IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 30, 2017, accepted June 23, 2017, date of publication August 8, 2017, date of current version September 6, 2017.

*Digital Object Identifier 10.1109/ACCESS.2017.2728720*

# NPi-Cluster: A Low Power Energy-Proportional Computing Cluster Architecture

**SEBASTIÃO EMIDIO ALVES FILHO[1], AQUILES MEDEIROS FILGUEIRA BURLAMAQUI[2], RAFAEL VIDAL AROCA[3], (Member, IEEE), AND LUIZ MARCOS GARCIA GONÇALVES[2], (Member, IEEE)**

[1]Departamento de Informática, Universidade do Estado do Rio Grande do Norte, Mossoró 59610-090, Brazil
[2]Universidade Federal do Rio Grande do Norte, Natal 59078-970, Brazil
[3]Universidade Federal de São Carlos, São Carlos 13565-905, Brazil

Corresponding author: Sebastião Emidio Alves Filho (sebastiaoalves@uern.br)

**ABSTRACT** This paper presents the NPi-Cluster, an energy proportional computing cluster that automatically powers ON or OFF the number of running machines according to the actual processing demand. A theoretical model is proposed, discussed, and implemented on a cluster composed of Raspberry Pi computer boards designed and built in order to test the proposed system architecture. Experimental results show adequate performance of the proposed platform when compared with other web servers running on traditional server architectures, but with considerably less power consumption. The power consumption of the entire cluster is about 14 W when running at maximum performance. In this situation, the system is able to handle more than 450 simultaneous requests, with about 1000 transactions per second, making it possible to be used as a server capable of handling real web workloads with acceptable quality of service. When the requests demand is reduced to a minimum, the power consumption is dynamically reduced until less than 2 W. Additionally, the proposed cluster architecture also provides high availability by reducing single points of failure on the system.

**INDEX TERMS** Energy efficiency, scalability, quality of service, distributed computing, low power electronics.

## I. INTRODUCTION

One of the biggest challenges for green computing is the reduction of energy consumption in data centers. In recent years, various techniques and solutions have been proposed to alleviate this problem, among them the energy proportionality and the use of architectures or clusters based on servers with ultra low power consumption are used. In this paper, we propose the NPi-Cluster, a low power consumption cluster architecture with energy proportionality, capable of automatically scaling the number of running machines according to the current processing demand. As a proof of concept, a cluster consisting of 7 nodes is presented, which acts as web server subjected to different demands of webpage requests. Raspberry Pi computers are used as the processing unit of each node.

Experimental evaluation shows that the proposed cluster has better energy efficiency when compared to other machines with ARM and x86 architectures. In addition, the maximum power consumption of the proposed system is 14 Watts: less than the energy used by an energy-saving light bulb, even with request response times similar to a machine with Intel Xeon processor with 4 cores, a machine typically used for enterprise grade servers. The results also show a notable performance difference between the Raspberry Pi boards with single-core and multi-core processors, in both cases having their performance limited by the network adapter. Tests performed with the proposed system show also that the use of energy proportionality enables a better efficiency when there is a lower demand. In that way, one contribution of this work is a dynamic provisioning algorithm and overload detection technique that is able to automatically compute the adequate number of active nodes, activating new nodes as fast as the demand grows up, which provides a trustworthy and predictable control system for energy efficiency.

Towards meeting the green computing paradigm, another contribution of this article is a web server architecture for clusters with load balancing features, scalability and low power consumption through hardware proportionality. Thus, it is known that green computing requires more efficient systems, so server systems should have energy consumption

adjusted according to their workloads, but unfortunately this does not occur frequently in modern servers [1]. In most cases, traditional computers typically consume dozens or even hundreds of watts (W) of power, and are often underutilized. One example from the academic environment, is that each laboratory has its own server, which is always powered on and consuming the same amount of power, but may be receiving only a few accesses per day. On the other hand, embedded computing systems such as smartphones and tablets have always been designed to optimize the use of energy, seeking to increase the duration of their batteries. One typical platform used in embedded systems is based on ARM processors, one of the most used architectures in the world, and commonly used in embedded systems and in the mobile industry [2]. Some works present ARM based low power clusters, where nodes are turned on all the time. Another approach used in traditional clusters with energy proportionality consists in setting underutilized nodes in suspend mode, thus consuming a reduced amount of energy. In these cases there is a consumption reduction, but there is still energy to be saved.

In order to provide a scalable cluster with low power consumption, we use the Raspberry Pi (RPi) boards as nodes of the proposed system, which was named NPi-Cluster. In such cluster, nodes are turned on and off automatically according to the overall system workload. NPi-Cluster architecture has been experimentally tested on two models of a 7-node RPi web server cluster and validated during several months of exhaustive automatic testing with several load conditions, which results in a system that presents a performance of 140 transactions per second per Watt with more than 450 simultaneous connections. The system also provides high availability, as in the eventual failure of any node, the remaining nodes are capable of taking over the services of failed nodes. In addition, we introduce quantitative and qualitative results carried out on the performance of the RPi as a web server and a comparison of these results with the performance of traditional web servers.

Besides the proposed system architecture and proof of concept system built and validated, we also propose algorithms for making the system work properly according to the necessary amount of nodes necessary for a given scale of workload. Moreover, the proposed architecture is generic: though validated with RPi boards, it is applicable to any type of computer architecture. So, although the RPi has been designed initially with an educational focus, it is explored in this article as a web server with low cost and low power consumption.

This text is structured as follows: sections II and III discusses the theoretical aspects and background of the proposed system, including a discussion of green computing aspects and related works. Section IV depicts details of the NPi-Cluster architecture, while section V presents and discusses the obtained results. Finally, section VI brings our final considerations regarding the conducted experiments, results and concluding remarks.

## II. GREEN COMPUTING

Paek [3] presents a different perspective on belief that information technology (IT) is environmentally friendly, even allowing to save materials that come from nature, such as paper. He argues that IT brings side effects such as the disposal of electronic products in short periods of time and causes increasing need for electricity, thus causing damage to nature in the same way, for our generation and for future generations. In this direction, efforts and initiatives denominated as Green Computing or Green IT are being adopted to attenuate such problem, which aims for more environmentally friendly computing.

Agarwal and Nath [4] points out that there is a growing global movement to implement *Green computing* that represents an environmentally responsible way to reduce power and environmental waste. He argues that green computing is about the environmentally friendly usage of computers and related technologies and that current trends include energy efficient CPUs and peripherals, power reduction through known energy-conserving approaches, and the proper recycling and disposal of all the components. Issues related to the use of toxic materials, recycling and e-waste disposal are well known by industry, which includes Green Chain Supply Management (GCSM) practices on products life cycle as discussed in the work of Srivastava [5]. Standards and certifications have been proposed too [6], but none of them are globally required. In such a way, the Green IT research has, currently, a major focus on energy consumption (or energy saving).

The *Key World Energy Statistics* report from the International Energy Agency [7] claims that more than 80% of primary energy supply uses oil, natural gas and coal, which are non-renewable energy sources that produce pollution, contributing to the greenhouse effect. In addiction, a report from Gartner Inc. [8] estimates that the information and communication industry accounts for approximately 2 percent of global carbon dioxide ($CO_2$) emissions. Thus, reducing energy consumption is fair not only for environmental protection, but also for organizations in order to save money. Nonetheless, this reduction should not affect productivity significantly.

As said, one goal of Green IT is to provide energetically efficient systems, for example, by fulfilling the largest amount of work as possible, given a power consumption level limit. In general, considering each equipment individually, manufacturers try to keep performance while requiring less energy to perform the same task. The most simple strategies for energy consumption reduction are to turn off components or to activate low power modes, when available. More complex strategies can also be adopted with changes at the architecture level, using multicore processors, or with the design of energy proportional hardware.

Multicore machines have two or more processing units in the same physical processor, sharing the same power supply, the same cooling system and other components. According to Blake [9], the main advantage of such systems is the

performance enhancement that is achieved by adding new cores that might have a reduced consumption when not used. This provides better energy efficiency if compared to the energy necessary for executing the same tasks in a single core system with a higher clock frequency, or yet if another complete processing unity would be attached separately. On the other side, the use of energy proportional hardware makes possible to define different levels of performance according to some criteria of use. For example, if the system is idle, ideally it should have no energy consumption. With little processing demand, the system would a small power requirement and would have to use the maximum of its performance for higher demands of processing [10]. A known technique that can be used in such cases is the dynamic voltage and frequency scaling (DVFS), in which steps of clock frequency that use different levels of power can be defined [11]. However, as DVFS systems have a number of fixed/discrete speed/frequency/power levels, it must be analyzed if the energy to be saved compensates the time taken with the levels changing and step granularity.

Energy consumption can not be thought individually for each equipment. With the increasing demand driven by the information technology and communication industries, and with the popularization of cloud computing, servers that were decentralized in office buildings are now concentrated in specialized data centers with clusters of dedicated computers in order to provide services. According to Bianzino *et al.* [12], data centers rely on machines with lots of resources and are generally projected to support peaks usages and adverse conditions. As depicted by Kaur and Chana [13], with the sudden rise in electricity consumption levels, many peer-reviewed studies were conducted to analyze the problems and levels of energy consumption in the data centers. In their work, they provide a survey and taxonomy for the main techniques for dealing with energy efficiency in cloud computing. Energy optimization measurements in the data centers are classified as location-based, hardware-based, software-based and infrastructure-based.

Location-based approaches propose that the choice of the places for data centers implantation should take into account the presence of environmentally friendly energy sources near the planned data center, for example, near an hydroelectric or a wind power plants. Infrastructure-based approaches are generally related to green buildings. That is, physical buildings that provide a better use of energy with lower consumption for non IT equipments, mainly the heating and cooling system. Techniques known as hardware-based focus on reduction of consumption in each equipment just as the ones cited previously. Finally, the software-based approach for green computing take into account the techniques performed in software design or through its use. Finally, others techniques use the network and the hardware characteristics in order to enhance performance or to share resources, as parallel programming, virtualization, resource throttling, resource provisioning, and scheduling techniques.

The work of Chen *et al.* [14] describes two techniques that we used to reduce energy consumption in this work: dynamic provisioning and load distribution. In dynamic provisioning, the machines can be turned on or off according to the demand, similarly to what is proposed in this work. If the services of a machine are underused, they can be migrated to another machine and the underused one can be shutdown until an increase in processing demand makes the other servers overloaded. Later, the machine can be turned on again, thus providing energetic scalability. The load distribution is used when a service is running with redundancy in more of one server or when a server is overloaded with several requisitions. In this case, services are migrated to machines that are less utilized, thus also contributing for an improvement in the service quality.

## III. RELATED WORK

With the advances of computer technologies, high performance processors have been developed, as the ARM architecture. These specific platforms are designed and developed to support network operating systems, thus making possible the implementation of a web server with low energy consumption. For example, a study [1] has demonstrated that a Cortex-A9 ARM processor presents energy efficiency up to eleven times superior than a classical server based on a Intel Xeon processor. Another work Aroca and Gonçalves [15] analyzes two ARM based systems, a PandaBoard and a BeagleBoard. The authors provide systematic measurements of performance and energy efficiency of several systems running with the x86 architecture in comparison to ARM Cortex-A8 and ARM Cortex-A9 processors. It is experimentally demonstrated that the ARM architectures are capable of attending hundreds or even thousands of web requests per second. Besides, it is also observed that ARM processors have better energy efficiency than several computers based on the x86 architecture.

