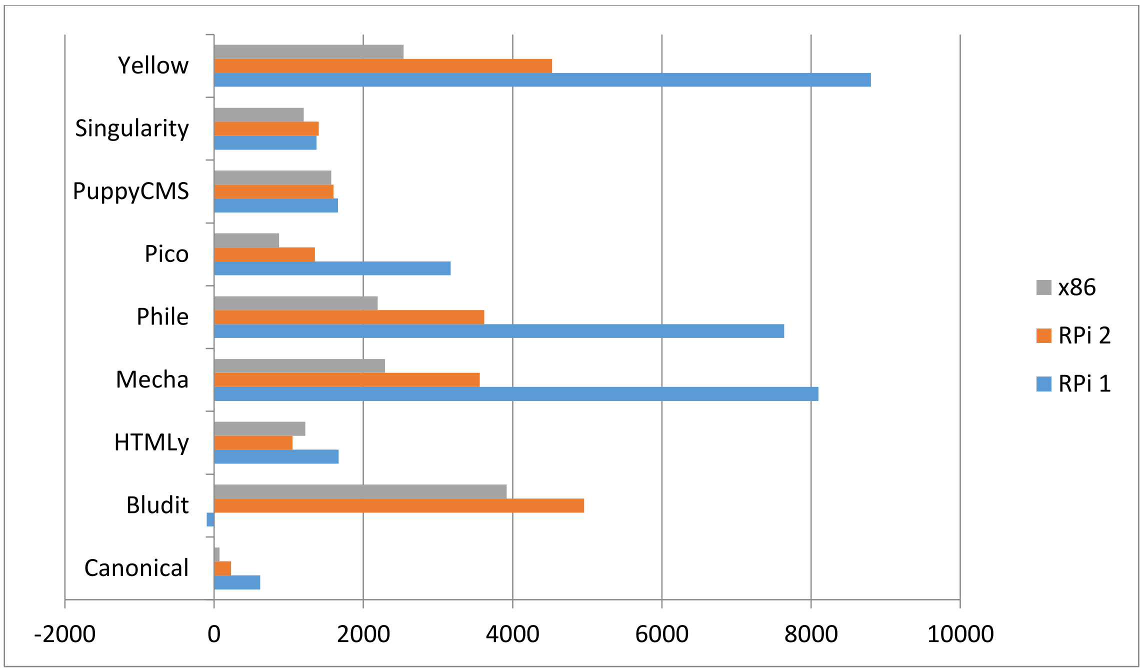

Figure 13. Page loading times in *Chrome* for sample file “alice”.

HTMLy was the fastest performing CMS on the Raspberry Pi 2, with Pico, Singularity and PuppyCMS not far behind. Singularity was the fastest performing CMS on the Raspberry Pi 1, followed closely by PuppyCMS and HTMLy. Singularity and PuppyCMS showed almost the same results on all platforms. In the case of other CMS-s, the Raspberry Pi 1 performed noticeably worse.

Figure 14 shows average loading times (time per request, mean; ms) with concurrency of 1 for all tested platforms.

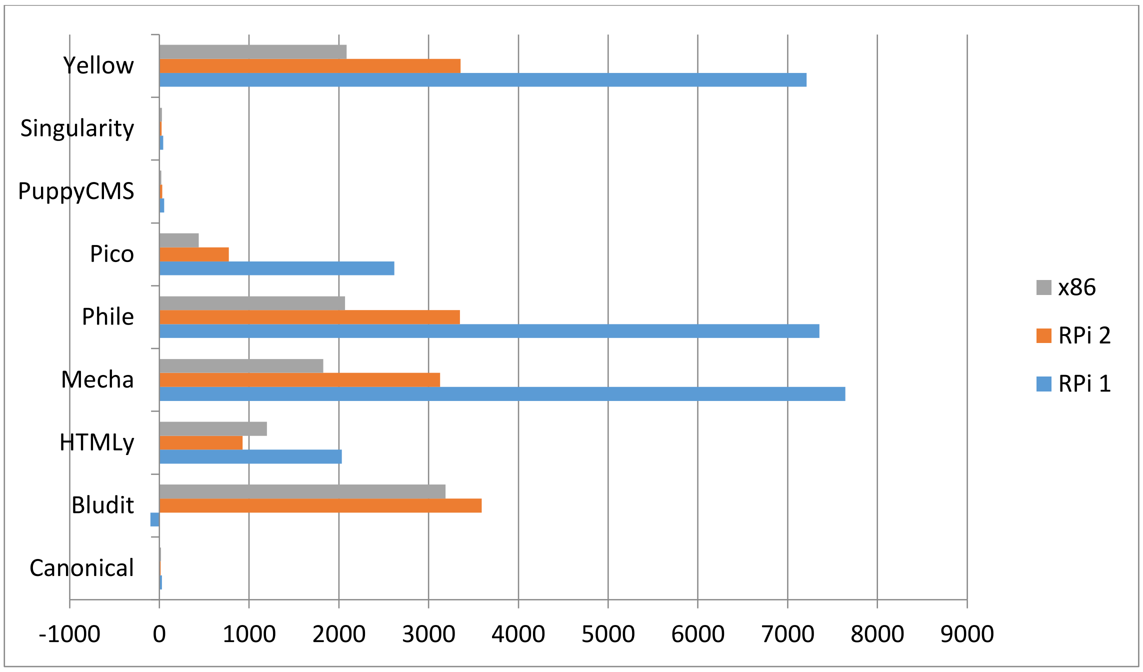

Figure 14. Average loading times for sample file "alice" with concurrency of 1.

HTMLy performed better on the Raspberry Pi 2 compared to the x86 system. In all other cases, the x86 system was faster than the Raspberry Pi 2. Raspberry Pi 1 results were noticeably worse in almost all cases.

Figure 15 shows average loading times (time per request, mean; ms) with concurrency of 2 for all tested platforms.

