Performance and Energy Evaluation of RESTful Web Services in Raspberry Pi

Luiz H. Nunes *, Luis H. V. Nakamura *, Heitor de F. Vieira *,
Rafael M. de O. Libardi *, Edvard M. de Oliveira *, Julio C. Estrella *, Stephan Reiff-Marganiec †
* *University of São Paulo (USP)*
Institute of Mathematics and Computer Science (ICMC), São Carlos-SP, Brazil
Email: {lhnunes, nakamura, heitorfv, mira, edvard, jcezar}@icmc.usp.br
† *University of Leicester*
University Road, Leicester, LE1 7RH - UK
Email: srm13@le.ac.uk

***Abstract*—Green computing has emerged as a hot topic leading to a need to understand energy consumption of computations. This need also extends to devices with limited resources as are common in the internet of things. RESTful services have shown their potential on such devices, but there are many choices of frameworks for their development and execution. Current research has analysed performance of the frameworks but no attention has been given to systematically studying their power consumption. In this paper we analyse the execution behaviour and power consumption of web services on devices with limited resources and make initial observations that should influence future development of web service frameworks. Specifically, we conduct experiments comparing web services in the Axis2 and CXF frameworks analysing the respective performance and power consumption. Bringing together the best features of small devices and SoC, it is possible to provide diverse, mobile and green applications – however careful selection of development environments can make significant differences in performance and energy consumption.**

***Keywords*-Web Services, Service Oriented Architecture, Quality of Service, Performance Evaluation, Apache Axis2, Apache CFX, Raspberry Pi**

I. Introduction

The combination of embedded devices, sensors and the Internet allows to link the physical world with the cyber space, expanding the Internet to the Internet of Things (IoT) [9]. Service computing plays a crucial role in this world as it provides the basic development paradigm used to integrate the various parts of this global computing development. The increasing use of mobile and embedded devices jointly with the expansion of IoT requires studies about which tools, protocols, frameworks, and applications should be used in order to increase efficiency of the systems while reducing costs. Some efforts have been made in this area, usually comparing performance and features of the different frameworks. However, a specific challenge arises from the desire to perform efficient and fast computation and communication while simultaneously reducing energy consumption. This challenge increases when considering that the IoT is commonly using small, embedded devices which are energy efficient but at a great expense to performance as the resources (such as CPU power or memory) are very limited [5]. A question arises as to whether such devices can execute web services (either as client or server) efficiently and what trade-offs exist between power consumption and performance. Furthermore, there is a question whether – considering previously identified performance differences between development frameworks – different frameworks show different energy consumption behaviours.

Due to service-oriented applications generally using different kinds of resources and platforms to perform tasks, they can overcome resource limitations and improve the overall application performance. For example, a project presented in [7] used a Raspberry Pi to monitor heart rate and body temperature, moving processing to other parts of the system. However, the use of service-oriented applications in embedded devices still need to be investigated with respect to other types of information, such as larger messages that demand more resources. This study is very relevant because services use large amounts of data to create additional information in the message body, which increases the size and processing time of those messages [11].

The service-oriented applications are implemented as web services that can be classified into two categories: RESTful and SOAP. REST (*REpresentational State Transfer*) is an architectural style where the resources are identified by an URI (*Uniform Resource Identifier*) and manipulated using HTTP methods. SOAP *Simple Object Access Protocol* services are described by a Web Services Description Language (WSDL) document and manipulated by XML messages. Finally, both RESTful and SOAP services enable the exchange of information through XML messages [16]. In this work we only considered RESTful applications, since they are more suited for embedded applications [4].

In this paper, we present a performance and energy consumption study evaluating the behaviour of RESTful web services using the Raspberry Pi. To perform this evaluation, services with the same features were developed using the Axis2 and CXF frameworks developed by the Apache Software Foundation. Times to marshal/unmarshal messages with different sizes were analysed in tests in real devices with normal CPU clock and overclocking configurations. The results of this evaluation contribute to the development of web services in several ways: 1) We can determine the performance difference between two RESTful WS frameworks exchanging messages with different loads. 2) We can determine whether the energy consumption would be affected due different implementations, configurations and workloads. While understanding energy consumption in its own right is important, the investigation about the relation (performance to energy consumption) is also crucial as saving energy will only be justified if the performance remains acceptable. We further show that the Raspberry Pi can be useful in service-oriented applications, that are able to perform different types of tasks. Bringing together the best features of small devices and SoC, it is possible to provide diverse, mobile and green applications.

The results from this systematic study should lead to further investigations for more complex scenarios and to better analysis of features of development frameworks – with a hope of eventually gaining frameworks that are easy to use and are energy efficient.

This paper is organized as follows: Section II presents a literature review of embedded web services performance evaluation studies and methodologies. Section III highlights the characteristics of hardware, frameworks and technologies used in this study. Section IV describes the methodology and configurations used for the experiments. The results are then discussed in Section V. Finally, the conclusions and directions for future work are presented in Section VI.

II. Related Work

The web services performance evaluation methodologies for embedded and mobile environments were categorized into simulation based on mathematical models and real device experiments.

Real experiments were found in [17], [19], [14] and [11] and are the most used. In real experiments, real devices and prototypes are used to measure response variables in a real environment using replications for non-deterministic variables. Although this is the most used approach, it is more expensive, needs to follow a methodology and requires statistical analysis.