Regarding clusters based on ARM processors, one of the first prototypes presented is composed of 196 ARM Cortex-A8 boards [16]. A similar, however smaller, project is the *Apple TV Cluster* [17], that is composed of six nodes. An important initiative is the *Mont Blanc* [18] project that aims to build an energy efficient ARM CPU/GPU based cluster.

Several other works have used clusters with different configurations (refer to Table 1) for different applications. Experimental evaluations [19]–[24] show that low-power clusters present a good performance for classical scientific and mathematical applications. They are usually submitted linear systems solving benchmarks (the LINPACK [25] and HPL [26]). In another work [27] a cluster was used to compile all the packages of the Ubuntu Linux for the ARM architecture.

Some works present solutions to build distributed web servers using a cluster [28]–[30]. Others [31], [32] relies on cluster nodes to support cloud computing infrastructure. Distributed filesystems is another application that has been receiving attention [23], [28], [30], [33]–[36], specially the

**TABLE 1.** Technical features about low power systems from literature.

| Work | Processor | Platform | # Nodes | Power (node) |
|---|---|---|---|---|
| Andersen et al. [40] | AMD Geode | Alix | 21 | 3-6 W |
| Brown [16] | ARM C-A8 | Gumstix | 192 | n/a |
| Fürlinger et al. [17] | ARM C-A8 | AppleTV | 4 | 2-3 W |
| Keville et al. [19] | ARM OMAP4 | PandaBoard | 40 | 3-5 W |
| Balakrishnan [20] | ARM C-A9/ARM11 | Panda/RPi | 6/2 | 1-5 W |
| Ou et al. [39] | ARM C-A9 | PandaBoard | 4 | 2-5 W |
| Abrahamsson et al. [31] | ARM-11 | RPi | 300 | 1-4 W |
| Rajovic et al. [21] | ARM C-A9 | SECO Q7 | 256 | 4 W |
| Tso et al. [32] | ARM-11 | RPi | 56 | 1-4 W |
| Pfalzgraf and Driscoll [22] | ARM-11 | RPi | 22 | 1-4 W |
| Cox et al. [23] | ARM-11 | RPi | 64 | 1-4 W |
| Kaewkasi and Srisuruk [34] | ARM C-A8 | CubieBoard | 22 | 4-10 W |
| Krpić et al. [24] | ARM C-A7 | CubieBoard | 4 | 4-10 W |
| Fox et al. [33] | ARM-11 | RPi | 2 | 1-4 W |
| Schot [35] | ARM C-A7 | RPi2 | 8 | 1-4 W |
| Velthuis [38] | ARM C-A7 | RPi2 | 3 | 1-4 W |
| Loghin et al. [30] | ARM C-A7/C-A15 | ODROID | 8 | 1-3 W |
| Mont Blanc [18] | ARM C-A15/T-604 | Samsung | 2160/1080 | 4-8 W |
| Zhao et al. [28] | Atom | Intel | 35 | 1-2 W |
| Xu and Chang [36] | ARMv7 | Marvell | 16 | 10 W |
| NPi-Cluster | ARM C-A7 | RPi2 | 7 | 1-4 W |

Hadoop [37]. Finally, other works [38], [39] discuss the usage of low power devices to provide services for video streaming.

However, some experiments and works do not present good results for all computer boards and applications, especially due to hardware limitations. Therefore, the application performance is as important as choosing low power consumption devices for the cluster composition. Abrahamsson *et al.* [31] describe tools for configuring cloud services, monitoring, and for infrastructure maintenance, which consume less resources than traditional ones. Tso *et al.* [32] present a system using Linux Containers, a lightweight operating system-level virtualization method for running multiple servers on a single host control.

For video manipulation, Velthuis [38] argues that the performance is not good, showing that the limited bandwidth and CPU performance affect the user experience. Ou *et al.* [39] explains that as the demand increases, more nodes would be needed on the cluster, reducing energy efficiency unlike other architectures. Finally, Loghin *et al.* [30] indicates that I/O-intensive MapReduce workloads are more energy-efficient on the x86 machine while database query processing was always more energy-efficient on ARM servers.

Table 1 shows the main characteristics of the selected works found in the literature. NPi-Cluster shares their advantages and limitations, but none of the cited applications has energetic scalability. The nodes are permanently turned on in all mentioned works, independently of the amount of workload. In other works [29], [41], [42], energy proportionality is achieved by placing the devices in suspend mode. In this mode, they consume less power and would take less time to return to normal operation. However, with suspend mode, the devices continue to consume an amount of energy that would be enough to sustain one or more RPi boards operating with maximum throughput. Furthermore, in our tests, after a RPi boots, it takes about 10 seconds to become fully operational.

## IV. NPi-CLUSTER ARCHITECTURE

The proposed architecture aims to enable the construction of low-cost clusters, with energy consumption proportional to its workload and also with high availability feature, thus being energy efficient. The energy efficiency is not only from the processor point of view. Computers and traditional clusters use hard disks, cooling fans, and other mechanical devices that wear out and eventually break with use, even causing unavailability of the system if not using redundancy. Server power consumption is not only related to its processing either. Heller *et al.* [43] claim that the largest consumers of energy in data centers are servers and cooling systems. The use of an ARM based system leads a lower demand for refrigeration, once these systems are projected for dissipating a much lower level of heating, with a passive ambient cooling [23]. In fact, we noticed during the tests that the temperature of the processor keeps below $45°C$, even without using any active cooling system.

Investigations about energy performance and its trade-offs, conducted with different ARM development boards, pointed that the Raspberry Pi was the best option to

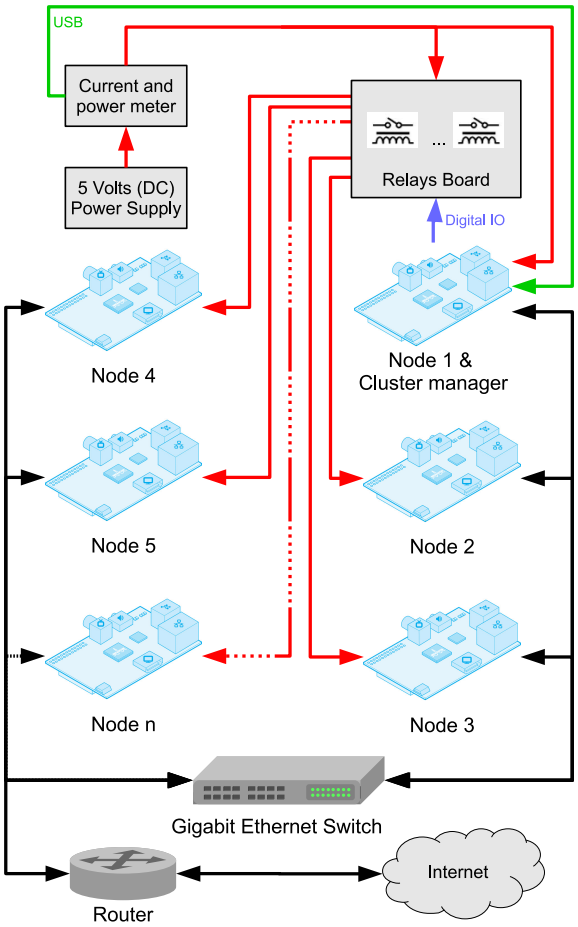


**FIGURE 1. Overview of the NPi-Cluster architecture with its electrical and networking connections shown.**

build a power-aware low cost embedded cluster [44], [45]. RPi has also shown to have one of the greatest cost/benefit when considering performance per cost. Moreover, it uses ARM processor, which has a low power design since its early project conception. As described by Tso *et al.* [32], some of the platforms developed in the Raspberry Pi project [46] look as "toy-devices" and have not been designed to perform the same amount of compute-intensive operations per unit of time as on traditional servers. However, these devices have been successfully used recently in clusters because they share many of the properties of cloud-based servers, such as limited storage and peripheral capability, albeit at a smaller scale. Such findings support the design decision made on this work, which proposes a cluster based on the Raspberry Pi boards.

Figure 1 shows an overview of the NPi-Cluster architecture. Several nodes are connected through a Gigabit Ethernet bus (in black). As shown by the red lines, only one node is directly connected to the power supply, while the power supply to all other nodes is managed by an 8-relay board, that allows each node to be fully powered on or off by digital commands sent from the cluster manager node, which also has the ability to monitor the system power usage by a USB power meter connected between the cluster and the power supply. Power measurement does not include the Ethernet switch because our focus is on the server consumption, but a typical 8-port desktop switch needs up to 5 Watts in full operation, which does not occur in the experiments.

Electrical connections going through relays can be normally closed (NC) or normally open (NO). Normally open connections only provides power to the connected device when the controlling unit (node manager, in this case) enables a digital output line, while normally closed will turn off the power when the digital input line is enabled. The digital lines used are output ports available in the General Purpose Input Output (GPIO) ports of the cluster manager Raspberry Pi. The proposed design uses normally closed contacts as a high availability approach: if the cluster manager fails or if digital output lines fail or even if the relay board fails to operate, the normal state of the relays is closed, so all boards are powered on. In fact, this is the default state when the system start up ends: all nodes are up and running even if the manager or the relays board fails. In that way, a manager failure does not mean a complete cluster failure but only a temporary loss of its dynamic aspect. However, it should be noted that such situation is difficult to happen because the manager does not receive any workload.

Right after the start up sequence, the node manager detects no workload and turns off all other nodes by opening the normally closed relays via the GPIO interface and the relays board. For the proposed setup, two RPi boards are turned on at the beginning of each experiment, one for serving HTTP requests and another for managing the cluster resources, however only the cluster manager could be on, aggregating the manager and service handler. As more processing is required, the manager provides the dynamic node provision. Thus, other nodes are turned on (or off) automatically through relays that are connected to the digital signal outputs of the other nodes. It should be clear that although the system was built with Raspberry Pi boards, each with a 100 MBit/s network interface, the usage of a 100 MBit/s switch should be avoided, as in experimental evaluation it caused considerable cluster performance bottleneck due a bandwidth usage about 350 MBit/s. In that way, a Gigabit switch must be used to connect all the parts.

Besides low energy consumption, it should be noted that some computers consume about 10 $W$ of energy simply by being in standby mode, however, these machines can not execute any operation. On the other side, a RPi machine at 700 MHz consumes less than 2 $W$ (in our tests) and it could execute several tasks thus corroborating with the studies described by Kaup *et al.* [47]. In that work, boards consume 1.75$W$ with CPU-bound applications and 2.1 $W$ for wi-fi data transfer. Another advantage of an ARM based system is space saving, as this could be a problem in certain cases [48].

### A. LOAD BALANCING AND DYNAMIC PROVISIONING

Load balancing in distributed systems is a well known problem with different approaches for solving it [49]. Among the solutions, one that is more generally applied to compact

systems is the centralized technique, that presents an optimally global performance. In such approach, all the nodes are passive regarding the reception and execution of tasks, which is managed by a central controller. Two of such approaches are discussed in the work of Cardellini [50].