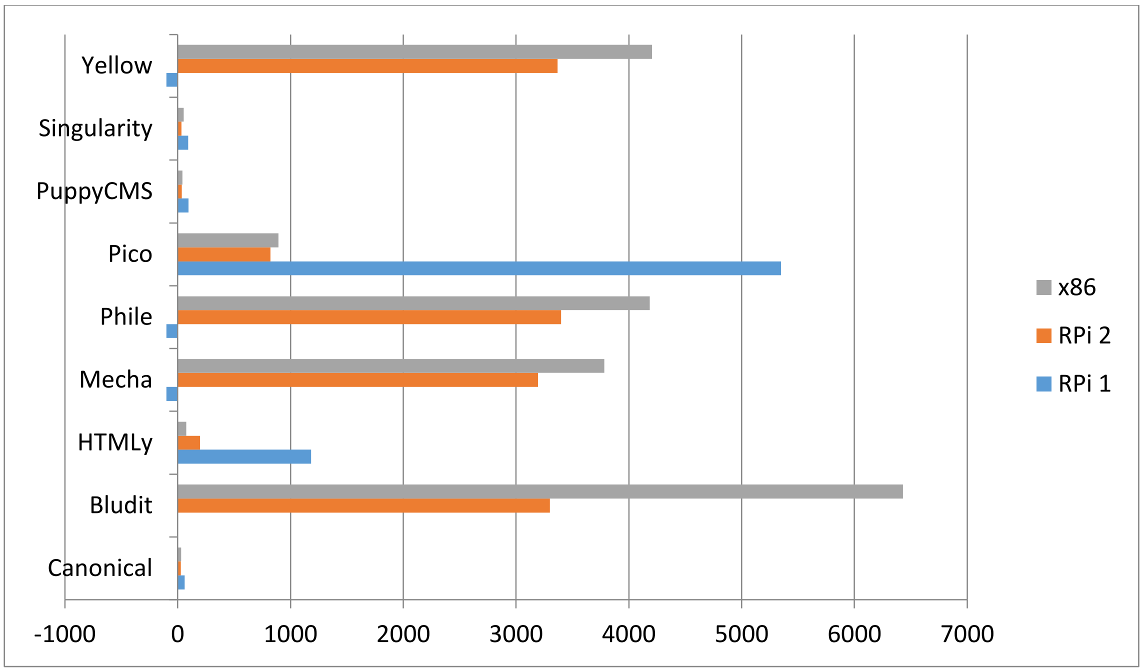

Figure 15. Average loading times for sample file "alice" with concurrency of 2.

Yellow, Pico, Phile, Mecha and Bludit performed better on the Raspberry Pi 2 compared to the x86 system, Bludit results being noticeably better. Raspberry Pi 1 results were noticeably worse in all cases.

Figure 16 shows average loading times (time per request, mean; ms) with concurrency of 4 for all tested platforms.

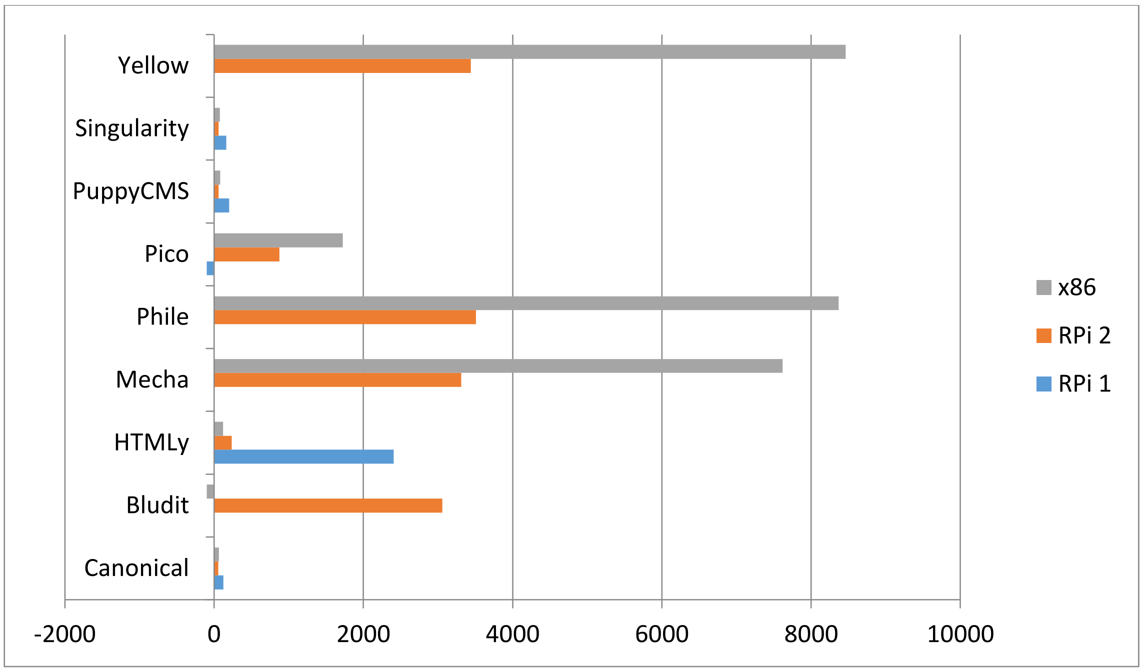

Figure 16. Average loading times for sample file "alice" with concurrency of 4.

Again, we can see the advantage of the 4 CPU cores of the Raspberry Pi 2. Yellow, Pico, Phile, Mecha and Bludit performed noticeably (over 2 times, except Pico) better on the Raspberry Pi 2 compared to the x86 system. Raspberry Pi 1 results were noticeably worse in all cases.

6.5 Jungle

Figure 17 shows page loading times in *Chrome* for all tested platforms.

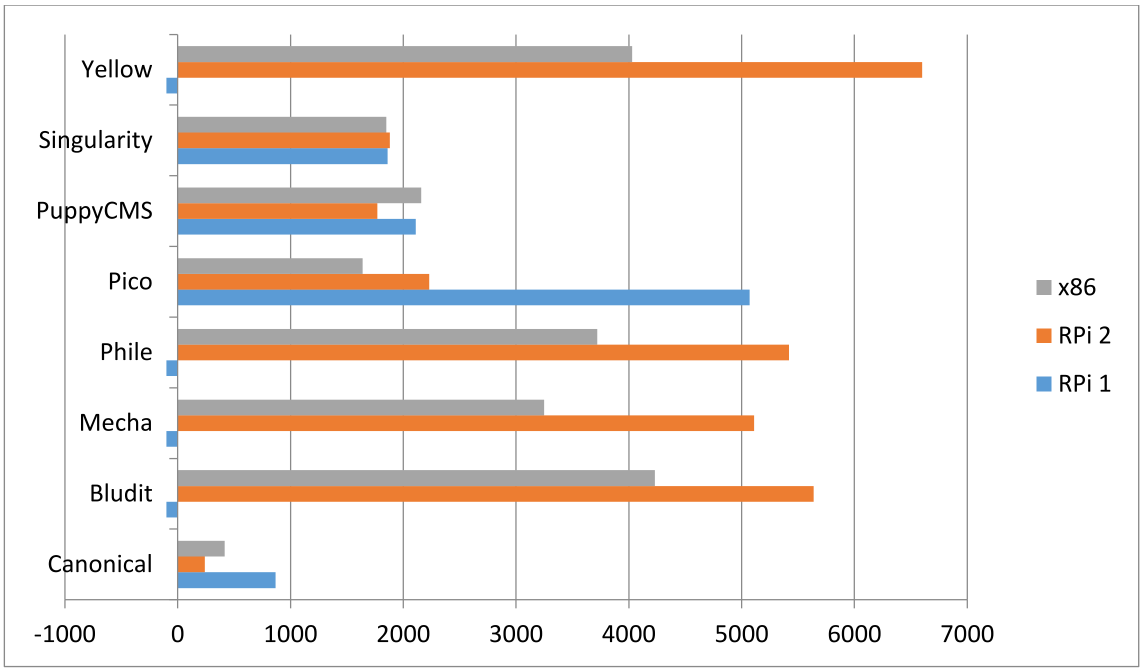

Figure 17. Page loading times in *Chrome* for sample file “jungle”.

PuppyCMS was the fastest performing CMS on the Raspberry Pi 2, closely followed by Singularity and Pico. Singularity, on the other hand, was the fastest CMS on the Raspberry Pi 1, followed by PuppyCMS. Singularity and PuppyCMS showed almost the same results on all platforms. In the case of other CMS-s, the Raspberry Pi 1 performed noticeably worse.