Mathematical models, used in [21] and [12] and simulation tools to evaluate proposed architectures. In these approaches, mathematical models along with models to evaluate proposed architectures. This approach is cheaper than real experiments and thus useful when it is too expensive to evaluate using a prototype. However, there is a risk that the model does not reflect the environment.

We used real device experiments using a testbed environment for this study. The work of [19] reviewed some web services approaches for embedded environments and concluded as we do that they are suitable for mobile devices. Although it was an extensive study, it crucially lacks RESTful web services experiments.

Mizouni [17] compared SOAP and RESTful for mobile devices, it evaluated the processing time, battery consumption and the throughput (req/s). It proposed an evaluation architecture and a scenario with phone-calls. The results from this works showed that RESTful was better than SOAP, the only exception was with the battery consumption using an image service provider.

Hamad [12] study also compared both technologies regarding the message size and the processing time. It used the Sun Mobile Emulator with two benchmarks (float manipulation and string manipulation). Although they used only a simulated environment, their results showed that RESTful is better than SOAP for the emulator. [6] evaluated the bandwidth, memory usage and response time. Their results also concluded that RESTful is better than SOAP for this response variables in a mobile environment.

Also, other works [4], [13] concluded that RESTful services are more suited for mobile devices, because they generate less overhead in data transfers and spend less time to pack/unpack the request and response messages. However, performance evaluation studies are superficial. In [8] the usage of RESTful web services in embedded devices is also discussed, concluding that they save a lot of resources and energy because they use small messages by removing their headers. But the work lacks a performance evaluation and replication tests to support this conclusion.

As we know from the literature, RESTful are better than SOAP for mobile devices, but there are different ways to implement and deploy a RESTful web service. Thus, this paper extends our preliminary work [18] and aims to complement the work presented in [19] showing that RESTful messages can be handled in different ways and comparing some of them. Also, this paper differs from the existing studies in at least three contributions. 1) We use small, low power devices, 2) present an energy consumption evaluation approach, and 3) demonstrate a performance evaluation analysis of RESTful applications using two different frameworks in embedded devices.

III. Tools and Technologies

1) Hardware: Raspberry PI and Arduino: The Raspberry Pi model B is a computer based on the SoC Broadcom BCM2835 (which has a 700 MHz ARM1176JZF-S processor integrated with a VideoCore IV GPU and 512 MB of RAM) with an SD card slot for data storage. The Raspberry Pi used in our experiments runs a customized version of Debian GNU/Linux called Raspbian (http://www.raspbian.org/)

Arduino is an open-source electronics prototyping platform used by developers and engineers to create personal and commercial projects. A major use of the Arduino is to control electronics like lights, motors, and other components.

Arduino can also sense environments by receiving input information from sensors; hardware extensions (known as *Shields*) can be attached to provide more features such as Ethernet and WiFi access [2]. The board used for the experiments in this paper was an Arduino Nano 3.0, based on the ATmega328 micro controller running at 16MHz with 1Kb SRAM, 32KB Flash memory and Mini-B USB interface.

2) RESTful Web Services and Development Frameworks: The REST style promises to emphasize the scalability of component interactions, generalization of interfaces and can encapsulate legacy systems [10]. According to [20], choosing REST removes the need to make a number of architectural decisions introducing a simple and lightweight alternative to services development based on a set of operations already defined in the HTTP protocol. However, extending RESTful web services to support advanced functionality, such as security, is complex.

Apache Axis2 is a Java project that aims to make it easier to develop web services for client and server side through a object-oriented approach and a modular architecture. Axis2 uses the Apache Axiom library to build their messages because it provides a set of XML methods that allows to organize the objects into a tree [3]. Axis2 uses two pipes (or flows) to send and receive messages, called **Inflow** and **Outflow**. The combination of these two flows enables the message exchange (MEPs) between clients and providers [15].

Apache CXF is another Java project that enables the SOAP and RESTful web services development. The Java API for RESTful Services (JAX-RS) is one of the several patterns by CXF framework to develop web services and abstracting implementation details from clients that can exchange data in different formats such as JSON and XML. The message flows in CXF are handled by interceptors that perform a particular functionality. They can combined into chains, grouped and arranged in stages [1].

IV. Evaluation of Power Consumption

The methodology used for the performance and energy evaluations is explained in this section. First of all, it is important to understand how the experiments were conducted. Figure 1 illustrates the messages flows between the applications (steps are marked 0 to 6).

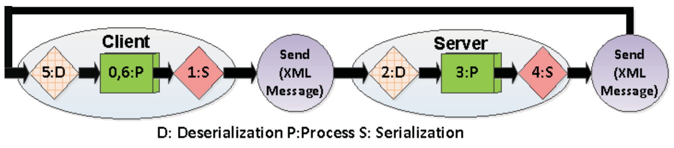

Figure 1: Messages flow in the experiments [18].

The application is started by the client device and generates a sequence of random numbers which is serialized into an XML message. This message is send to the server device where it will be deserialized and processed by a sort method. As soon as this process is concluded, the sorted numbers are serialized in XML format and sent back to the client device. Finally, the client deserialize the message received from the server and ends its execution.