The first is called DNS-based, where the name server is properly responsible for distributing the tasks between a pool of servers, according with the demands that come from the clients. In the Dispatch-based approach, there is a mediating component between the clients and the cluster. This element is responsible for verifying the load attributed to each node and to send solicitations to the nodes that have better conditions to serve them. Besides, another distributed approach known as server-based is also introduced by Cardellini [50], which is adequate for the local area network context. We tested all of these three solutions using the approaches mentioned above.

For the Dispatch-based, we use the proxy balancer module of the Apache web server [51] and the TCP/HTTP HAProxy load balancing software [52]. Note that both of them work in a similar way, receiving requests and sending them to the available servers, equally, randomly, or according to the number of answers received. However this approach could not work as well as desired, as the dispatcher is known to be the bottleneck of the system [50]. In fact, our experiments did not show performance gains when more than two nodes were used, due to the dispatcher processing limitation, that may have been limited by hardware or software resources.

For the second tested approach (server-based), the scalable high availability cluster resource manager Pacemaker [53] is used. When operating, the resources and services made available by the cluster are managed in two ways. They can be divided among available nodes and can migrate in case of fails (active/passive configuration) or they are equally available in all nodes that perform the load balancing (active/active configuration). In both cases, there is a large amount of exchanged messages in order to keep the high availability. Conducted tests have shown that such setup can affect the performance substantially in Raspberry Pi nodes.

In that way, the chosen solution for distributing load is the DNS-based approach, more specifically the *Round-Robin DNS* (RRDNS). With this technique, the DNS server answers each name resolution request in the Internet with a different IP, generally in a sequential way [54]. For example, in the case that a client tries to access the address http://www.example.com and supposing that this one has a sequential range of addresses reserved starting with 93.184.216.34, the DNS server will answer the first request with the IP 93.184.216.34. In the following request, the IP will be 93.184.216.35. Next, the response will be 93.184.216.36 and it will proceed in this way until the end of the IP list is reached, then the process restarts with the first IP. The use of this technique allows direct, distributed and easy implementation of a load balancing system as it can be noted that it is often used joint with other techniques based on the pooling of IP in network firewalls and switches [55]. In such cases, it is possible to attribute weights to determined addresses or yet distribute load based on some criteria as the minimum number of active connections.

A possible problem with this approach is that the DNS server configurations should be changed every time a node is inserted or removed from the cluster, in order to have a consistently updated list of IP addresses. To avoid this, we use a static set of addresses that is proportional to the number of nodes in the cluster. This set is distributed among active nodes using IP aliasing, which allows the same network card to be configured with different addresses. In this way, every time that a user requires a web page, the RRDNS server answers with a different IP that is not necessarily in a different machine.

The resource manager constantly monitors accesses and, when the load exceeds some configurable threshold, the manager node automatically turns on the next RPi and stays monitoring the instant when this one will be available to service requests. Once it gets ready to receive web requests, the manager node sends a message to others RPis deactivating an address of its list of configurable IP addresses that are, then, activated on the newer one. As the Ethernet switch can have the correspondence between the physical network address (MAC) and the IP address stored in cache, the nodes also send ARP messages to update the MAC table of the switch. From this moment on, each node has a proportional list of the IP addresses, thus each node receives a proportional number of the HTTP requests. This IP division process happens while there are available nodes. Figure 2 shows an example for the case in which only a single node is on (top) and three nodes are on (bottom). In such case, the 6 red labels represent IP addresses attributed to the hosts, initially all of them concentrated in a single node and divided among the nodes in the second situation (two for each node).

When the processing load decreases below a configurable limit, the last active nodes are turned off one by one. This can happen until only a single node is active (or a defined number of nodes). It can be noted that the number of active nodes decreases, but is enough to still provide processing capacity to service requests, thus keeping processing capacity above the minimum limit. Also, it is important to set a time interval for measuring the processing load of the nodes in order to avoid an oscillatory behavior of on/off operations and avoiding the system of loosing control. This technique, which is called Vary-On Vary-Off (VOVF), turns nodes on and off to adjust the number of active servers according with the workload. The work of Zheng and Cai [56] shows a mathematical model of energy proportionality based on service quality metrics (slowdown and request time), similar to those presented in this work, but just simulated.

Algorithm 1 details the processing of the cluster control to perform the dynamic provisioning. Note that four states are established: low, normal, high and critical. In the normal state, it is considered that the number of active nodes is adequate to the demand. In the low demand, the cluster would be underutilized thus meaning a energy waste, so some nodes could be

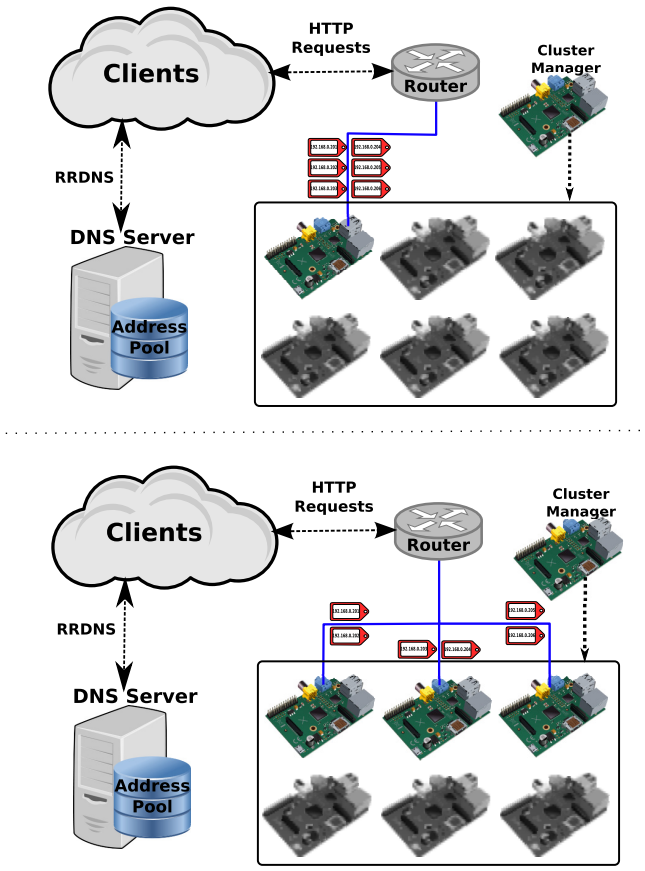


**FIGURE 2.** Example situations of the load balancing by IP in a 6-node NPi-Cluster with 1 and 3 hosts turned on.

deactivated. In the high and critical demands, the nodes are overloaded, with the total cluster processing capacity below the needed. The critical state is undesired, since it may cause some requests to be thrown away besides a simple delay in the answer that configure the high demand.

Some parameters must be provided to the algorithm: the set of nodes (*Nodes*), the number available addresses (*Addresses*) for the balancing and the minimum number of nodes that must be on. Utilization rate thresholds are defined as ($t_{high}$, $t_{critical}$) for classifying non normal use of the cluster (overloaded). In order for the cluster to be set in a given state, the corresponding value have to be kept in the same state using the above limits during a determined number of times ($r_{low}$, $r_{high}$, $r_{critical}$) in repetitive measures done regularly at time intervals (*sleep_interval*).

To avoid influence of an isolated measure on the previous ones, some counters are defined at line 4 ($c_{low}$, $c_{high}$, $c_{critical}$). During execution these counters are incremented each time a state is detected and decremented in case it does not occur in the current measure. For being considered as a high or critical utilization state, the current level of use $u_{now}$ of the cluster (line 6) should be higher than the thresholds (lines 7 to 10). The mean of the utilization rate for the current demand is calculated for $u^*_{now}$ considering less one node in the cluster (line 12). It can be considered in the low state if this average is below $t_{high}$ (line 13). That is, the cluster would not get overloaded in the case that one node is deactivated. In the case that one of the counters accumulate the number

**Algorithm 1** Algorithm for NPi-Cluster Dynamic Provisioning

**Input:** cluster features
$Nodes = \{RPi_1, RPi_2, \ldots, RPi_n\}$ set of $n$ work nodes
$Addresses = \{IP_1, IP_2, \ldots, IP_a\}$ set of $a$ addresses in the DNS server($a \geq n$)
$t_{high}, t_{critical}$ tresholds for the cluster utilization state
$r_{low}, r_{high}, r_{critical}$ number of repetitions of a same state in order for the controller actuate
*min_nodes* minimum number of active nodes all the time
*sleep_interval* time interval for cluster state checkings
***Initialisation*** :
1: $Active = Nodes$ set of active nodes
2: $Inactive = \varnothing$ set of inactive nodes
3: $c_{low}, c_{high}, c_{critical} = 0$ counters (positive integers) for number of accumulated repetitions of each state
4: Assign virtual IPs in *Addresses* to *Active* nodes
***Control loop*** :
5: **while** *true* **do**
6: $u_{now} = average\ (u_i)$ where $u_i$ a weight representing the resource utilization at node $RPi_i \mid RPi_i \in Active$
7: **if** ( $u_{now} > t_{critical}$) **then**
8: Increment $c_{high}$ and $c_{critical}$, decrement $c_{low}$
9: **else if** ( $u_{now} > t_{high}$) **then**
10: Increment $c_{high}$, decrement $c_{low}$ and $c_{critical}$
11: **else**
12: Compute $u^*_{now} = \frac{\sum u_i}{|Active|-1}$ resource utilization if an active node would be turned off ($|Active| > 1$)
13: **if** ($u^*_{now} < t_{high}$ **and** $|Active| > min\_nodes$) **then**
14: Increment $c_{low}$, decrement $c_{high}$ and $c_{critical}$
15: **else**
16: Decrement $c_{low}$, $c_{high}$ and $c_{critical}$
17: **end if**
18: **end if**
19: **if** ($c_{high} > r_{high}$ **or** $c_{critical} > r_{critical}$) **and** $|Inactive| > 0$) **then**
20: Turn on a node $RPi_j \in Inactive$
21: $Active = Active \cup \{RPi_j\}$
22: $Inactive = Inactive \setminus \{RPi_j\}$
23: **else if** ($|Active| > min\_nodes$ **and** $c_{low} > rlow$) **then**
24: Turn off a node $RPi_k \in Active$
25: $Active = Active \setminus \{RPi_k\}$
26: $Inactive = Inactive \cup \{RPi_k\}$
27: **end if**
28: **if** ( $|Active|$ changed ) **then**
29: Reassign virtual IPs to *Active* nodes proportionally
30: Reset counters $c_{low}, c_{high}, c_{critical} = 0$
31: **end if**
32: Sleep for a *sleep_interval* time
33: **end while**

of repetitions enough for characterizing an state, the cluster should turn on or off a node (lines 19 to 27) and reassign the IP accordingly (line 29). To the end of each run there is a pause to then proceed with the next measurement (line 32).

### B. OVERLOAD CRITERIA

In principle it would be natural to establish measures of CPU and memory usage as the overload criteria of a server. However, we noticed that the CPU usage varies a lot in a short time slot and that some processes consumes a substantial processing time to handle input-output interruptions. On its turn, memory is not substantially affected with the increasing in processing demand. In this way, as we could not observe a straight relation between server demand and state of use of the CPU, we decide to adopt another criteria for overload detection.