Figure 18 shows average loading times (time per request, mean; ms) with concurrency of 1 for all tested platforms.

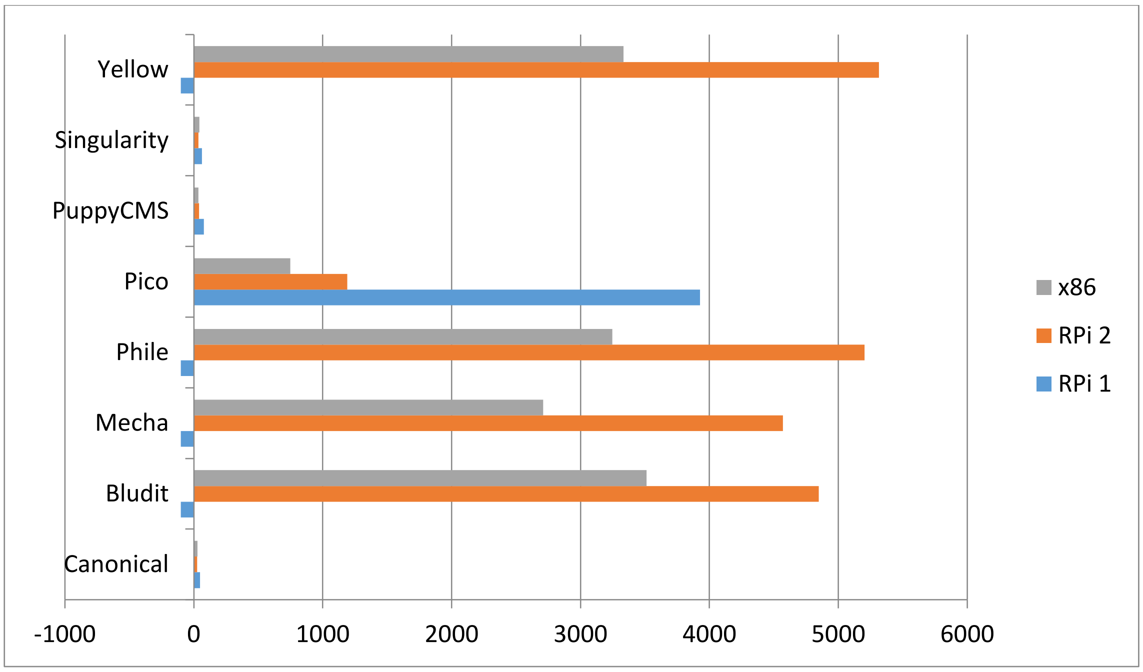

Figure 18. Average loading times for sample file “jungle” with concurrency of 1.

Singularity performed better on the Raspberry Pi 2 compared to the x86 system. In all other cases, the x86 system was faster than the Raspberry Pi 2. Raspberry Pi 1 results were noticeably worse in almost all cases.

Figure 19 shows average loading times (time per request, mean; ms) with concurrency of 2 for all tested platforms.

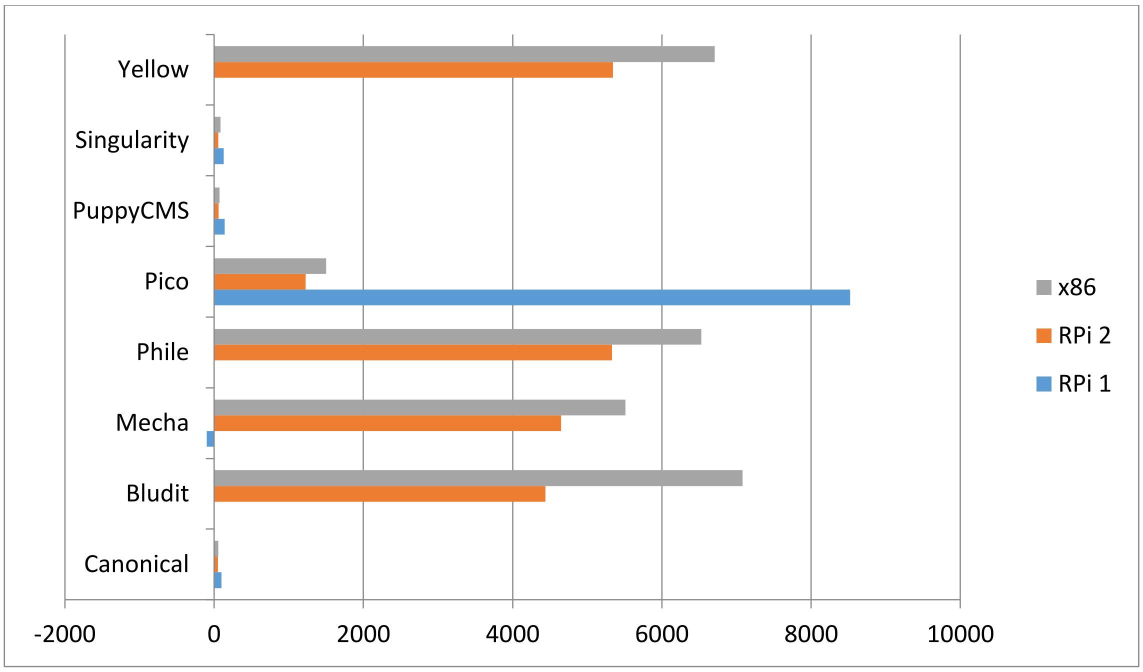

Figure 19. Average loading times for sample file “jungle” with concurrency of 2.

Yellow, Singularity, PuppyCMS, Pico, Phile, Mecha and Bludit performed better on the Raspberry Pi 2 compared to the x86 system, Bludit results being noticeably better. Raspberry Pi 1 results were noticeably worse in all cases.

Figure 20 shows average loading times (time per request, mean; ms) with concurrency of 4 for all tested platforms.