To monitor the energy consumption, we used Arduinos Nanos because they are capable to ensure accuracy and synchronism in data collection through an automated code to start and finish the measure – which can be linked to the start and finish times of events in the Raspberry Pi. It is possible to keep electric current within 800mA by using a 1Ω resistor in series with the Raspberry Pi (Figure 2) – this allows to measure the voltage drop in the resistor to infer the electric current of the circuit precisely.

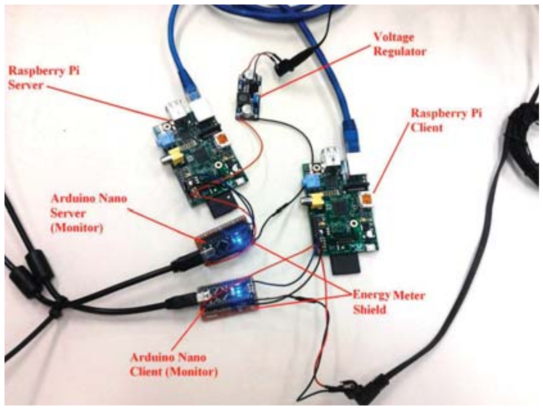

Figure 2: Experimental setup.

The Arduino was used to collect data like time, voltage and current 170 times per second from both client and server devices through a shield that was made to measure the energy. The data is stored in a desktop computer using the Mini-B USB interface. Joule (J) is a unit commonly used to measure mechanical energy (work) or thermal energy (heat). In the International System of Units (SI), all work or energy are measured in Joules. The energy consumption, in Joules, is calculated from the voltage and current at time t ($V_{(t)}$ and $I_{(t)}$) as:

$$Joules = \int_0^x V_{(t)} I_{(t)} \, \mathrm{d}t.$$

A. Experimental Setup

Two Raspberry PI devices model B [1] were used with the settings modified for each experiment. The first configuration uses the standard CPU clock (700Mhz) and the second one uses an overclocked configuration (950MHz). Moreover, the maximum amount of memory configured in the JVM (Java Virtual Machine) was set to 128Mb and the remainder was used by the system.

[1]ARMv6-compatible processor rev 7 (v6l), 697.95 BogoMIPS, Hardware Rev BCM2708, 512MB RAM, 8GB SD Card running Raspbian (GNU/Linux) and JDK 1.6, Apache Tomcat 6.0, Axis2 1.6.2, CXF 2.7.4, Jetty 7.0).

The experiments were designed to gather as much information as possible to compare the performance and energy differences between the *Devices CPU Frequency Configurations* (700Mhz and 950Mhz) using two different development *Frameworks* for web services (Axis2 and CXF). Eight experiments involving combinations of factors (Device CPU Frequency, Framework and Message Size) with different configurations or levels were designed (Table I).

Table I: Design of experiments

Exp	Device CPU Frequency	Framework	Message Size
1	700MHz	Axis2	100Kb
2	700MHz	Axis2	500Kb
3	700MHz	CXF	100Kb
4	700MHz	CXF	500Kb
5	950MHz	Axis2	100Kb
6	950MHz	Axis2	500Kb
7	950MHz	CXF	100Kb
8	950MHz	CXF	500Kb

The web services developed and their respective frameworks were instrumented to collect the time of serialisation and de-serialisation in both client and service. The total time of both were also recorded. Each experiment was repeated 50 times in order to calculate the average time and guarantee a statistically correct result. Besides, the standard deviation and confidence intervals (assuming a 95% confidence level) were also calculated for each of the average times collected. The results of the experiments are presented and discussed in the next section.

It should be noted that further configurations (other frequencies, different memory configurations, more frameworks etc.) could be explored. However in this study we wanted to explore whether there are any significant differences that merit such further studies.

V. Results

This section presents the results of experiments using bar graphs. Subsections divide these results by server and client devices, where it is possible to observe the behaviour of both frameworks considering different sizes of messages. In the last subsection a general analysis of the results is made. Note that the frameworks provide different support for developing server and client applications (e.g, some automatic conversion from REST to SOAP is conducted by Axis on the server side), so we consider the two roles separately in detail.

A. Server Results

1) ***Server serialisation:*** Figure 3 shows the time in milliseconds for the message serialisation on the server, where both Axis2 and CXF frameworks had an increased time for serializing 500Kb messages compared to 100Kb messages.

This behaviour occurs because both frameworks use a "flow" of methods to process the data and perform the message serialisation, making its time proportional to the amount of serialised data. The Axis2 framework was faster than CXF to process both messages. This difference occurs because during the message de-serialisation all information on how to serialise that message is obtained by Axis2. On the other hand, the JAX-RS library is less optimized, because it uses methods of generic classes to serialise the message. Figure 4 shows energy consumption during the server message serialisation. The energy results reflect the same behaviour as performance results, because the longer the processing time is, the greater the energy consumed will be. An interesting observation was noticed when the Raspberry Pi is working with 950Mhz (Overclock). It consumes less energy than when working with 700Mhz. This can be explained, because when it is using 950Mhz the processing time is shorter. The total consumption is lower as the effect of the shorter processing time outweighs the extra energy used. This behaviour is discussed further in section V-C.