A key issue that can determine the success or failure of a web system is the time spent for loading pages. An investigation done with more than 25 hundred users of on-line stores at England and USA [57] shows that two thirds (67%) of UK and more than a half of US shoppers (51%) cited the slow loading times as the top reason for giving up a purchase. Thus, when considering the performance of a web server cluster, its capacity to answer the higher number as possible of requisitions at a small time slot has to be taken into account. Some steps for answering an HTTP request are cited by Dilley et. al [58]: 1) client sends the request; 2) request is received in the server side; 3) request is sent to the service process; 4) a parsing is performed on the request; 5) the request is processed; 6) an answer is sent to the client; and, 7) finally, the client receives the answer. The time spent for completing all of the above processing is called request-response time. It can be divided in two phases. The connection phase involves all steps until the requisition becomes to the server process. In the response sending phase the server effectively attends the client requisition (see Figure 3).

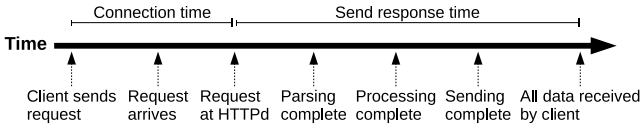


**FIGURE 3.** Phases of a HTTP request-response processing (adapted from [58]).

For the server connection, all the requests follow the same processing sequence, independently of their content. The nature of the request (static or dynamic) is only detected after the parse phase of the server process. In the context of static files (HTML files, style-sheets, images, scripts, and others) the time for sending a response refers to the time spent for locating the file on the disk and sending it to the client. This depends, basically, of the size of the file to be sent. For dynamic contents, it is hard to estimate the amount of time taken to process each request because this might depend on several operations, such as access to a database and processing of the obtained data. Nonetheless, on other than synchronous services as streaming the server transforms the response in static contents and send it to the client. Thus the request-response time can be given approximately by:

$$t_{response}(page) \approx t_{connection} + t_{send}(page) \quad (1)$$

Here, the connection time is assumed to be constant for all requests and the sending time increases according to the size of the page. Assuming that an empty page (file without any content) would not have a relevant sending time ($t_{send}(\varnothing) = 0$), the connection time could be understood as the time to access an empty page, that is:

$$t_{response}(\varnothing) \approx t_{connection} + t_{send}(\varnothing) = t_{connection} \quad (2)$$

To provide a function for approximating the sending time according to the size of the page we have done experiments in one of the cluster machines. Basically, we provide HTML pages with sizes varying from 3 to 30 KB, incremented by 3 KB each time. For each page size, requests from a client are done during 5 minutes and each test is repeated 10 times. At the end, we calculate the average response time of each request, which is summarized in the plot shown in Figure 4. These results are then smoothed by a Bézier filter. This shows that the growing of response time is somewhat linear with respect to the page size. With this, we can get the relationship $t_{send}(page) \approx t_{perbyte} * size(page)$. In this way, Equations 1 and 2 can be obtained.

$$t_{response}(page) \approx t_{response}(\varnothing) + t_{perbyte} * size(page) \quad (3)$$

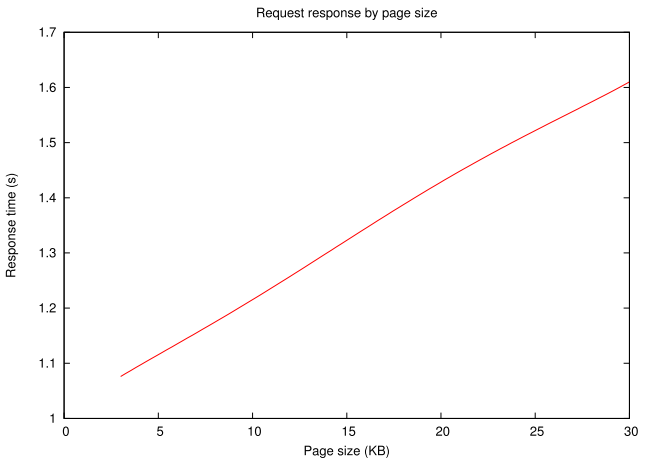


**FIGURE 4.** Relation between time spent in each requisition and the size of the requested file for a web server on Raspberry Pi machine.

This time refers to a single request, however, as the cluster proposal is to treat situations of high demand, it is necessary to determine when a node is overloaded (or not). If a server receives several requests simultaneously, not all of them are immediately processed. Some of them are received and stay waiting to be sent to the referring processes in the server and other are received and put in a queue (say of type first in first out) in order to be sequentially processed. Even the ones being processed have to wait and share the processor

time according to the operating system scheduling algorithm. In this way, the connections established at the web server port (in general the 80) consumes a certain time to be processed that would cause a delay at the next requisitions that arrive. This time can be calculated from Equation 3 as:

$$t_{delay} \approx \sum_{page_q \in Queuedrequests} t_{response}(page_q) \quad (4)$$

In order for this approach to work, it is necessary to previously know the size of the file, before the the parsing is performed, but this is not possible. However, a previous study of the accesses patterns of the web server allows to estimate the average size of the files that compose the pages that are requested and assume that the forthcoming requisitions would have this size. In this way, it is possible to estimate the time that it takes to process all the requests that are in the row updating Equation 4 as:

$$t_{delay} \approx \mid Queuedrequests \mid \times t_{response}(averagefile) \quad (5)$$

The resource utilization $u_i$ for the proposed dynamic provisioning algorithm is calculated by taking $t_{delay}$ values for each node $RPi_i \in Active$ set. In this way, it is possible to define that a host is overloaded when average response time expectation is above the acceptable threshold by a certain period of time.

## V. RESULTS AND DISCUSSION

Section III discusses that some works use equipments with ARM architecture for cluster implementations and, further, section IV discusses the low energy consumption of the RPi boards. In this direction, we aim to verify the viability of the use of a cluster with dynamic provision composed of Raspberry Pi nodes. The NPi-Cluster includes load balancing for HTTP service from the performance, quality of service, and energetic efficiency perspectives. For this empirical verification, two experiments are performed: the first one verifies the performance of the cluster on a static manner, that is, with all devices continuously poewered on. The second experiment compares the static and the dynamic provisioning with the overload criteria proposed in Section IV. These experiments are conducted for two models of RPi boards, whose characteristics can be seen in Table 2, one of them is a single core and the other is multicore.

**TABLE 2. Features of the Raspberry Pi machines used in the NPi-Cluster experiments.**

| Model | Features |
|---|---|
| RPi Model 1 Revision 1 | Single core ARM1176JZFS Processor at 800 MHz, 256 MB RAM, 4 GB SDCard |
| RPi Model 2 | Quad-core ARM Cortex-A7 Processor at 1 GHz, 1GB RAM, 16 GB SDCard |

Software used on each node is the Nginx [59] web server with its standard configurations over the GNU/Linux Raspbian distribution [60]. The web content served is stateless, that is, no session or cookie is created. Both clusters use 7 machines and a Gigabit Ethernet switch in order to connect them to a wired network. For the RPi model 1 based cluster, all of the nodes are for requisition processing while for the cluster based on RPi2, six of the nodes are used for serving web requests and the last one is used to manage the cluster. Power measurement is obtained using the same methodology used by Aroca and Gonçalves [15] and a shell script is used to collect these measures automatically. Figure 5 shows a picture of the environment where the experiments were performed. Note that such cluster is assembled as a wall-mountable frame, which can be fixed to walls, reducing the needed space for the system.

**FIGURE 5. Complete NPi-Cluster, assembled on a wall-mount frame (version with Raspberry Pi 1 nodes).**

### A. FIRST EXPERIMENT - ENERGETIC PERFORMANCE WITH STATIC CLUSTER

A performance comparison between x86 and ARM machines is provided in a previous work [15]. In that work, HTTP requests for a page of about 3KB are generated simultaneously. The number of simultaneous requests varied from 1 to 1000, incremented by 25 each time, and, from the responses received, the tool Apache Benchmark (ab) [61] calculated some metrics, such as the number of requests effectively attended per second, throughput, time to serve each request, and availability rate. Data related to performance, temperature, and energetic performance were also obtained.

In that way, we decided to repeat the same experiment described above in order to verify the energetic performance

of RPi based clusters, based on the models listed on Table 2, in comparison to the machines described in the mentioned work. In both cases, all the machines are previously and continuously turned on. For each test batch, the average of the results for 10 different runs is obtained, which requires 150,000 times the same page originally used. The results are presented using graphics (analyzed and discussed next) that are smoothed by using Bézier curves.

The first result to be analyzed (Figure 6) relates to the average power consumption. Note that both clusters consumes roughly the same amount of power (about 14 watts), which is higher than other devices with ARM and Atom processors. This is naturally comprehensible since each cluster has 7 machines. However, even in this case, the cluster presents less power consumption than the Turion processor and almost 4 times less power consumption than the Xeon processor machines.

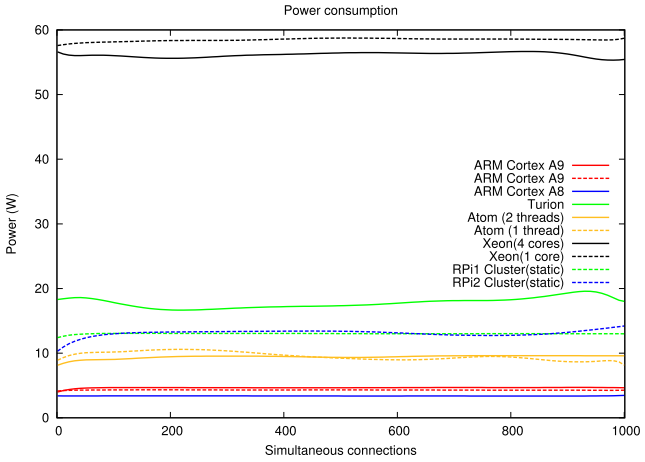


**FIGURE 6.** Average power consumption for the first experiment, with all nodes turned on for each NPi-Cluster model.

An interesting fact is that besides having processors with 4 cores in each node and higher clock speed, the energy consumption of the system based on the RPi2 boards is very similar. Moreover, in the tests where the number of concurrent connections is smaller, the RPi2 consumption is smaller than the RPi1 machines showing the possible action of efficiency policies, such as the DVFS.

Figure 7 depicts the results relative to performance with the number of requests served per second. This first test shows that the clusters have obtained results that are better than the majority of other evaluated platforms. The RPi1 cluster has inferior performance when compared with the Xeon machine with 4 cores and at the initial scenarios to the Xeon with only 1 core enabled. However the RPi2 cluster is better than all the other machines in practically all the scenarios.