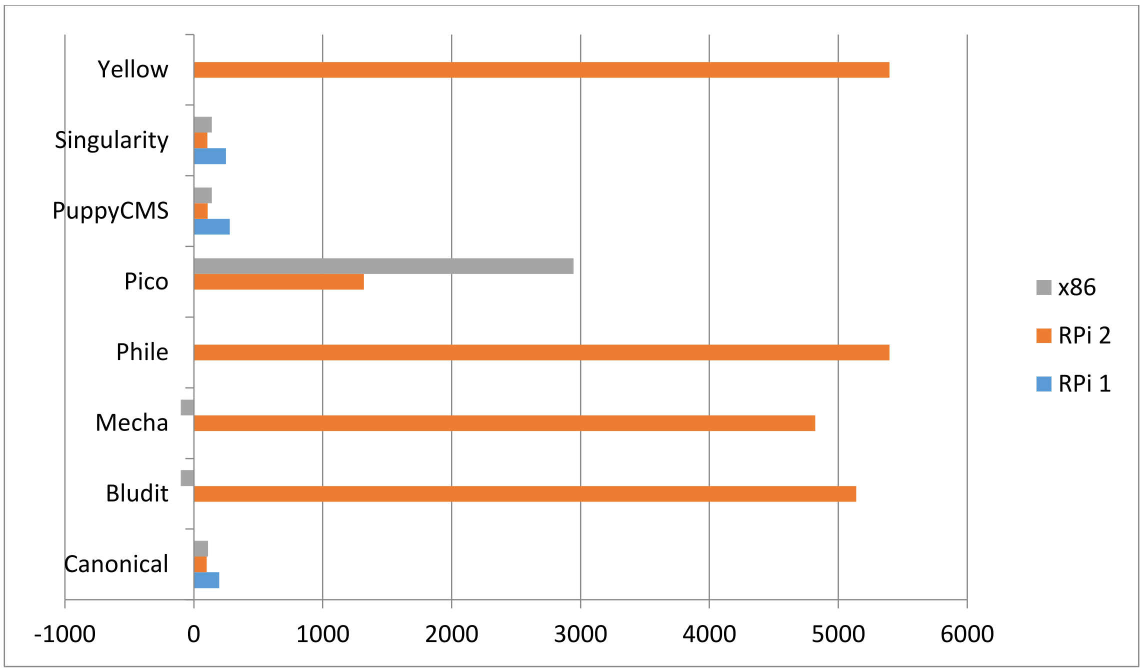

Figure 20. Average loading times for sample file “jungle” with concurrency of 4.

Again, we can see the advantage of the 4 CPU cores of the Raspberry Pi 2. All CMS-s performed almost 2 times better (except PuppyCMS and Singularity) on the Raspberry Pi 2. Raspberry Pi 1 results were noticeably worse in all cases.

6.6 Sherlock

Figure 21 shows page loading times in *Chrome* for all tested platforms.

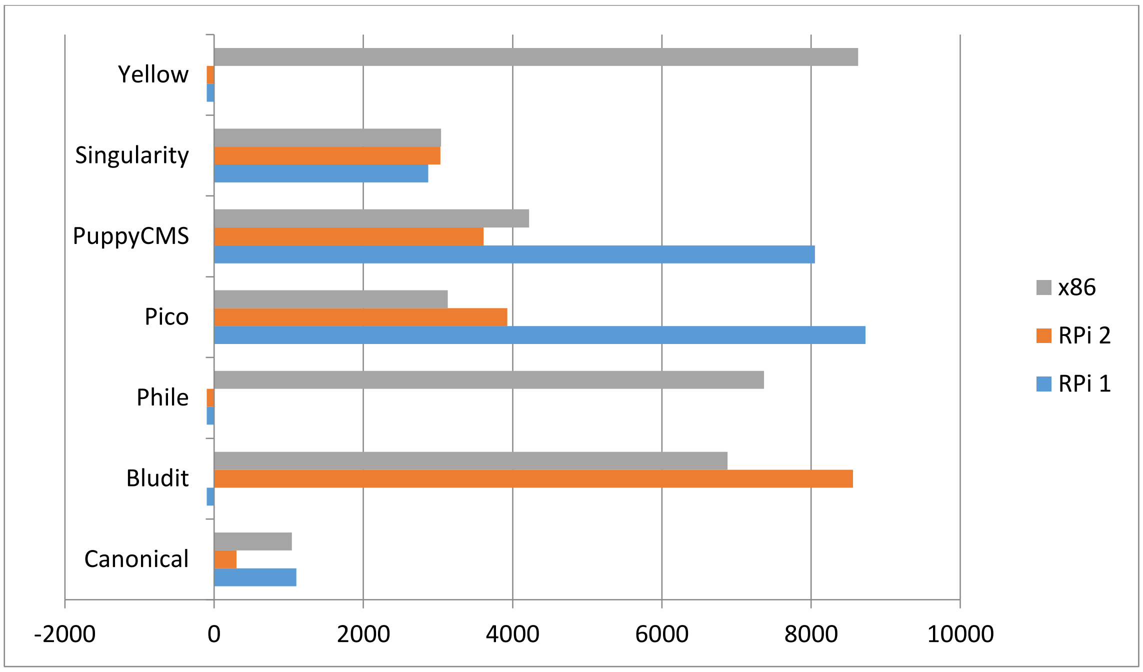

Figure 21. Page loading times in *Chrome* for sample file “sherlock”.

Singularity was the fastest performing CMS on all platforms. It also showed almost the same results on all platforms. In the case of other CMS-s, the Raspberry Pi 1 performed noticeably worse.

Figure 22 shows average loading times (time per request, mean; ms) with concurrency of 1 for all tested platforms.

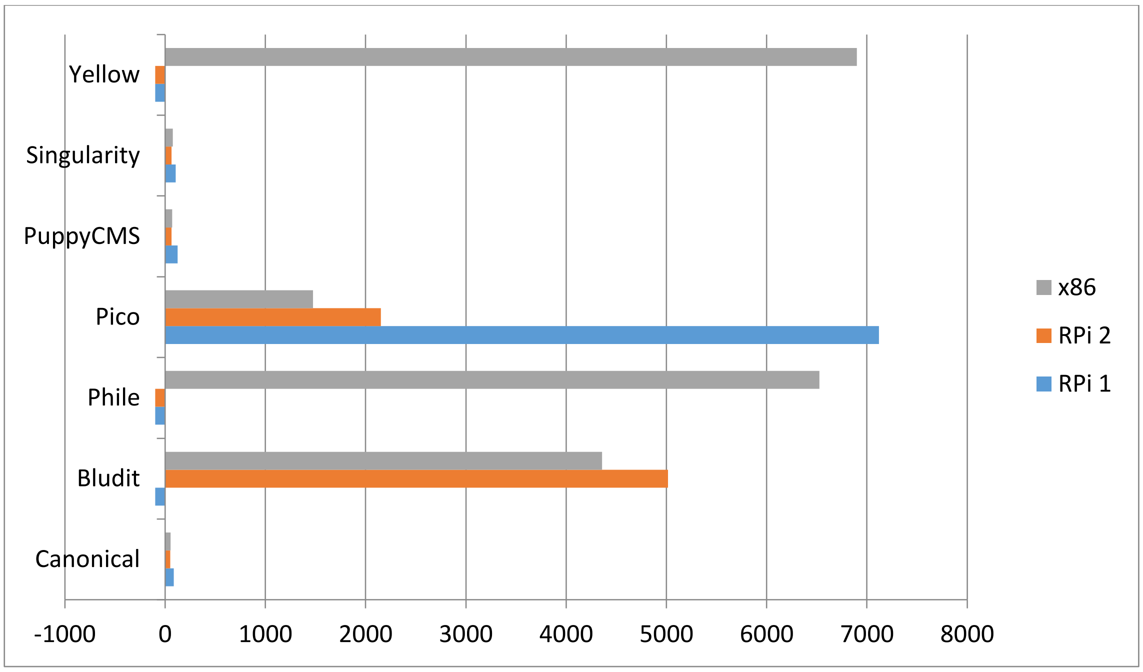

Figure 22. Average loading times for sample file "sherlock" with concurrency of 1.

Singularity and PuppyCMS performed better on the Raspberry Pi 2 compared to the x86 system. In all other cases, the x86 system was faster than the Raspberry Pi 2. Raspberry Pi 1 results were noticeably worse in almost all cases.

Figure 23 shows average loading times (time per request, mean; ms) with concurrency of 2 for all tested platforms.