2) ***Server de-serialisation:*** Figure 5 presents the de-serialisation message results from the server device. It is noted that due to the use of libraries (JAX-RS), the CXF framework had shorter de-serialisation times for messages of 100Kb and 500Kb than Axis2. This behaviour occurs be-

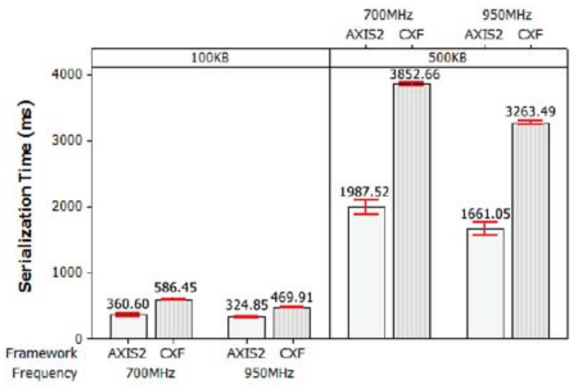

Figure 3: Server serialisation time.

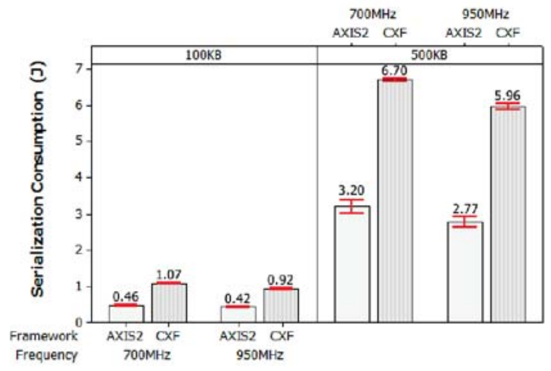

Figure 4: Server serialisation energy consumption.

cause before starting the de-serialisation, the Axis2 libraries need to read and parse a WSDL file, which contains the format of input/output messages and services descriptions.

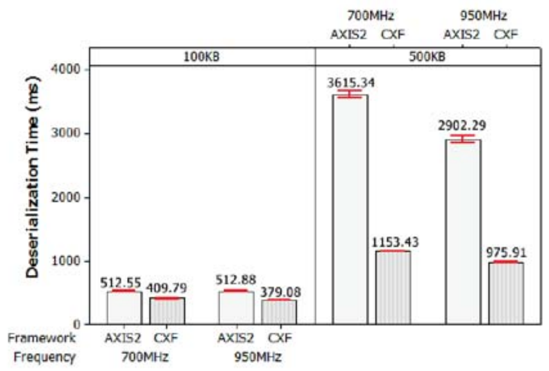

Figure 5: Server de-serialisation time.

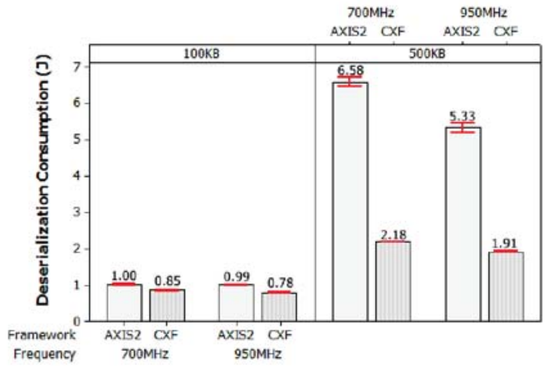

Figure 6: Server de-serialisation energy consumption.

Figure 6 shows the energy consumption during the server message de-serialisation. Again, the energy results are proportional to the performance results, with CXF performing better in this task.

*3) **Server Total**:* Figure 7, shows the total time in seconds to execute the service on the server device. As expected, both frameworks had increased service execution time when the size of the message is increased, since the processing time is directly proportional to the amount of data. It is possible to notice that the CXF framework requires less time than the Axis2 framework to execute services with 500Kb messages. This reduction is explained by the use of the JAX-RS library, which handles RESTful messages directly, whereas Axis2 has an overhead to convert REST to SOAP messages during service execution. CXF also was faster for messages of 100Kb when the server device was configured to 950Mhz.

Figure 8 shows the energy consumption during the server execution (serialisation + Service Processing + de-serialisation). In all results the CXF saves more energy than Axis2, except in 100Kb messages and 700Mhz of

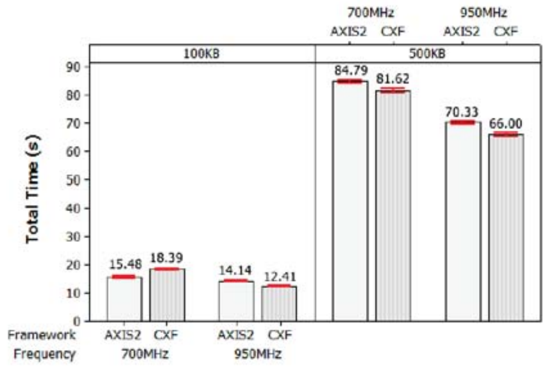

Figure 7: Server total execution time.

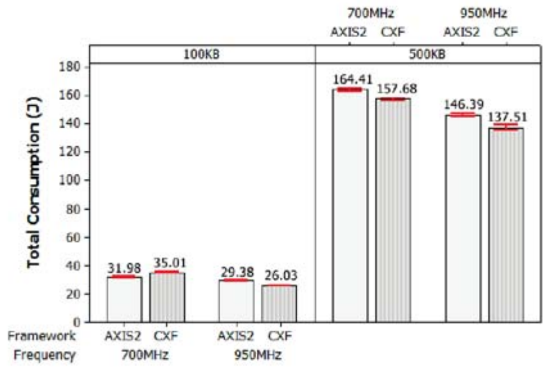

Figure 8: Server total power consumption.