The plot shown on figure 7 depicts the performance difference between the two clusters. For RPi2, even with 4 cores and a higher clock frequency in each node, and considering that the requests do not obey an ordered sequence, there is no system or process that can run 100 % in parallel. In fact,

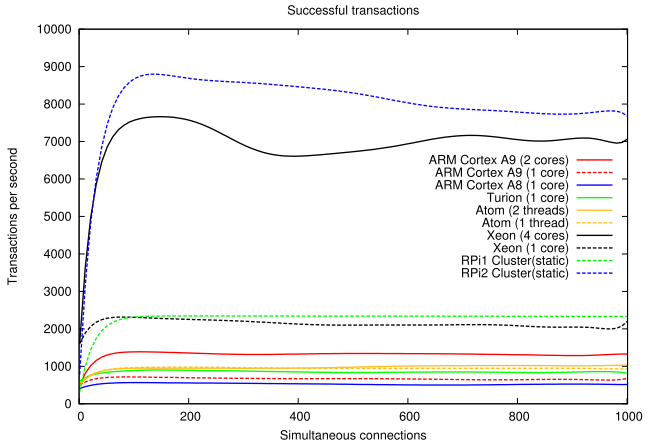


**FIGURE 7.** Comparison of successful transactions per second for the first experiment (all nodes always on).

according to the laws of Amdhal and Gustafson [62] this limits a global speedup. A more detailed analysis of performance shows that this difference occurs due to the way interrupts from the Ethernet chipset are handled by the Raspberry Pi hardware.

As described in Section IV, the RPi boards were not originally designed to be used as web servers. In its implementation, network hardware interrupts are treated by the CPU. Considering this fact, a test was performed, where several network packages of small sizes were generated, and it was observed that the operating system process that handles network interrupts occupies more than 90 % of the processor capacity in the RPi1. This makes the request processing slower than it should be once the web server processes are left in background, due to lower priority.

On the RPi2 machine a similar issue occurs, that is, the process that services network interrupts also occupies the CPU more than 90% of the time. However, as there are four cores and this occurs only in one of them, the other ones are available to run the web server processes. Even with this, we can conjecture that the performance of these machines is also prejudiced by the way its implementation is done. It concentrates the device interruptions in a single core instead of allowing the operating system to manage this issue. This is known as the SMP affinity at Linux [63]. It can be noticed during experiments that the web server processes occupy little more than 30% of the other cores, system memory is not full, and network throughput is about 50 MBit/s. Thus, network interrupt handling is also a bottleneck for the RPi2 machines performance.

This problem can also be seen when we analyze the quality of service that can be obtained from the plot shown in Figure 8 that brings the average response time for each request. It is expected that, for having a better performance in terms of requests served per second and also for having the overall effort divided between the several machines, the response time of the cluster requests could be smaller. However the results show that the RPi1 gives a response time that is similar

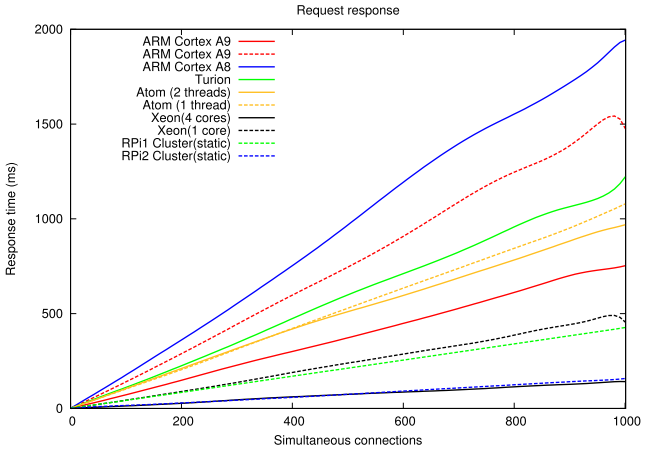


**FIGURE 8.** Average time to complete each transaction during the first experiment (all nodes always on).

to the machine with Xeon with one enabled core, which is about 250 ms. The RPi2 machines response time is similar to the machine with Xeon 4 cores, about 100 ms. Yet, it can be noticed that the number of failed requests is inferior to 0.5% of the total, including when 1000 requests are simultaneously generated.

Even with the above mentioned problems, the plot shown in Figure 9 demonstrates that the energetic performance of the clusters relative to the number of effectively served request per second per each Watt of energy consumed are better. The RPi1 devices have the third better energy performance measure. Next, the machines with ARM Cortex-A9 processors with two cores. Further, the RPi2 cluster is the best one with a performance that is at least twice better than the second one.

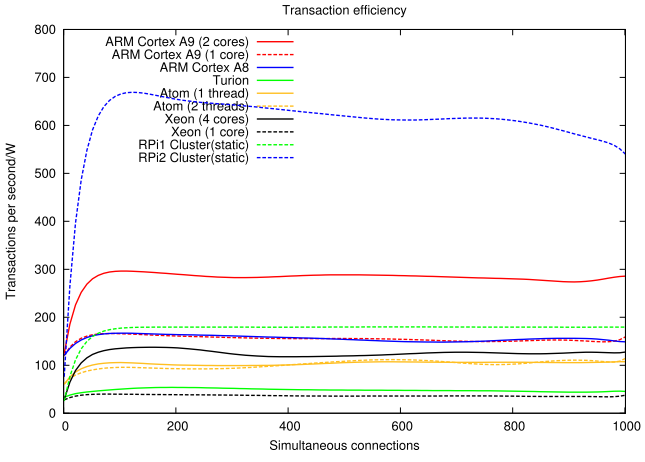


**FIGURE 9.** Power efficiency on successful transactions for the first experiment (all nodes always on).

As it can be seen from the plots, even with the bottleneck caused by the network interface interrupts, the clusters have presented results that are better than a computer with x86 multi-core CPU. This is the computer type generally used as server in the web server infrastructure. Furthermore, the clusters have consumed a low quantity of energy compared to the x86 and, at the same time, obtaining a high number of served requests per second with low latency. It can be noticed from the experiments reported that they get the first and third best results in terms of energetic performance including overcoming other ones based on ARM architecture. So it is empirically verified their capacity of being web servers.

### B. SECOND EXPERIMENT - STATIC VERSUS DYNAMIC CLUSTER

From the first experiment, it is clear that the RPi2 hardware is better than RPi1 for the cluster since its multi-core processor smooths the mentioned network interrupt handling problem, resulting in a better energy efficiency. However, the tests conducted on the first experiment were not energy-proportional. In this sense, it is possible to enhance this aspect through a police for the partial (or proportional) use of the hardware with dynamic provisioning. In this way, the second experiment aims to compare the performance of the RPi2 cluster when the algorithm for dynamic provisioning described in Section IV-A is used. In order to do that, two changes have to be done in the methodology used in the first experiment.

The first change is related to the available web-pages. A typical web server has not only one file in its file system to be served, and the size is not only of 3 KB as assumed in the study done by Aroca and Gonçalves [15]. In order to provide a real-world workload, an analysis performed at the site HTTPArchive.org [64] was conducted. This site collects and permanently stores the web digitized contents. The graphic illustrated at Figure 10 shows that, as of first of July, 2016, a web page has in average 2,046 KB distributed between files of the type HTML, style-sheets, scripts, source codes, videos, images, and others. In this way, in order to run this experiment, we created a repository of files with different sizes, names, types, and paths relative to the default URL.

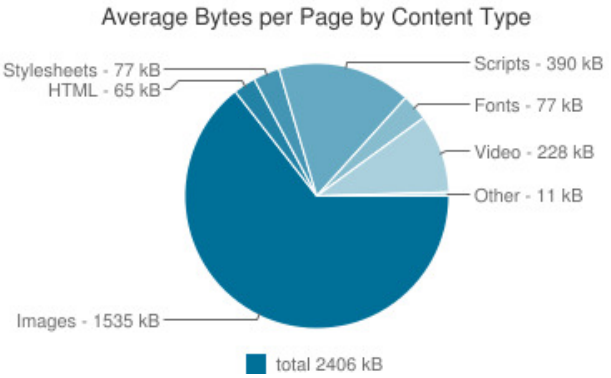


**FIGURE 10.** Average bytes per page by content type on 2016, July 1st (Source: HTTPArchive [64]).

In order to make the tests closer to real world scenarios, we use a methodology based on the work of Urdaneta *et al.* [65]. In that work, the authors collected about 10% of all the accesses of WikiMedia Foundation site.

All of the languages were sampled during the period between September 19th 2007 and January 2nd 2008, and they used a database dump and files to recreate the same environment and to test their benchmark software, the Wikibench.

Due to some complications brought by Wikipedia system updating, availability of the dumps of the registered data access and the huge size of the files and database we decided to simplify our approach as follows: First, we arbitrarily choose a subset of sequential access registers to pages that are present at the log files that could be more available (october 2007). In addition, this is done in such a way that the contents to be downloaded could fit in the memory cards of the RPi machines. Thereafter, all the selected register files are downloaded from Wikipedia site through a script and copied to the memory cards of all nodes. Thus, a request to a static file could be attended by any cluster node since they have the same contents. Data related to the characteristics of these contents are present at Table 3.

**TABLE 3. File set characteristics for the static web content prepared for the second experiment.**

| File type | % of total | Average size |
|---|---|---|
| Audio/Video | 0.02% | 1,443 KB |
| HTML page | 54.16% | 123 KB |
| Image | 45.22% | 23 KB |
| Other | 0,60% | 215 KB |
| **Number of files** | | 153,897 |
| **Disk usage** | | 11.63 GB |

The second change with respect to the first experiment is related to the benchmark software used to measure performance. The Apache Benchmark does not have support to multiple URL (at least until the date when the experiments where done), which is necessary in this new context. We use then a new software, the Siege [66], which allows to configure the number of simultaneous connections and produces similar statistics as the Apache Benchmark, as the number of successful transactions, throughput, and average response time, among others. In addition, it allows the benchmark to be done with a set of URL defined in a file and also that the test could be applied by a determined period of time without limits on the number of requests.

To our tests, we created a file containing about 480 thousand requests from the access log files of the Wikipedia. In those, the field relative to the server domain was substituted by the intern DNS server address. This server uses the round-robin algorithm to balance the load between the attributed addresses. Initially, the cluster is turned on with two active nodes, a manager and one for attending the requests ($min\_nodes = 1$). This last node is commissioned with all addresses. As there is an increase in demand, a new node is turned on and the addresses are divided with the new nodes as shown in Figure 2.

Note that the cluster can have up to 6 nodes acting as servers. So, to get a better load distribution, we attribute a set of 12 possible IP addresses. With 2 active nodes, each one answers to 6 addresses, and, so on, down to 2 addresses (with 6 active nodes). Nonetheless, it can be noted that with 5 active nodes as servers there occurs a disparity in the number of requests sent by the DNS to each node, as 1, 2, 3, 4, and 6 are divisors of 12. In this way, the cluster and DNS servers are independent, being only necessary that the manager node has the list of addresses available for the round-robin. It will distribute these addresses between the nodes that it would judge are necessary to keep on according to the cluster current state. The manager gets the current state of the cluster by looking at the number of requisitions (connections) to the web server process in each activated node. The load is given by the expected average delay $t_{delay}$ according to Equation 5 of Section IV-B.

Assuming that the traffic characteristics is unknown by the server, the average file size for the component $t_{response}$ ($averagepage$) is given by the rate obtained by dividing the total transferred size and the number of requests per web page using the statistics given by the HttpArchive.org [64]. The value obtained (as of February 2016) is $2,253\ KB/100\ files$ thus resulting in 23,070 bytes per file in average. Through previous tests that are partially shown in the graphic of Figure 4 we get the values of $t_{response}(\varnothing)$ and $t_{perbyte}$ in such a way that $t_{response}(averagefile) = 1.47\ ms$. With this, the average delay for obtaining each file would be $t_{delay} = 1.47\times \mid queuedrequests \mid$.