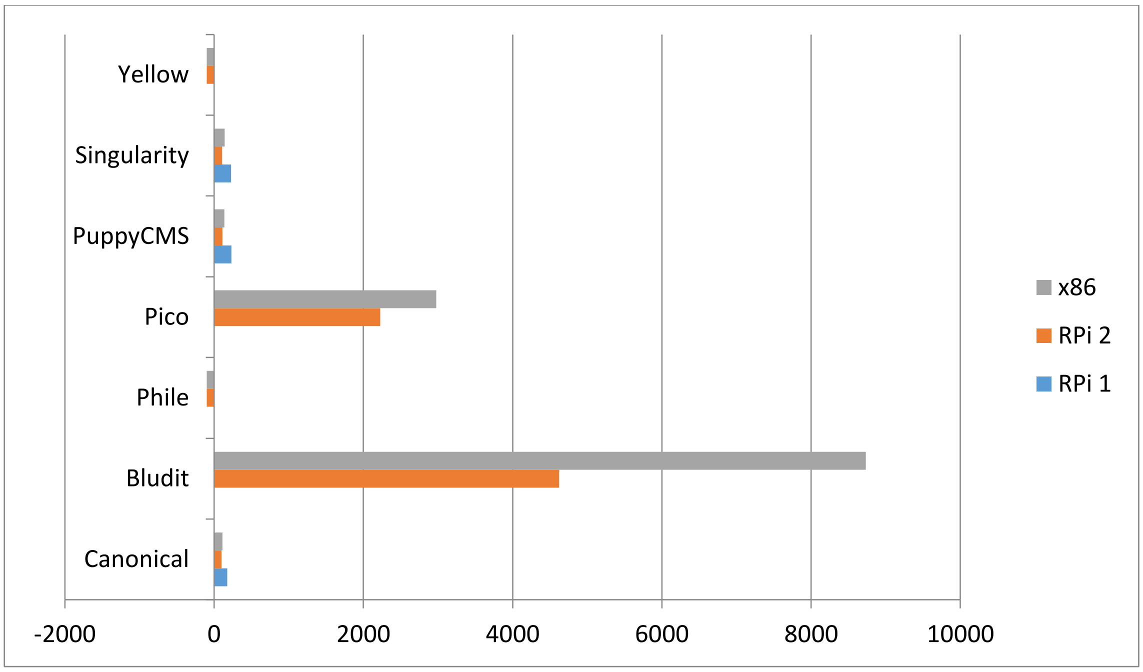

Figure 23. Average loading times for sample file "sherlock" with concurrency of 2.

All CMS-s performed better on the Raspberry Pi 2 compared to the x86 system, Bludit results being noticeably better. Raspberry Pi 1 results were noticeably worse in all cases.

Figure 24 shows average loading times (time per request, mean; ms) with concurrency of 4 for all tested platforms.

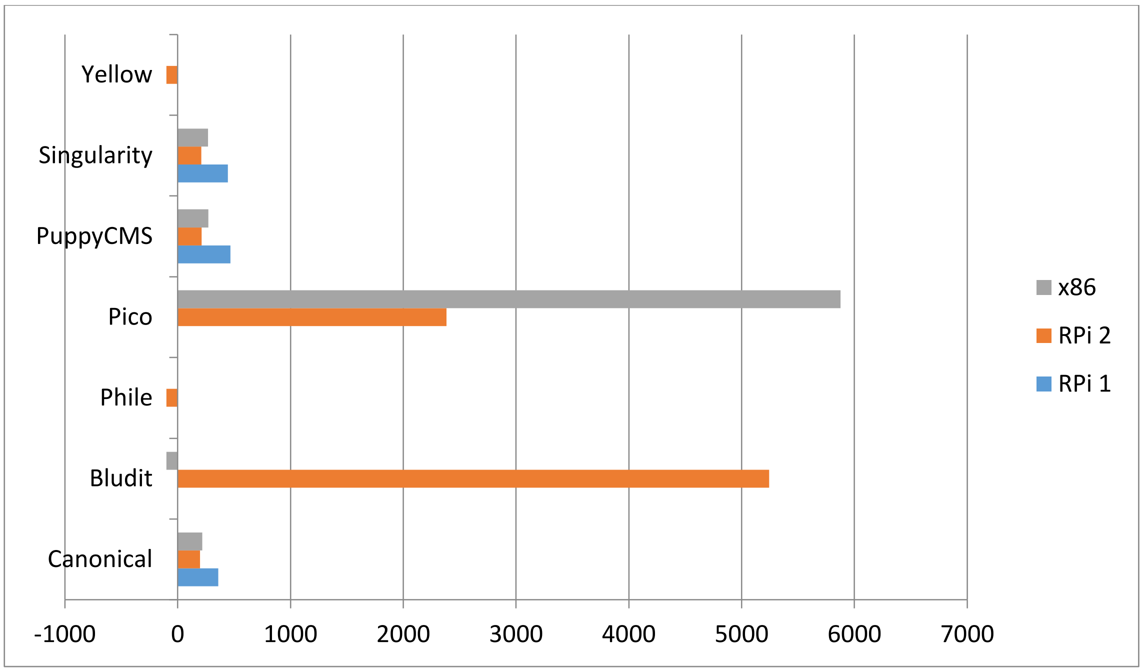

Figure 24. Average loading times for sample file "sherlock" with concurrency of 4.

All CMS-s performed better on the Raspberry Pi 2 compared to the x86 system. Raspberry Pi 1 results were noticeably worse in all cases.

7 Summary

The goal of this work was to analyse the performance of flat-file CMS-s that have *Markdown* support on low-power hardware to determine whether they are a viable option for certain use cases.

Performance metrics, such as web page load times, CPU load and RAM usage were analysed and compared on Raspberry Pi 1 Model B, Raspberry Pi 2 Model B and x86 systems. Overall system responsiveness under higher load was also examined.

Raspberry Pi 1 Model B could deliver reasonable performance only when serving static files and pages. The smallest sample file loading times exceeded 1s for most CMS-s, except Singularity. CPU load was over 90% throughout the tests, average load exceeded 1.00, meaning there were operations constantly waiting for CPU time. RAM usage never exceeded 50%. Multiple concurrent requests often brought the system to a halt, sometimes crashing the *Apache* web server.

Raspberry Pi 2 Model B clearly takes advantage of the 4 CPU cores. In the case of Bludit, Mecha, Phile, Pico and Yellow the loading times are almost the same for 1, 2 and 4 concurrent connections to the same resource. In the case of HTMLy, PuppyCMS and Singularity the smaller sample data surprisingly meant faster loading times for 4 concurrent connections compared to 1.

Raspberry Pi 2 Model B was slower than the test x86 system in most of the cases, however, Raspberry Pi 2 managed to outperform x86 when using PuppyCMS with larger sample files. The advantages of multiple cores could also be seen during *ApacheBench* tests where Raspberry Pi 2 outperformed the x86 system in a lot of cases as the number of concurrent requests grew.

Results by *ApacheBench* or *JMeter* did not match the loading times reported by the *Chrome* web browser. In the case of PuppyCMS and Singularity, results varied by orders of magnitude, so the author made his assumptions based on real-world results from the browser. *ApacheBench* results were only considered in the context of measuring relative performance with various numbers of concurrent connections.