CPU clock. Thus, in general CXF framework presents better performance and energy results than Axis2 for those experiments executed on the server side.

B. Client Results

*1) **Client serialisation**:* Figure 9 presents the message serialisation time in the client device in which the Axis2 framework shows a large difference in regard to the time while serializing 100Kb and 500Kb CXF messages. This difference is due to the implementation of the serialisation step in the Axis2 framework, two points of which should be highlighted:

Firstly, in the Axis2 client a "fake" message serialisation is done directly by the developer through the AXIOM library, where no pre-processing of data is needed, constructing the message to be sent directly. When these messages come to the transport level, they are converted into a SOAP message before being sent to the server provider. Secondly, the uniform times in Axis2 shown in Figure 9 occur because the message only has one field for the payload, which is already on the client device memory. This is the procedure adopted by the developer to send REST messages using the Axis2 framework.

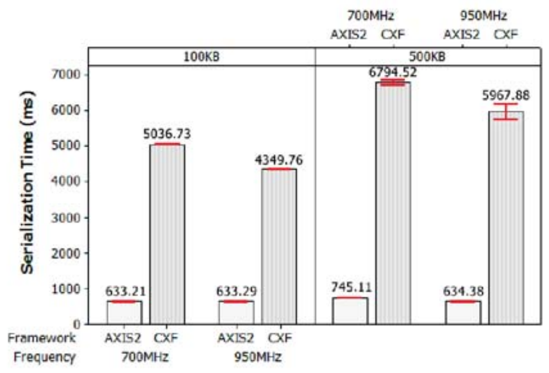

Figure 9: Client serialisation time.

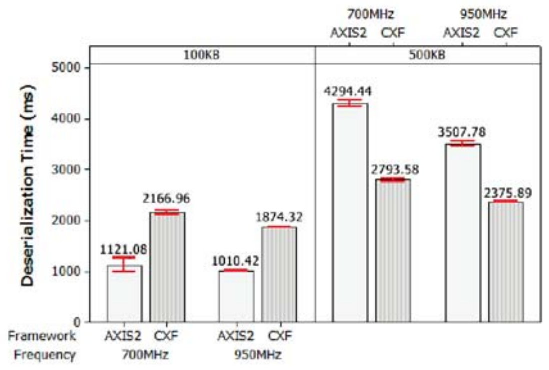

Figure 11: Client de-serialisation time.

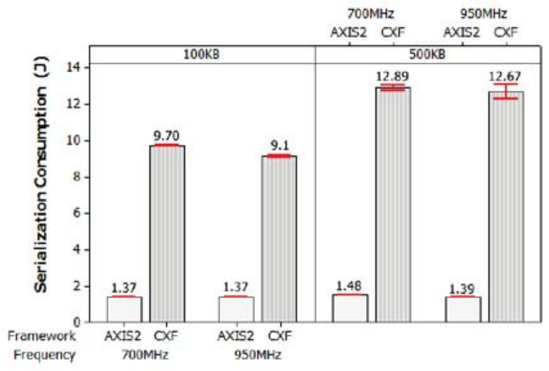

Figure 10: Client serialisation energy consumption.

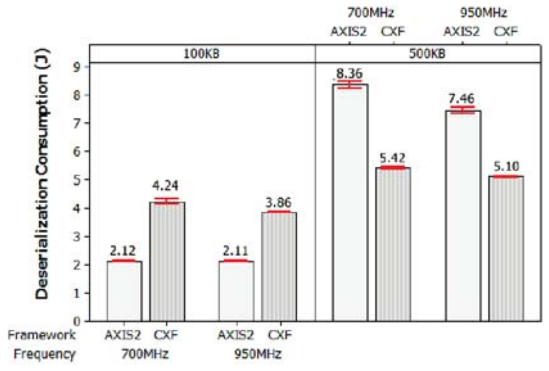

Figure 12: Client de-serialisation energy consumption.

Figure 10 shows the energy results obtained from the client serialisation experiments. As the client serialisation time in Axis2 was shorter than in CXF, Axis2 had lower energy consumption. Using the CXF, the results related to energy with 500Kb messages are statistically equal using 700MHz and 950Mhz.

*2) **Client De-serialisation**:* Figure 11 presents the results of the de-serialisation time in the client device. The CXF framework has shorter times to deserialise 500Kb messages. This difference is due to the de-serialisation performed through CXF framework libraries that are optimized to deserialise the messages efficiently as we saw with server de-serialisation.

The Axis2 framework was more efficient to deserialise 100Kb than 500Kb messages, once they showed a considerable increase in the de-serialisation time. This increase is due to Axis2 de-serialisation being executed sequentially by the Axiom libraries, where the information is extracted from the object tree received in the message. Thus, the larger the size of the information contained in the message, the longer the time to extract the information required for their de-serialisation. Therefore, unlike the serialisation time, the de-serialisation time in Axis2 framework was faster to 100Kb messages, once this de-serialisation is implemented sequentially by the developer. Otherwise, the CXF framework was more efficient to deserialise messages of 500Kb, once its de-serialisation is performed through the JAX-RS library, which handles messages with generics classes before starting the extraction of information, and consequently improving the de-serialisation of larger messages. Figure 12 presents the client energy consumption results in the de-serialisation step. When considering 500Kb messages this process is faster with CXF and consequently the average energy consumption is lower. But, this de-serialisation process for 100Kb messages saves energy when using Axis2 framework.