In order to serve the requests of a complete web page, the total time depends on the $tdelay$ of each file plus the time taken to process each file. In a favorable scenario, the requests distributed in parallel to more than a cluster node. And, it could happen, in a single node, requisitions of other files to the same page. However, in general a client gets access to the DNS server only once and all requisitions are sent to the same address. In an overloaded scenario it could happen that no other requisitions in the list are for files of the same page thus accumulating delays for the page requisition. In other words, if a page is composed of 100 files, the total waiting time would be the sum of all $t_{delay}$.

We could not observe in the literature any standard about what can be considered an acceptable time for waiting the downloading of a web page. To this end we came up that even the number of files can and their sizes vary from year to year. Thus it is a good idea to let the network administrator to determine these parameters for some desired quality of service. With this in mind, we take the time of 5s as acceptable for a page to be completely downloaded. This means that each one of the 100 files that compose a page, in average, could have the maximum delay of 50 ms in the worst case. In the same way, we consider that a time above 10 seconds (100ms per file) is undesirable thus coming up with the thresholds $t_{high} = 50ms$ and $t_{critical} = 100ms$.

In order to complete the algorithm parameters with the goal of establishing a minimum time for detecting overloading of under-utilization we define from the tests the verification interval $sleep\_interval = 5s$. Also, the number of repetitions

of a same state in order for the controller to act are set as for $r_{low} = 6$, $r_{high} = 6$ and $r_{critical} = 3$. These values are chosen in such a way that the cluster has be in the critical state during 30s or in the overloaded during 60s. In these cases a new node is made active. Also, it has to present the sub-utilization state for at least 30s in order for, in this case, a node to become inactive. Table 4 provides a synthesis of the parameters for the algorithm of dynamic provisioning.

**TABLE 4. Dynamic provisioning algorithm parameters.**

| Parameter | Value |
|---|---|
| $Nodes$ | $RPi_1, RPi_2, \ldots, RPi_6$ |
| $Addresses$ | $192.168.0.1 - 12$ |
| Tresholds | $t_{high} = 50ms, t_{critical} = 100ms$ |
| Recurrences | $r_{low} = 6, r_{high} = 6, r_{critical} = 3$ |
| $sleep\_interval$ | 5 seconds |
| $min\_nodes$ | 1 node (as server) |

Finally, about the number of concurrent connections to be used during the test of the cluster, we noticed that the main web browsers establish from 6 to 8 simultaneous connections with the server when files are requested. Thus, the benchmark tests start with 8 simultaneous requests simulating a browser, using another independent machine connected to the network. At each test, this request number is increased by 24 thus representing 3 new clients and it scales up to 488, that corresponds to 61 total clients, simultaneously. As it is necessary some time to turn on and off the nodes according to demand, each test is performed during 10 minutes and the results presented in Figure 11 represent the mean for 10 consecutive executions. At the end of each run, and previous to starting a new execution, there is a waiting time for the system to get to its initial state with only one node operating.

Figure 11 shows the evolution of energy consumption with time, during 10 minutes of each execution. Each line represents a different scenario, that is, with a different number of simultaneous connections configured. The graph includes delimiters to relate the number of nodes, allowing it to represent the energy consumption in terms of the number of active nodes. It can be noticed that with the increase in demand the bigger is the number of active nodes and that this happens earlier. From 200 simultaneous connections on, the cluster turns all of the 6 nodes active with the last one turned on with about 300s. Thus at each minute a new node gets active. Two interesting conditions could be noted, separated in two figures for better understanding (next).

Figure 12 depicts some situations where the cluster starts making new nodes active until they get a use level considered normal thus stabilizing. Any new node is made active from this moment and consumption keeps the same until the end of this run.

In Figure 13 some deactivation occurs, that is, new nodes are made active until a certain condition is detected. At this moment it is detected that a node is made active without enough demand for justifying this. This occurs because when

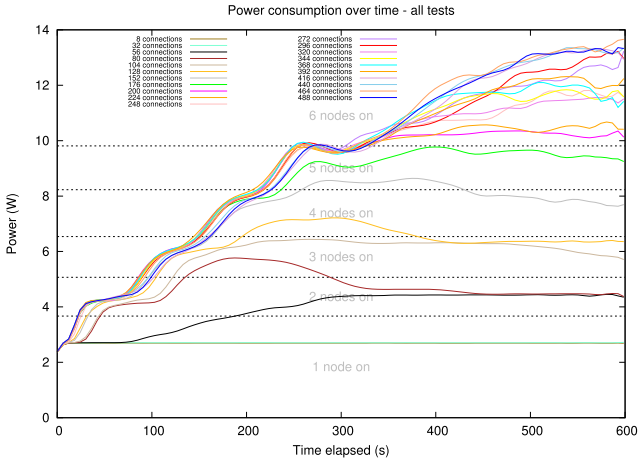


**FIGURE 11. Power consumption over time for the proposed dynamic provisioning system with different demands.**

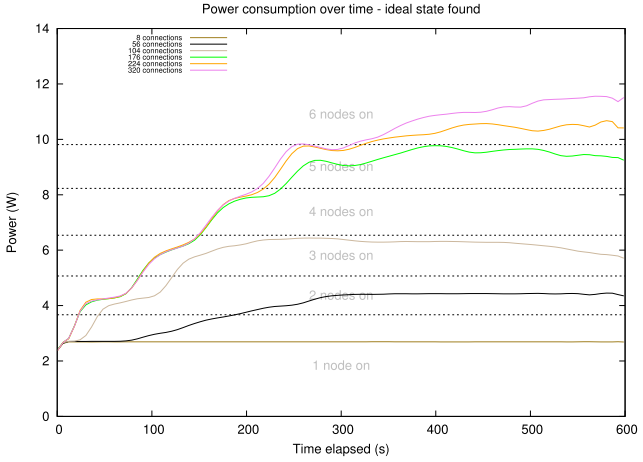


**FIGURE 12. Demand scenarios where dynamic provisioning algorithm finds a ideal number of nodes.**

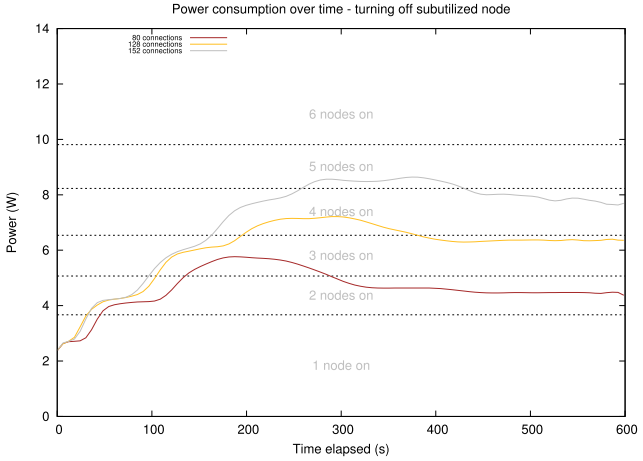


**FIGURE 13. Demand scenarios where nodes are turned off when cluster is considered underutilized.**

the cluster is overloaded there is an excessive number of requisitions to be attended at the list, however, as new nodes are turned on these requisitions are processed and the demand is

not sustained anymore. After turning off one of the machines the cluster has found its equilibrium state and this current configuration has stabilized. Some oscillation is expected during the node on/off transitions, as the boot process consumes more power than the system operation after the boot.

Figure 14 shows a comparison of average power consumption during 10 minutes for the static and dynamic clusters. It can be noticed that the static cluster has an average consumption of about 13W with a little variation in the tests with less demand since the hosts work with a certain clearance. In the dynamic cluster the average is smaller. In the first scenarios with very low demand its value is still smaller once there is no need of turning on all of the nodes. Each mark in the line means that a new node is made active in relation to the previous one for that scenario. The first line segment that has an ascent curve representing not only an increasing in the machine use as well the new nodes that are made active more fast. This is because as higher the demand gets easier the cluster enters in the critical situation and requires the provisioning in a faster way.

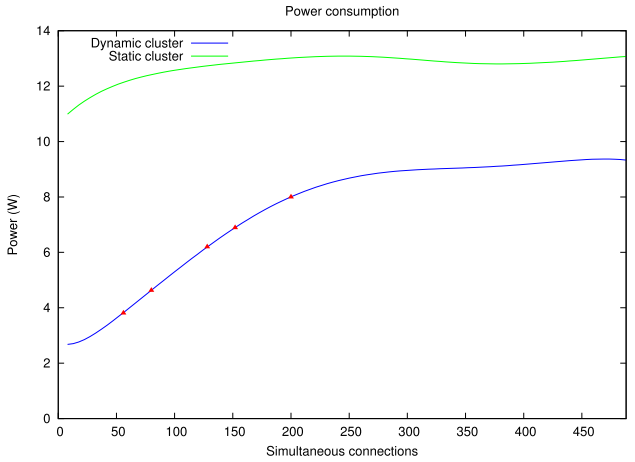


**FIGURE 14.** Average power consumption comparing the static and dynamic provisioning approach for different demands.

This capability of offering scalability according to demand can be noted in Figure 15, which shows a plot of the response time versus the number of requests. The curves that represent the static and dynamic behaviors are somewhat similar. Nonetheless, the response time of the static cluster is smaller once all of the nodes are initially on and ready to receive requisitions. In the dynamic cluster the nodes are activated according to demand not allowing the uncontrolled growing of latency. The values obtained for response time are above the desired ones between 50 and 100ms coming to some 400ms in the case of high demand. However, it can be noted that the requested files sizes as shown in Table 3 are bigger than the mean size defined as 23KB for the algorithm calculations. It can yet be noted that with time the cluster manager could recalculate these values according to status data given by the web servers and better adjust its perception about the nodes overload.

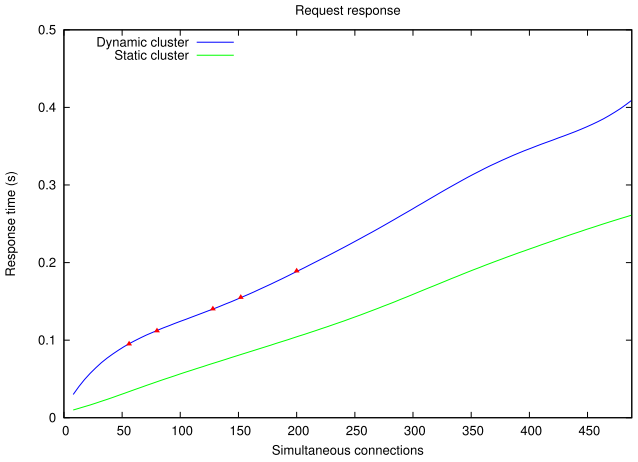


**FIGURE 15.** Average time to complete each web file transfer transaction for static and dynamic provisioning approach.

Regarding the performance, the static cluster has always a higher number of served requests per second, as expected, and this is shown in Figure 16. This difference is higher in the beginning once it is ready to giving its better performance. However, in the scenarios with higher demand, in which the nodes of the cluster are activated faster, this difference becomes gradually reduced because the performance of the static cluster stays stable while the dynamic cluster one enhances.