The best-performing CMS turned out to be Singularity. It had the fastest loading times for both ARM-based test systems. Those never exceeded 3.1s even for the largest sample file. It managed to withstand all concurrency tests without crashing the systems and not going above 10s for page loading times according to *ApacheBench*. Although the results by *ApacheBench* cannot be considered trustworthy, they still suggest that Singularity can serve up to 64 users simultaneously.

The difference between the fastest CMS and the canonical performance of the same static files in the case of the Raspberry Pi 1 was 2-5 times, in the case of the Raspberry Pi 2, the difference was 2-10 times, depending on the sample files.

Results observed clearly show that most of the tested flat-file CMS-s are not suitable in conjunction with both Raspberry Pi models. Dynamic page loading and rendering from text files requires a lot of CPU resources which the current generation of these products can merely provide. CPU is the main system component impacting performance. With the given hardware, static web pages or static site generators should be preferred.

Performance of at least 1 flat-file CMS (Singularity) can be considered adequate. The author believes that this CMS-s can be used for small organisations where up to 64 people are accessing a given web resource simultaneously.

Results show that software used for web hosting is not optimised for ARM architecture. In nearly all cases, a single-core x86 system outperformed a quad-core ARM system with almost the same clock speed. The only cases where the Raspberry Pi 2 managed to outperform the test x86 system were situations with multiple concurrent connections, where 4 CPU cores of the Raspberry Pi 2 managed to split the workload to achieve better results.

A topic for further study could be the performance impact of using the network interface as a limiting factor. The same USB wireless adapter could be used to artificially limit the network performance to mimic the real world scenario where latency is an issue to see if overall system performance differences are distinguishable.

Another topic worth investigating would be to measure performance on more devices with a wider choice of system architectures. ARM is a fast-evolving system architecture and new devices are being manufactured constantly, to the point where it is impossible to get one’s hands on all of them.

Lastly, results achieved in this work could be compared to similar setups where source data is stored on a traditional storage device, e.g. spinning hard drive, solid-state drive, USB flash drive, external USB hard drive.

References

[1] Usage Statistics and Market Share of Content Management Systems for Websites. [WWW] http://w3techs.com/technologies/overview/content_management/all/ (28.03.2016) (web article)

[2] Daring Fireball. [WWW] http://daringfireball.net/projects/markdown/ (28.03.2016) (project website)

[3] Implementations. [WWW] https://github.com/markdown/markdown.github.com/wiki/Implementations (28.03.2016) (web article)

[4] Raspberry Pi. [WWW] https://en.wikipedia.org/wiki/Raspberry_Pi (28.03.2016) (web article)

[5] FAQs. [WWW] https://www.raspberrypi.org/help/faqs/#powerReqs (28.03.2016) (project website)

[6] Flat File CMS Systems. [WWW] https://github.com/ahadb/flat-file-cms (02.03.2016) (web article)

[7] Automad / Installation. [WWW] http://automad.org/documentation/installation (28.03.2016) (project website)

[8] Docs | Kirby [WWW] https://getkirby.com/docs (28.03.2016) (project website)

[9] GitHub - monstra-cms/monstra [WWW] https://github.com/monstra-cms/monstra (28.03.2016) (project website)

[10] How to Install Sphido CMS [WWW] https://www.sphido.org/docs/setup (28.03.2016) (project website)

[11] Ruby SSL issue => gem install not working [WWW] http://forum.tinycorelinux.net/index.php/topic,18901.0.html (01.05.2016) (internet forum thread)

[12] Bludit [WWW] http://www.bludit.com (01.05.2016) (project website)

[13] HTMLy [WWW] https://www.htmly.com/ (01.05.2016) (project website)

[14] Mecha [WWW] http://mecha-cms.com/ (01.05.2016) (project website)

[15] Phile [WWW] http://philecms.github.io/Phile (01.05.2016) (project website)

[16] Pico [WWW] http://pico.dev7studios.com/index.html (01.05.2016) (project website)

[17] PuppyCMS [WWW] http://puppycms.com (01.05.2016) (project website)

[18] Singularity [WWW] http://christophersu.net/2012/singularity-cms-single-php-file/ (01.05.2016) (project website)

[19] Yellow [WWW] http://datenstrom.se/yellow/ (01.05.2016) (project website)

[20] ARM architecture. [WWW] https://en.wikipedia.org/wiki/ARM_architecture (28.03.2016) (web article)

[21] HP Extends Benefits of ARM Architecture into the Datacenter with New Servers. [WWW] http://www8.hp.com/us/en/hp-news/press-release.html?id=1800094 (28.03.2016) (web article)

[22] How big is the ARM Server Market? [WWW] http://armservers.com/2014/05/09/how-big-is-the-arm-server-market/ (28.03.2016) (web article)

[23] Canonical's cloud-in-a-box: The Ubuntu Orange Box | ZDNet [WWW] http://www.zdnet.com/article/canonicals-cloud-in-a-box-the-ubuntu-orange-box (web article) (28.03.2016)

[24] Specifications [WWW] https://en.wikipedia.org/wiki/Raspberry_Pi#Specifications (28.03.2016) (web article)

[25] GNU General Public License, by Free Software Foundation, Inc. [WWW] http://www.gnu.org/licenses/gpl-3.0.txt (web resource) (30.04.2016)

[26] The Communist Manifesto, by Friedrich Engels and Karl Marx [WWW] http://www.gutenberg.org/ebooks/61.txt.utf-8 (web resource) (29.04.2016)

[27] Alice in Wonderland, by Lewis Carroll [WWW] http://www.gutenberg.org/ebooks/11.txt.utf-8 (web resource) (29.04.2016)

[28] The Jungle Book, by Rudyard Kipling [WWW] http://www.gutenberg.org/ebooks/236.txt.utf-8 (web resource) (29.04.2016)

[29] The Adventures of Sherlock Holmes, by Arthur Conan Doyle [WWW] http://www.gutenberg.org/ebooks/1661.txt.utf-8 (web resource) (29.04.2016)

[30] The Core Project File Architecture Diagram [WWW] http://www.tinycorelinux.net/arch_core.html (28.03.2016) (web article)

[31] php5.tcz.info [WWW] http://distro.ibiblio.org/tinycorelinux/7.x/x86/tcz/php5.tcz.info (web resource) (28.04.2016)

[32] Unix Top [WWW] http://www.unixtop.org/ (28.03.2016) (project website)

[33] free [WWW] http://linuxcommand.org/man_pages/free1.html (28.03.2016) (web article)

[34] IPTraf [WWW] http://iptraf.seul.org/ (28.03.2016) (project website)

[35] ApacheBench [WWW] https://httpd.apache.org/docs/current/programs/ab.html (01.05.2016) (project documentation)

[36] Apache JMeter [WWW] https://jmeter.apache.org/ (28.03.2016) (project website)

Appendix 1 – Performance results

Page loading time in *Chrome* (ms) and average loading times (time per request, mean; ms) at a set concurrency rate for a given file

Appendix 1. Table 1. Canonical performance results for the Raspberry Pi 1.