*3) **Client Total**:* The Figure 13, shows the total time for the client application execution. This time includes the client serialisation and de-serialisation times, the network traffic, and the service execution time on the server side (Server Total Time). As expected, both frameworks have increased the client execution time according to the increasing size of the messages. This occurs because the service execution time is proportional to the amount of data to be processed and this time has great influence on the final client time.

Yet we can verify that Axis2 framework had a better performance for 100Kb messages while CXF framework had

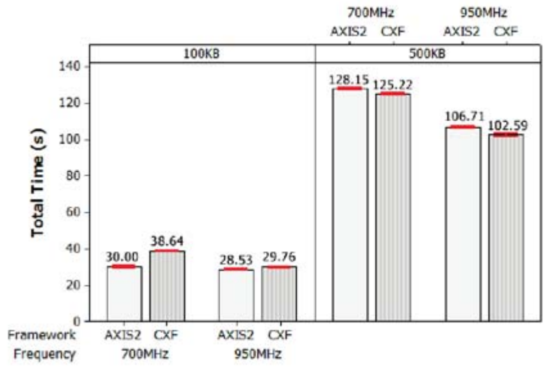

Figure 13: Client total execution time.

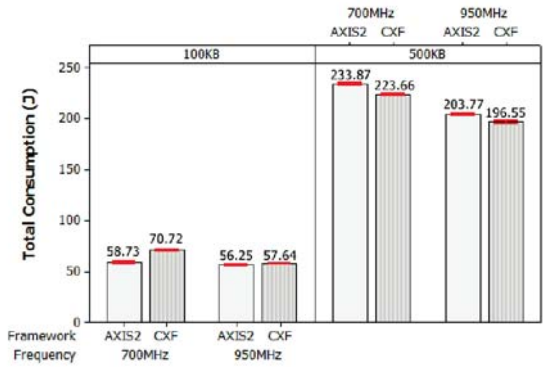

Figure 14: Client total energy consumption.

a better performance for 500Kb messages. JAX-RS libraries use methods of generic classes to handle RESTful messages which has better performance for larger messages. On the other hand, Axis2 framework handles REST messages as SOAP messages in its core which causes an overhead proportional to the quantity of converted data. In Figure 14, the energy consumption results are presented. They represent the total energy consumption in the client side including the time to create the message, serialise it, sending it through the network, waiting for a server response, deserialises the message, and having it as a Java object. Summarizing, the results show that CXF framework saves energy with 500Kb messages while Axis2 saves energy with 100Kb messages. Moreover, the overclocking configuration had influence in the time results and consequently in the energy consumption.

C. Results Analysis

A general analysis of the results obtained from the client and servers devices indicate that the Axis2 framework is more effective for messages with small amounts of data (100Kb) while the CXF framework is more suitable for large amounts of data (500Kb). However, the differences in the execution time among the frameworks are small. Table II shows the percentages occupied by the serialisation time, de-serialisation and execution time in the service on the client device. It can be seen that for both frequencies the serialisation and de-serialisation times practically have no influence on the total time. It is also noticeable that the average time of client applications is primarily influenced by the service time followed by other times that include the manipulation of data and network times.

Table II: Average client times (%)

Client Times				
Frequency	**700MHz**			
Framework	**Axis2**		**CXF**	
Message Size	**100Kb**	**500Kb**	**100Kb**	**500Kb**
Serialization Time(%)	0,002	0,001	0,013	0,005
Deserialization Time(%)	0,004	0,003	0,006	0,002
Service Times(%)	62,946	73,495	49,713	66,545
Others(%)	37,048	26,501	50,269	33,447
Total Time (s)	30,00	124,91	38,64	125,22
Frequency	**950MHz**			
Framework	**Axis2**		**CXF**	
Message Size(%)	**100Kb**	**500Kb**	**100Kb**	**500Kb**
Serialization Time(%)	0,002	0,001	0,015	0,006
Deserialization Time(%)	0,004	0,003	0,006	0,002
Service Times(%)	61,190	71,324	44,292	65,607
Others(%)	38,804	28,672	55,687	34,385
Total Time (s)	28,54	106,71	29,77	102,59

As this table shows, the Axis2 framework spends more time to receive the response message, as the server side handles it with a SOAP message. The CXF framework on the other hand spends more time doing others tasks like preparing and recovering content with generics classes. Furthermore, the frequency increase reduces the total time needed to conclude the operation.