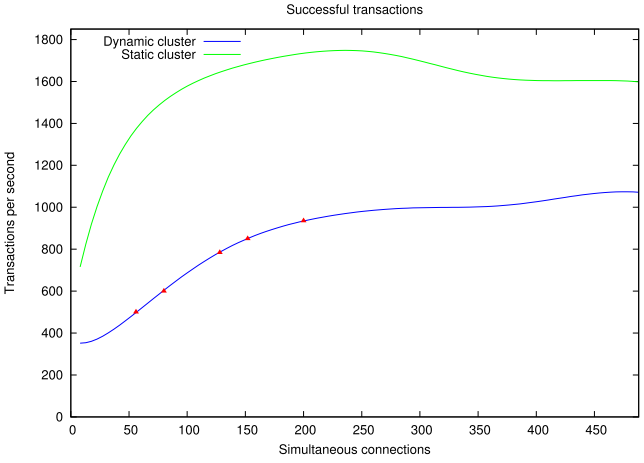


**FIGURE 16.** Successful web file transfer transactions per second for static and dynamic provisioning approach.

Figure 17 shows the values for the energy efficiency for the two methodologies. Here the used metric is the number of requisitions attended per second by each unity of energy consumed. As it was expected, in the scenarios where demand is smaller the energetic efficiency is lower for the static approach. This is due to the under utilization of the cluster that keeps all of its machines on without necessity. On the other side, when there is an increase in demand, its energetic efficiency also increases and keeps stable because it reaches its maximum capacity for requests processing.

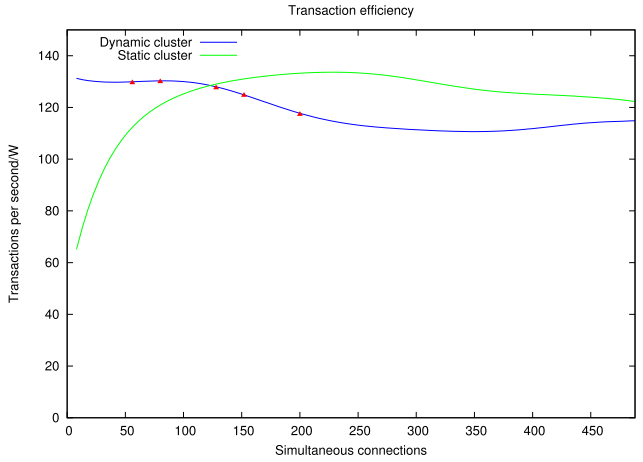


**FIGURE 17.** Power efficiency on successful web file transfer transactions for static and dynamic provisioning approach.

In the approach with dynamic provisioning, the behavior is the opposite. The energetic performance begins with better values just because it keeps off the nodes are not necessary for attending the current demand. However, with an increasing in the number of connections, the energetic efficiency decreases until it gets smaller than the static cluster. This indicates that, for that scenario, it is necessary a maximum of performance of the cluster what occurs some 300s later as seen in Figure 11. At higher demands where a faster provisioning is performed, the energetic efficiency values approximate to each other tending to about 120 requisitions/second per power unity (Watt).

It can be noted that while the energetic efficiency of the static cluster may vary considerably (twice) depending on demand, the values for the dynamic cluster present approximately 10% of variability. For an IT infrastructure administrator this means a trustworthy and predictable approach for estimating the costs and energy consumption since the energetic efficiency keeps close values independent of demand.

## VI. CONCLUSION

The main contribution of this work is a generic cluster architecture with a dynamic provisioning algorithm. It controls the number of active nodes according to the demand in real time to obtain energy consumption proportional to the demand, while keeping acceptable quality of service and even high availability. In order to validate the proposal, a real prototype was built and is now fully functional. Moreover, in order to be classified as a green computing cluster we provide, also as a contribution of this work, several analyses of the obtained performance. In the work, the NPi-Cluster is empirically proven to have performance similar or even better than conventional computers used to this same goal. For example, we have shown that the efficiency can be compared to the x86 architecture with multi-core Xeon machines, however with a substantially reduced energy consumption. Thus, the proposed cluster solution has demonstrate to have an energetic efficiency that is superior to the others presented.

It could be shown also, through empirical tests that the policy for dynamic provisioning provides an increase in energy efficiency, mostly in the cases in which there is a low demand for requisitions. In these situations, the cluster keeps nodes that are not necessary powered off for the current processing. For other scenarios in which the processing demand is higher the cluster manager makes new nodes active in a manner that is as fast as necessary to effectively serve the requests. This also demonstrates that the methodology used for measuring overloading and sub-utilization of the cluster nodes is able to detect when these situations happen.

Besides the advantages above presented, the proposed system also generates less heating. This reduces or even eliminates the needs of cooling systems. The use of physical space is also considerably reduced since in a cluster about 7 nodes can be embedded in a space that is smaller than a standard desktop computer, or simply mounted on the wall, as the system is built on a frame. Furthermore, it is not necessary much more space and cooling to provide the cluster scalability. And, for its operation it would be enough to exchange the set of nodes and addresses that are in the algorithm parameters.

Besides these promising results we can conjecture that in futures versions of hardware the Raspberry Pi could have a better result than the ones obtained here with these current experiments. We could notice that a small however not compromising degradation in performance occurs mainly due to the treatment of network interrupts. They, more recurrent in the case of the using the cluster nodes as web server, are treated by the main processor thus occupying almost all of its capacity. This problem makes the network use the bottleneck in order for this proposal to work properly. However, this problem gets attenuated in the RPi2 model, since it has a multicore processor. That is, when using the RPi2, we could notice that, while one of the cores gets responsible for providing the network connections, the other ones could perform other tasks.

On 2016, the Raspberry Pi Foundation launched the 3rd generation of their boards, bringing improvements in processor (64-bit), built-in wireless network (Wifi) and Bluetooth connectivity [67]. Some experimental tests were performed with this new Raspberry PI model (3), with the hope that the interrupt problem could have been solved, but the same interrupt treatment issue observed on Raspberry PI 1 and 2, persists on Raspberry PI 3. Keeping almost the same energy consumption, RPI 3 managed to improve the performance of RPi2 by about 17 %, primarily due the processor upgrade. But likewise, one of the cores is still occupied with network device interrupts and continues to be a bottleneck for its operation. For this reason we do not repeat the tests with a cluster RPi3 because the results would show the same behavior.

Furthermore, an interesting aspect of NPi-Cluster architecture is that it can be applied without changes for several different types of services and machines, since the form of cluster overload determination is generic. Besides the low cost, the proposed architecture also provides flexibility on its parameters, scalability for future expansion and proportional-

ity between computing and energy use, and does not represent bottleneck for cluster management.

Despite all advantages, the proposed architecture has some limitations. The dynamic provisioning algorithm needs a consistent detection method to determine when a node is overloaded or underutilized. This work proposes a static approach based on the size of webserver requests queue and the average file size. Other techniques could be applied by using statistics, estimates, software agents, prediction algorithms, or artificial intelligence. There are also issues related to the hardware itself that may not meet applications constraints. The energy efficiency of these boards is achieved through limitations regarding the amount of memory and bus speed, for example, in addition to software availability for ARM architecture the bottleneck resulting from the network interruptions discussed here. In future works we will apply NPi-Cluster to more dynamic scenarios, such as dynamic web applications and variable demand, in addition to other types of applications.

Finally, to this end, the system prototype is not only fully finished but up and running at SÃčo Carlos Federal University, where it is physically set. This is some 3 thousand kilometers from Natal, where almost all the software architecture has been developed at the Natalnet Laboratory. As a final remark, and just to close, we all agree that this collaboration has allowed a great team job.

## REFERENCES

[1] O. Svanfeldt-Winter, S. Lafond, and J. Lilius, "Cost and energy reduction evaluation for ARM based web servers," in *Proc. IEEE 9th Int. Conf. Dependable, Auton. Secure Comput. (DASC)*, Dec. 2011, pp. 480–487.

[2] J. Fitzpatrick, "An interview with Steve Furber," *Commun. ACM*, vol. 54, no. 5, pp. 34–39, May 2011.

[3] M. H. Paek, "An analytical framework and promotion for Green IT strategy," in *Proc. Int. Conf. Inf. Commun. Technol. Convergence (ICTC)*, Oct. 2014, pp. 585–592.

[4] S. Agarwal and A. Nath, "Green computing—A new horizon of energy efficiency and electronic waste minimization: A global perspective," in *Proc. Int. Conf. Commun. Syst. Netw. Technol. (CSNT)*, Jun. 2011, pp. 688–693.

[5] S. K. Srivastava, "Green supply-chain management: A state-of-the-art literature review," *Int. J. Manage. Rev.*, vol. 9, no. 1, pp. 53–80, 2007.

[6] R. Harmon and H. Demirkan, "The next wave of sustainable IT," *IT Prof.*, vol. 13, no. 1, pp. 19–25, Jan. 2011.

[7] *Key World Energy Statistics*, International Energy Agency, Paris, France, 2016.

[8] STAMFORD. (2007). *Gartner Estimates ICT Industry Accounts for 2 Percent of Global $CO_2$ Emissions*. [Online]. Available: http://www.gartner.com/newsroom/id/503867

[9] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors," *IEEE Signal Process. Mag.*, vol. 26, no. 6, pp. 26–37, Nov. 2009.

[10] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.

[11] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *Int. J. Comput. Aided Eng. Technol.*, vol. 6, no. 4, pp. 440–459, 2014.

[12] A. Bianzino, C. Chaudet, D. Rossi, and J. Rougier, "A survey of green networking research," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 1, pp. 3–20, 1st Quart., 2012.

[13] T. Kaur and I. Chana, "Energy efficiency techniques in cloud computing: A survey and taxonomy," *ACM Comput. Surveys*, vol. 48, no. 2, pp. 22:1–22:46, Oct. 2015.

[14] G. Chen *et al.*, "Energy-aware server provisioning and load dispatching for connection-intensive Internet services," in *Proc. 5th USENIX Symp. Netw. Syst. Design Implement.*, 2008, pp. 337–350. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=76111

[15] R. V. Aroca and L. M. G. Gonçalves, "Towards green data centers: A comparison of x86 and ARM architectures power efficiency," *J. Parallel Distrib. Comput.*, vol. 72, no. 12, pp. 1770–1780, Dec. 2012.

[16] E. Brown. (2011). *Sandia's Mini Supercomputer Runs Linux on 196 Gumstix Arm Modules*, Accessed on May 2017. [Online]. Available: http://linuxdevices.linuxgizmos.com/sandias-mini-supercomputer-runs-linux-on-196-gumstix-arm-modules/

[17] K. Fürlinger, C. Klausecker, and D. Kranzlmüller, "Towards energy efficient parallel computing on consumer electronic devices," in *Information and Communication on Technology for the Fight against Global Warming: First International Conference, ICT-GLOW*, D. Kranzlmüller and A. M. Toja, Eds. Berlin, Germany: Springer, 2011, pp. 1–9.

[18] Mont Blanc. (2011). *Mont Blanc Project*, accessed on May 2017. [Online]. Available: http://www.montblanc-project.eu/arm-based-platforms/

[19] K. L. Keville, R. Garg, D. J. Yates, K. Arya, and G. Cooperman, "Towards fault-tolerant energy-efficient high performance computing in the cloud," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2012, pp. 622–626.