	Index.html	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	163.000	210.000	193.000	439.000	616.000	867.000	1100.000
1	3.921	3.958	9.999	20.079	29.946	47.128	85.815
2	7.827	7.920	20.016	40.665	60.856	96.807	175.942
4	15.424	15.633	39.662	82.195	123.520	197.228	359.028
8	30.752	31.056	78.806	165.906	249.893	400.891	730.528
16	62.074	61.933	163.295	334.113	510.459	819.624	1486.285
32	123.709	124.414	317.694	669.175	1034.863	1698.177	3097.963
64	246.342	248.009	654.190	1360.230	2101.639	3485.763	6429.196

Appendix 1. Table 2. Canonical performance results for the Raspberry Pi 2.

	Index.html	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	176.000	213.000	202.000	216.000	225.000	241.000	300.000
1	1.510	1.604	4.420	9.428	15.480	26.477	51.069
2	1.842	1.852	6.938	16.085	28.354	50.038	99.162
4	2.732	2.737	12.725	31.877	56.131	99.998	198.294

8	5.309	5.093	24.856	63.676	112.177	199.980	396.594
16	10.035	9.721	49.930	127.303	224.402	399.973	793.201
32	19.481	18.904	100.589	254.817	449.456	800.164	1587.852
64	37.874	36.540	201.269	514.790	906.332	1604.974	3174.284

Appendix 1. Table 3. Canonical performance results for the x86 system.

	Index.html	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	51.000	44.000	57.000	149.000	72.000	417.000	1040.000
1	1.408	1.435	4.576	9.920	16.609	28.593	55.447
2	2.603	2.327	7.454	18.309	31.332	55.699	109.358
4	5.288	4.704	14.670	35.803	61.810	110.338	217.148
8	10.478	9.366	30.900	74.883	129.842	230.568	434.566
16	20.907	19.471	58.957	140.599	247.001	439.248	869.087
32	41.362	37.928	116.912	285.250	493.385	877.909	1736.909
64	82.653	78.957	222.911	582.506	986.700	1756.338	3475.994

Appendix 1. Table 4. Bludit performance results for the Raspberry Pi 1.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	13490.000	14460.000	13120.000	14070.000	15400.000	20630.000
1	13049.802	13506.279	13131.593	13860.494	15322.296	18600.760

Appendix 1. Table 5. Bludit performance results for the Raspberry Pi 2.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	4290.000	4860.000	4520.000	4960.000	5640.000	8560.000
1	3157.210	3659.125	3361.354	3590.727	4848.978	5015.075
2	3193.946	3715.348	3415.967	3299.221	4437.912	4624.222
4	3314.266	3880.645	3561.541	3059.802	5138.314	5244.526
8	6723.180	7703.823	7457.018	7392.622	10171.276	10750.880

Appendix 1. Table 6. Bludit performance results for the x86 system.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	3150.000	3350.000	3620.000	3920.000	4230.000	6880.000
1	2794.164	3085.659	3002.012	3187.380	3512.433	4357.789
2	5774.391	6334.087	6078.844	6431.523	7079.384	8733.057
4	11499.210	12594.405	12071.331	12769.517	14053.689	17327.528

Appendix 1. Table 7. HTMLy performance results for the Raspberry Pi 1.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	1020.000	1160.000	1390.000	1670.000	-	-
1	1604.989	1684.909	1837.980	2033.974	-	-
2	1130.384	1133.446	1160.068	1182.432	-	-
4	2230.684	2283.334	2367.796	2407.297	-	-
8	4525.462	4595.171	4642.380	4750.278	-	-

16	9140.513	9166.981	9498.950	9459.302	-	-

Appendix 1. Table 8. HTMLy performance results for the Raspberry Pi 2.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	615.000	727.000	911.000	1050.000	-	-
1	696.320	749.023	813.129	927.380	-	-
2	167.312	171.896	184.016	197.743	-	-
4	204.908	205.695	223.913	233.667	-	-
8	392.040	419.672	402.561	431.335	-	-
16	764.194	754.108	840.705	879.516	-	-
32	1472.630	1550.656	1535.932	1627.803	-	-
64	2895.844	2930.817	3024.697	3186.829	-	-

Appendix 1. Table 9. HTMLy performance results for the x86 system.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	404.000	603.000	837.000	1220.000	-	-
1	845.923	928.500	1036.834	1199.214	-	-
2	45.408	50.305	63.829	75.407	-	-
4	91.820	80.490	93.571	119.346	-	-
8	118.409	135.511	175.757	233.310	-	-
16	238.887	270.823	350.782	465.322	-	-

32	517.318	617.738	774.503	934.164	-	-
64	1033.001	-	-	-	-	-

Appendix 1. Table 10. Mecha performance results for the Raspberry Pi 1.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	5070.000	5850.000	6930.000	8100.000	10680.000	-
1	4701.224	5411.641	6377.527	7643.466	10144.167	-
2	9478.346	10836.940	13159.107	15978.517	20367.633	-
4	18610.161	22171.864	22429.440	-	-	-

Appendix 1. Table 11. Mecha performance results for the Raspberry Pi 2.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	1740.000	2060.000	2770.000	3560.000	5110.000	-
1	1341.594	1758.286	2341.246	3128.568	4571.440	-
2	1404.859	1826.219	2391.985	3195.645	4650.884	-
4	1501.954	1929.005	2496.120	3311.143	4821.270	-
8	3235.962	3981.075	5105.142	6841.880	9743.696	-
16	6013.494	7930.762	10383.933	11050.843	17144.579	-
32	12136.629	11707.600	16793.717	24947.554	35419.833	-

Appendix 1. Table 12. Mecha performance results for the x86 system.

	1024	Gpl	Communist	Alice	Jungle	Sherlock

Chrome	1130.000	1380.000	1710.000	2290.000	3250.000	-
1	774.182	1016.962	1333.688	1826.653	2709.467	-
2	1632.324	2122.790	2773.339	3783.479	5513.278	-
4	3271.943	4265.582	5613.168	7618.242	11073.467	-
8	6617.060	8528.899	11229.778	-	-	-
16	13215.550	-	-	-	-	-

Appendix 1. Table 13. Phile performance results for the Raspberry Pi 1.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	1670.000	3100.000	4620.000	7640.000	10690.000	21170.000
1	1514.824	2911.819	4393.444	7353.330	10479.915	20921.675
2	3038.053	5936.822	8808.748	14849.847	-	-
4	6074.848	11749.802	27019.673	-	-	-
8	12327.940	-	-	-	-	-

Appendix 1. Table 14. Phile performance results for the Raspberry Pi 2.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	612.000	1390.000	2220.000	3620.000	5420.000	10620.000
1	449.519	1186.649	1980.243	3348.901	5203.104	10457.217
2	485.702	1232.769	2033.578	3400.639	5331.597	10494.704
4	567.185	1347.868	2124.304	3508.220	5397.806	10708.401

8	1301.638	2728.472	4541.597	7127.078	11072.914	-
16	2408.064	5314.569	8976.370	14271.506	-	-
32	5048.842	10793.538	-	-	-	-
64	9722.676	-	-	-	-	-

Appendix 1. Table 15. Phile performance results for the x86 system.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	318.000	815.000	1300.000	2190.000	3720.000	7370.000
1	235.458	689.848	1175.346	2067.588	3248.060	6525.354
2	469.672	1398.817	2387.945	4186.829	6526.008	13260.151
4	906.146	2760.938	4810.691	8369.440	-	-
8	1783.860	5552.630	9654.746	-	-	-
16	5208.138	13425.999	-	-	-	-
32	10351.199	-	-	-	-	-