Table III: Average client energy consumption (%)

Client Energy Consumption				
Frequency	**700MHz**			
Framework	**Axis2**		**CXF**	
Message Size	**100Kb**	**500Kb**	**100Kb**	**500Kb**
Serialization(%)	2,348	13,717	0,634	5,767
Deserialization (%)	3,614	6,001	3,579	2,424
Service (%)	59,734	47,924	73,987	64,846
Others (%)	34,304	32,359	21,800	26,963
Total (J)	58,73	233,88	70,73	223,67
Frequency	**950MHz**			
Framework	**Axis2**		**CXF**	
Message Size	**100Kb**	**500Kb**	**100Kb**	**500Kb**
Serialization (%)	2,449	15,795	0,684	6,450
Deserialization (%)	3,764	6,701	3,662	2,599
Service (%)	58,534	41,024	68,746	66,349
Others(%)	35,253	36,480	26,908	24,601
Total Consumption(J)	56,26	203,78	57,64	196,55

Table III shows the percentages occupied by serialisation, de-serialisation and execution energy consumption on client device. In this table, it can be seen that for both frequencies the major factors are the same as in Table II. However, the serialisation and de-serialisation consumption have higher influence on energy consumption than on performance, because more processing is needed in this phase which raises the energy consumption.

Table IV shows time percentages for serialisation, de-serialisation and execution of service on server device. In

Table IV: Average server times (%)

Server Times				
Frequency	700MHz			
Framework	Axis2		CXF	
Message Size	100Kb	500Kb	100Kb	500Kb
Serialization Time(%)	2,202	1,871	3,025	4,447
Deserialization Time(%)	3,134	2,986	2,114	1,287
Service Times(%)	94,664	95,143	94,861	94,265
Total Time (s)	16,36	93,13	19,39	86,63
Frequency	950MHz			
Framework	Axis2		CXF	
Message Size	100Kb	500Kb	100Kb	500Kb
Serialization Time(%)	2,168	2,218	3,543	4,646
Deserialization Time(%)	3,423	3,875	2,858	1,389
Service Times(%)	94,408	93,908	93,598	93,965
Total Time (s)	14,98	74,90	13,26	70,25

this table, the greatest impact factor is the service execution time. because a CPU bound service is executed and takes the most part (around 93%) of the total time due to high processing demands. It is important to emphasize that for Axis2 framework the execution time includes the times to handle a SOAP message which raise the execution time compared to CXF framework. Serialisation and deserialisation times have little influence in this scenario, as the time to perform these steps is less than 7% in all cases. Again, higher frequencies reduces the total time of service execution.

Table V: Average server energy consumption (%)

Server Energy Consumption				
Frequency	700MHz			
Framework	Axis2		CXF	
Message Size	100Kb	500Kb	100Kb	500Kb
Serialization (%)	1,453	0,257	3,060	4,253
Deserialization (%)	3,140	4,007	2,441	1,387
Service phases(%)	95,407	95,735	94,499	94,360
Total (J)	31,98	164,42	35,01	157,69
Frequency	950MHz			
Framework	Axis2		CXF	
Message Size	100Kb	500Kb	100Kb	500Kb
Serialization (%)	10,909	1,897	3,571	4,336
Deserialization (%)	3,403	3,641	3,016	1,513
Service (%)	85,688	94,462	93,413	94,151
Total (J)	29,39	146,39	26,04	137,51

Table V shows the percentages of serialisation, deserialisation and execution consumption in the server device. For both frequencies the major factors are the same as in Table IV. But the serialisation and de-serialisation consumption has again higher influence on energy need than on performance because more processing is necessary which raises the energy consumption. Again higher frequencies reduce the total energy consumption needed to conclude the operation. It is important to highlight that higher frequencies

Table VI: Client power consumption

Client Power Consumption (Watts)				
Frequency	700MHz		950MHz	
Message Size / Framework	100Kb	500Kb	100Kb	500Kb
Axis2	1,96	1,87	1,97	1,91
CXF	1,83	1,79	1,94	1,92

Table VII: Server power consumption

Server Power Consumption (Watts)				
Frequency	700MHz		950MHz	
Message Size / Framework	100Kb	500Kb	100Kb	500Kb
Axis2	1,96	1,77	1,96	1,95
CXF	1,81	1,82	1,96	1,96

reduce total energy consumption. However, the relation between energy consumption and total time or power, as shown in Tables VI and VII, is proportional. Thus, the reduction in energy consumption related with higher frequencies is due to the shorter processing times caused by higher power levels. In a controlled environment (air conditioning at 23^{0}C), we reached better results using 950Mhz for both performance and energy consumption. Obviously, overclocking might not be possible in real environments, unless loads are such that the device has long rest periods to cool down in between operations or good cooling can be provided. Note though that the Raspberry Pi working as a server increases the delay to the client application.

VI. Conclusion

A wide variety of projects explore the resources of mobile and embedded devices which, although their hardware is limited, are able to perform personal and commercial tasks. In this paper, the performance and energy results showed that using Raspberry Pi resources for the execution of tasks that depend mostly on the processor is not effective. However, the results showed that these devices can be used to serialise and de-serialise REST messages in a timely manner. Thus, with the expansion of IoT that usually does not require massive computational power in 'things', such devices are suitable to accomplish the most general tasks on both client and server sides.

Regarding the frameworks, both bring different benefits to the developer. However, the CXF framework was more appropriate in this environment, as it provides support for the creation of both service providers and client applications. Furthermore, when using CXF in service provider applications it achieves shorter overall times for large messages (500Kb) while Axis2 is better for small messages (100Kb). In terms of development experience, the small time and energy consumption difference between the frameworks does not justify the use of the Axis2 framework, as it is more complicated to develop RESTful clients in Axis2 and performance advantages quickly disappear when message sizes increase. What can be learned is that overall the different frameworks have little impact on energy consumption, however they can behave very differently for different tasks. So, further development of web services frameworks for mobile devices should consider how certain aspects can be implemenetd as to better balance performance and energy consumption.