[20] N. Balakrishnan, "Building and benchmarking a low power ARM cluster," M.S. thesis, Dept. Edinburgh Parallel Comput. Centre (EPCC), Univ. Edinburgh, Edinburgh, Scotland, 2012.

[21] N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, and A. Ramirez, "The low power architecture approach towards exascale computing," *J. Comput. Sci.*, vol. 4, no. 6, pp. 439–443, 2013.

[22] A. M. Pfalzgraf and J. A. Driscoll, "A low-cost computer cluster for high-performance computing education," in *Proc. IEEE Int. Conf. Electro/Inf. Technol.*, Jun. 2014, pp. 362–366.

[23] S. J. Cox, J. T. Cox, R. P. Boardman, S. J. Johnston, M. Scott, and N. S. O'Brien, "Iridis-pi: A low-cost, compact demonstration cluster," *Cluster Comput.*, vol. 17, no. 2, pp. 349–358, 2014.

[24] Z. Krpić, G. Horvat, D. Zagar, and G. Martinović, "Towards an energy efficient SoC computing cluster," in *Proc. 37th Int. Conv. Inf. Commun. Technol., Electron. Microelectronincs (MIPRO)*, May 2014, pp. 178–182.

[25] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: Past, present and future," *Concurrency Comput., Pract. Exper.*, vol. 15, no. 9, pp. 803–820, 2003.

[26] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. (2017). *HPL—A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*, accessed on May 2017. [Online]. Available: http://www.netlib.org/benchmark/hpl/

[27] M. Humphries. (2011). *Canonical Builds a 42-Core Arm Cluster Server Box for Ubuntu*, accessed on Sep. 2016. [Online]. Available: http://www.geek.com/chips/canonical-builds-a-42-core-arm-cluster-server-box-for-ubuntu-1390095/

[28] Y. Zhao *et al.*, "An experimental evaluation of datacenter workloads on low-power embedded micro servers," in *Proc. VLDB Endowment*, vol. 9, no. 9, pp. 696–707, May 2016.

[29] G. D. Costa, "Heterogeneity: The key to achieve power-proportional computing," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGrid)*, May 2013, pp. 656–662.

[30] D. Loghin, B. M. Tudor, H. Zhang, B. C. Ooi, and Y. M. Teo, "A performance study of big data on small nodes," *Proc. VLDB Endowment*, vol. 8, no. 7, pp. 762–773, Feb. 2015.

[31] P. Abrahamsson *et al.*, "Affordable and energy-efficient cloud computing clusters: The Bolzano Raspberry Pi cloud cluster experiment," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, vol. 2. Dec. 2013, pp. 170–175.

[32] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros, "The Glasgow Raspberry Pi cloud: A scale model for cloud computing infrastructures," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst. Workshops*, Jul. 2013, pp. 108–112.

[33] K. Fox, W. M. Mongan, and J. Popyack, "Raspberry HadooPI: A low-cost, hands-on laboratory in big data and analytics," in *Proc. SIGCSE*, 2015, p. 687.

[34] C. Kaewkasi and W. Srisuruk, "A study of big data processing constraints on a low-power Hadoop cluster," in *Proc. Int. Comput. Sci. Eng. Conf. (ICSEC)*, Jul. 2014, pp. 267–272.

[35] N. Schot, "Feasibility of raspberry Pi 2 based micro data centers in big data applications," in *Proc. 23th Univ. Twente Student Conf. IT*, Enschede, The Netherlands, 2015, p. 22.

[36] S. S. D. Xu and T. C. Chang, "A feasible architecture for ARM-based microserver systems considering energy efficiency," *IEEE Access*, vol. 5, pp. 4611–4620, 2017.

[37] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10.

[38] P. Velthuis, "Small data center using raspberry Pi 2 for video streaming," in *Proc. 23th Twente Student Conf. IT*, 2015. [Online]. Available: http://referaat.cs.utwente.nl/conference/23/paper/7515/small-data-center-using-raspberry-pi-2-for-video-streaming.pdf

[39] Z. Ou, B. Pang, Y. Deng, J. K. Nurminen, A. YläJääski, and P. Hui, "Energy- and cost-efficiency analysis of ARM-Based clusters," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGrid)*, May 2012, pp. 115–123.

[40] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: A fast array of wimpy nodes," in *Proc. ACM SIGOPS 22nd Symp. Oper. Syst. Principles*, 2009, pp. 1–14.

[41] D. Schall and V. Hudlet, "WattDB: An energy-proportional cluster of wimpy nodes," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 1229–1232.

[42] J. Xie, P. Jin, S. Wan, and L. Yue, *Energy-Proportional Query Processing on Database Clusters*. Cham, Switzerland: Springer, 2015, pp. 324–336.

[43] B. Heller *et al.*, "ElasticTree: Saving energy in data center networks," in *Proc. NSDI*, vol. 10, 2010, pp. 249–264.

[44] M. F. Cloutier, C. Paradis, and V. M. Weaver, "Design and analysis of a 32-bit embedded high-performance cluster optimized for energy and performance," in *Proc. 1st Int. Workshop Hardware-Softw. Co-Design High Perform. Comput.*, 2014, pp. 1–8.

[45] M. F. Cloutier, C. Paradis, and V. M. Weaver, "A raspberry Pi cluster instrumented for fine-grained power measurement," *Electronics*, vol. 5, no. 4, p. 61, 2016. [Online]. Available: http://www.mdpi.com/2079-9292/5/4/61

[46] (2017). *Raspberry Pi Project Home Page*, accessed on May 2017. [Online]. Available: https://www.raspberrypi.org/

[47] F. Kaup, P. Gottschling, and D. Hausheer, "PowerPi: Measuring and modeling the power consumption of the raspberry Pi," in *Proc. 39th Annu. IEEE Conf. Local Comput. Netw.*, Sep. 2014, pp. 236–243.

[48] E. C. Joseph and S. Conway. (2011). *Heterogeneous Computing: A New Paradigm for the Exascale Era (Adapted From IDC HPC End-User Study of Processor and Accelerator Trends in Technical Computing*, accessed on May 2017. [Online]. Available: http://blogs.nvidia.com/wp-content/uploads/2011/11/IDC-Exascale-Executive-Brief_Nov2011.pdf

[49] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 585–599, Feb. 2016.

[50] V. Cardellini, M. Colajanni, and P. S. Yu, "Dynamic load balancing on Web-server systems," *IEEE Internet Comput.*, vol. 3, no. 3, pp. 28–39, May 1999.

[51] (2016). *Apache Module Mod_Proxy_Balancer*, accessed on May 2017. [Online]. Available: https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html

[52] (2016). *HAProxy the Reliable, High Performance TCP/HTTP Load Balancer*, accessed on May 2017. [Online]. Available: http://www.haproxy.org/

[53] (2016). *Pacemaker: A Scalable High Availability Cluster Resource Manager*, accessed on May 2017. [Online]. Available: http://clusterlabs.org/

[54] T. P. Brisco, "DNS support for load balancing," in *Proc. RFC*, Mar. 2013, p. 1794. [Online]. Available: https://rfc-editor.org/rfc/rfc1794.txt

[55] (2017). *Cisco IOS IP Configuration Guide: Configuring Server Load Balancing*. accessed on May 2017. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/ios/12_2/ip/configuration/guide/fipr_c/1cfsflb.html

[56] X. Zheng and Y. Cai, "Achieving energy proportionality in server clusters," *Int. J. Comput. Netw.*, vol. 1, no. 1, pp. 21–35, 2009.

[57] J. Strawson and N. Ayres. (Nov. 2012) *Adventures in Retail: The Other Line's Moving Faster*. Accessed May 2017. http://img01.thedrum.com/s3fs-public/drum_basic_article/99200/additional_media/online-retail-research-report-november-2012.pdf

[58] J. Dilley, R. Friedrich, T. Jin, and J. Rolia, "Web server performance measurement and modeling techniques," *Perform. Eval.*, vol. 33, no. 1, pp. 5–26, Jun. 1998.

[59] (2017). *Nginx—High Performance Load Balancer, Web Server & Reverse Proxy*, accessed on May 2017. [Online]. Available: https://www.nginx.com/

[60] (2017). *Raspbian—Raspberry Pi Foundation's Official Supported Operating System*, accessed on May 2017. [Online]. Available: https://www.raspberrypi.org/downloads/raspbian/

[61] *Ab—Apache HTTP Server Benchmarking Tool*, accessed on May 2017. https://httpd.apache.org/docs/2.4/programs/ab.html

[62] J. L. Gustafson, "Reevaluating Amdahl's Law," *Commun. ACM*, vol. 31, no. 5, pp. 532–533, May 1988.

[63] R. Love, "Introducing the 2.6 kernel," *Linux J.*, vol. 23, no. 109, p. 2, May 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=770650.770652

[64] (2017). *HTTP Archive*, accessed on May 2017. [Online]. Available: http://httparchive.org

[65] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Elsevier Comput. Netw.*, vol. 53, no. 11, pp. 1830–1845, Jul. 2009. [Online]. Available: http://www.globule.org/publi/WWADH_comnet2009.html

[66] (2017). *Siege HTTP Load Testing and Benchmarking Utility*, accessed on May 2017. http://www.joedog.org/siege-home/

[67] *Raspberry Pi 3 Model b*, accessed on May 2017. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

**SEBASTIÃO EMIDIO ALVES FILHO** received the degree in computer sciences from Federal University of Rio Grande do Norte (UFRN), Brazil, and the M.Sc. degree in computer science from State University of Rio Grande do Norte (UERN), Brazil. He is currently pursuing the Ph.D. degree with the Electrical and Computing Engineering Graduate Program at UFRN. Since 2004, he has had experience in IT systems, combinatorial optimization, free software, server administration, and cluster/grid/cloud computing. He is also an Adjunct Professor with the Informatics Department, UERN. His main research interests are in distributed computing and green IT.

**AQUILES MEDEIROS FILGUEIRA BURLAMAQUI** received the degree in computer sciences and the M.Sc. degree in systems and computing from Federal University of Rio Grande do Norte (UFRN) and the Ph.D. degree in electrical and computing engineering. He is currently a Professor with UFRN and also a Researcher with NatalNet Laboratories, with interest in several fields, including multimedia, software engineering, digital TV, and educational robotics.

**RAFAEL VIDAL AROCA** (M'05) received the degree in informatics and the M.Sc. degree in mechatronics engineering from University of Sao Paulo, and the Ph.D. degree in electrical and computing engineering. He has over ten years of industry experience in embedded systems, IT systems, and servers administration. He is currently an Adjunct Professor with Federal University of Sao Carlos. His main research interests are in embedded systems, operating systems, and robotics.

**LUIZ MARCOS GARCIA GONÇALVES** received the Ph.D. degree in systems and computing engineering from Federal University of Rio de Janeiro in 1999. He is currently an Associate Professor with the Computing Engineering and Automation Department, Federal University of Rio Grande do Norte, Brazil. His research interests are in computer vision, robotics, and all aspects of graphics processing. He has been a member of the IEEE Latin American Robotics Council since 2002. He was the Chair of the Brazilian Committee on Robotics and on Computer Graphics and Image Processing, both under the Brazilian Computer Society.