Appendix 1. Table 16. Pico performance results for the Raspberry Pi 1.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	1510.000	2540.000	2210.000	3170.000	5070.000	8730.000
1	1281.235	2291.115	1888.973	2616.150	3928.329	7119.776
2	2529.461	4648.785	3763.954	5350.594	8522.850	-
4	5108.066	9199.444	7558.922	10932.874	-	-

8	10207.156	-	-	-	-	-

Appendix 1. Table 17. Pico performance results for the Raspberry Pi 2.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	612.000	1260.000	1010.000	1350.000	2230.000	3930.000
1	368.011	869.172	557.334	773.167	1189.936	2150.499
2	392.653	893.836	591.954	822.789	1225.434	2226.003
4	445.185	926.091	648.380	873.300	1320.878	2384.520
8	1185.205	2197.875	1316.731	1769.776	2753.892	4920.685
16	2037.653	3891.512	2648.160	3670.706	5443.460	9516.163
32	3878.204	7812.589	5450.163	7247.537	10776.438	-
64	7402.073	-	10967.790	-	-	-

Appendix 1. Table 18. Pico performance results for the x86 system.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	246.000	609.000	588.000	869.000	1640.000	3130.000
1	115.455	405.209	261.003	438.925	749.458	1473.864
2	226.944	810.414	518.176	892.661	1501.084	2978.355
4	421.738	1592.003	1020.431	1724.976	2946.427	5877.397
8	833.559	3191.139	2046.562	3434.066	5865.920	11796.916
16	1662.894	6380.652	4095.918	6888.757	13421.095	-

32	3400.850	13093.606	8505.353	16645.803	-	-
64	6881.644	-	-	-	-	-

Appendix 1. Table 19. PuppyCMS performance results for the Raspberry Pi 1.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	1030.000	1020.000	1090.000	1660.000	2110.000	8050.000
1	36.807	42.030	43.793	53.746	78.795	125.013
2	47.698	53.024	78.502	96.814	138.764	231.524
4	80.265	106.304	149.087	202.925	280.120	468.433
8	177.634	209.391	348.772	400.850	565.737	934.753
16	307.272	418.407	604.793	826.497	1155.967	1911.939
32	647.184	1023.812	1262.487	1627.969	2388.793	3948.129
64	1241.234	1743.603	2607.108	3538.699	4862.308	7971.302

Appendix 1. Table 20. PuppyCMS performance results for the Raspberry Pi 2.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	730.000	1040.000	1050.000	1600.000	1770.000	3610.000
1	20.294	21.182	23.320	30.497	40.698	63.018
2	12.770	15.428	25.492	37.373	59.672	112.273
4	15.172	24.621	43.316	60.772	107.761	210.889
8	29.831	39.828	77.502	136.894	219.011	441.440

16	57.164	85.292	160.008	261.062	438.947	827.639
32	102.687	148.257	309.774	500.402	855.373	1651.920
64	196.751	264.091	615.763	995.747	1694.769	3295.828

Appendix 1. Table 21. PuppyCMS performance results for the x86 system.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	565.000	1130.000	947.000	1570.000	2160.000	4220.000
1	3.473	6.487	12.746	21.142	35.900	69.383
2	6.361	11.671	25.311	41.111	71.385	137.416
4	13.009	23.732	48.244	79.067	139.055	273.002
8	25.826	46.074	94.562	157.069	278.336	546.323
16	51.586	93.359	187.461	314.379	558.647	1088.360
32	103.431	180.826	373.889	615.846	1066.274	2151.889
64	204.547	368.292	757.458	1200.917	2347.061	4176.617

Appendix 1. Table 22. Singularity performance results for the Raspberry Pi 1.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	481.000	980.000	1000.000	1370.000	1860.000	2870.000
1	28.597	32.845	37.409	43.421	62.161	105.140
2	29.686	42.487	66.230	93.747	129.378	228.184
4	59.173	81.642	119.610	162.218	248.468	444.299

8	81.584	141.996	232.830	329.155	509.810	876.074
16	181.359	283.672	464.693	663.260	1029.899	1801.815
32	323.657	575.850	1096.926	1401.449	2123.480	3717.595
64	795.751	1195.784	2044.764	2855.282	4447.001	7718.801

Appendix 1. Table 23. Singularity performance results for the Raspberry Pi 2.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	451.000	946.000	949.000	1400.000	1880.000	3030.000
1	16.474	14.987	20.892	25.133	36.976	64.583
2	11.146	13.066	22.473	33.507	54.071	107.374
4	9.604	19.432	36.571	60.240	105.601	210.047
8	18.613	36.361	73.464	135.939	211.302	412.920
16	34.295	69.882	150.595	240.695	422.084	819.244
32	67.663	134.579	291.911	491.297	833.462	1641.646
64	121.463	234.207	564.135	937.661	1671.033	3312.504

Appendix 1. Table 24. Singularity performance results for the x86 system.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	562.000	904.000	945.000	1200.000	1850.000	3040.000
1	9.870	13.044	19.223	26.733	42.500	75.655
2	19.599	25.115	37.943	52.919	84.646	140.001

4	10.281	21.073	46.204	76.953	139.445	270.240
8	20.438	42.697	91.414	147.323	273.517	540.250
16	39.583	81.128	180.380	296.222	552.624	1084.044
32	79.951	157.712	360.280	569.932	1060.307	2142.142
64	162.222	318.781	723.385	1129.872	2176.017	4274.241

Appendix 1. Table 25. Yellow performance results for the Raspberry Pi 1.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	2510.000	4020.000	5850.000	8880.000	12470.000	24300.000
1	920.077	2385.459	4040.168	7209.563	10588.551	21879.445
2	1851.177	4806.730	8120.502	14550.867	-	-
4	3634.216	9688.519	-	-	-	-
8	7258.081	-	-	-	-	-

Appendix 1. Table 26. Yellow performance results for the Raspberry Pi 2.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	980.000	2030.000	2920.000	4530.000	6600.000	12730.000
1	263.940	1049.580	1908.517	3354.701	5314.745	10902.640
2	280.240	1065.788	1928.760	3367.860	5343.446	10991.489
4	318.803	1090.047	1986.378	3440.277	5397.017	11104.678
8	662.709	2208.256	3924.617	6933.566	10908.432	-

16	1308.930	4415.013	7993.736	13908.514	-	-
32	2433.471	8762.435	-	-	-	-
64	4704.002	-	-	-	-	-

Appendix 1. Table 27. Yellow performance results for the x86 system.

	1024	Gpl	Communist	Alice	Jungle	Sherlock
Chrome	312.000	950.000	1500.000	2540.000	4030.000	8630.000
1	94.204	585.111	1126.839	2084.608	3332.640	6899.007
2	169.693	1164.009	2259.424	4205.981	6706.179	13853.792
4	341.535	2350.066	4565.429	8463.421	-	-
8	684.352	4723.680	9156.676	-	-	-
16	1363.500	9458.661	-	-	-	-
32	2771.133	-	-	-	-	-
64	5678.244	-	-	-	-	-