The overclocking configuration also has influence in time and energy consumption results. Particularly results are achieved faster when the Raspberry Pi is overclocking (as one would expect), but this can surprisingly also cause less energy to be consumed. This is especially true for large messages, but it is important to remember that the overclocking was done in a controlled environment and general suitability in practice is not yet clear.

The Raspberry PI has impressed because of its easy installation, configuration and use of resources to provide and access web services. It can be a good low-cost solution for applications that require little processing capacity, such as a monitoring system. In future work we intend to monitor the energy consumption using batteries and evaluate other kinds of applications (IOBound and MemoryBound), devices (Parallela, BeagleBone, etc.), frameworks (Jersey, Restlet, etc.) and protocols (Constrained Application Protocol (CoAP)). We also plan to analyse features of the frameworks in more detail to identify where optimisations can be made to reduce power consumption.

Acknowledgment

We thank Coordination of Improvement of Personal Higher Education (CAPES) and São Paulo Research Fundation (FAPESP, processes 2011/09524-7, 2013/26420-6, 2011/12670-5), for the support of this research. We also thank ICMC-USP and LaSDPC for offering the necessary equipments for this study. Some of this work was conducted while Stephan Reiff-Marganiec was on study leave from the University of Leicester.

References

[1] Apache CXF - how it works. Avaliable in http://cxf.apache.org/docs/custom-transport.html. Last access: 06/06/2013.

[2] Arduino homepage. Avaliable in http://www.arduino.cc/. Last access: 06/06/2013.

[3] Welcome to Apache Axiom. Avaliable in http://ws.apache.org/axiom/index.html. Last access: 06/06/2013.

[4] T. Aihkisalo and T. Paaso. Latencies of service invocation and processing of the rest and soap web service interfaces. In *Services (SERVICES), 2012 IEEE Eighth World Congress on*, pages 100–107, 2012.

[5] F. AlShahwan and K. Moessner. Providing soap web services and restful web services from mobile hosts. In *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*, pages 174–179, 2010.

[6] F. AlShahwan, K. Moessner, and F. Carrez. Distributing resource intensive mobile web services. In *Innovations in Information Technology (IIT), 2011 International Conference on*, pages 41–46, 2011.

[7] C. Andrews. Easy as pi. *Engineering Technology*, 8(3):34–37, 2013.

[8] C. Chang, F. Mohd-Yasin, and A. Mustapha. An implementation of embedded restful web services. In *Innovative Technologies in Intelligent Systems and Industrial Applications, 2009. CITISIA 2009*, pages 45–50, 2009.

[9] P. Fan and G. Zhou. Analysis of the business model innovation of the technology of internet of things in postal logistics. In *Industrial Engineering and Engineering Management (IE EM), 2011 IEEE 18Th International Conference on*, volume Part 1, pages 532–536, 2011.

[10] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. AAI9980887.

[11] C. Groba and S. Clarke. Web services on embedded systems - a performance study. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 726–731, 2010.

[12] H. Hamad, M. Saad, and R. Abed. Performance evaluation of restful web services for mobile devices. *Int. Arab J. e-Technol.*, 1(3):72–78, 2010.

[13] K. Hameseder, S. Fowler, and A. Peterson. Performance analysis of ubiquitous web systems for smartphones. In *Performance Evaluation of Computer Telecommunication Systems (SPECTS), 2011 International Symposium on*, pages 84–89, 2011.

[14] M. Jansen. Evaluation of an architecture for providing mobile web services. *International Journal On Advances in Internet Technology*, 6(1 and 2):32–41, 2013.

[15] D. Jayasinghe. *Apache Axis2 Web Services, 2nd Edition*. Packt Publishing, February 2011.

[16] R. Kanagasundaram, S. Majumdar, M. Zaman, P. Srivastava, and N. Goel. Exposing resources as web services: A performance oriented approach. In *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2012 International Symposium on*, pages 1–10, 2012.

[17] R. Mizouni, M. Serhani, R. Dssouli, A. Benharref, and I. Taleb. Performance evaluation of mobile web services. In *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*, pages 184–191, 2011.

[18] L. H. Nunes, L. H. V. Nakamura, H. F. Vieira, R. M. O. Libardi, E. M. Oliveira, L. J. Adami, J. C. Estrella, and S. Reiff-Marganiec. A study case of restful frameworks in raspberry pi: A perfomance and energy overview. In *International Conference on Web Services (ICWS), 2014 21th IEEE International Conference on*, pages 722–724, 2014.

[19] A. Papageorgiou, J. Blendin, A. Miede, J. Eckert, and R. Steinmetz. Study and comparison of adaptation mechanisms for performance enhancements of mobile web service consumption. In *Services (SERVICES-1), 2010 6th World Congress on*, pages 667–670, 2010.

[20] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.

[21] Q.-D. Vu, B.-B. Pham, D.-H. Vo, and V.-H. Nguyen. Towards scalable agent-based web service systems: performance evaluation. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, iiWAS '11, pages 481–484, New York, NY, USA, 2011. ACM.