Greening Web Servers: A Case for Ultra Low-power Web Servers

Benoy Varghese*†, Niklas Carlsson‡, Guillaume Jourjon*, Anirban Mahanti* and Prashant Shenoy§
*NICTA, Australian Technology Park, Eveleigh, NSW Australia. Email: firstname.lastname@nicta.com.au
†University of New South Wales, Sydney, NSW Australia
‡Linkoping University, Sweden, Email: Niklas.Carlsson@liu.se
§University of Massachusetts, Amherst, MA, USA; Email: shenoy@cs.umass.edu

***Abstract*—This paper studies the feasibility and benefits of greening Web servers by using ultra-low-power micro-computing boards to serve Web content. Our study focuses on the tradeoff between power and performance in such systems. Our premise is that low-power computing platforms can provide adequate performance for low-volume Websites run by small businesses or groups, while delivering a significantly higher request per Watt. We use the popular Raspberry Pi platform as an example low-power computing platform and experimentally evaluate our hypothesis for static and dynamic Web content served using this platform. Our result show that this platform can provide comparable response times to more capable server-class machines for rates up to 200 requests per second (rps); however, the scalability of the system is reduced to 20 rps for serving more compute-intensive dynamic content. Next, we study the feasibility of using clusters of low-power systems to serve requests for larger Websites. We find that, by utilising low-power multi-server clusters, we can achieve 17x to 23x more requests per Watt than typical tower server systems. Using simulations driven by parameters obtained from our real-world experiments, we also study dynamic multi-server policies that consider the tradeoff between power savings and overhead cost of turning servers on and off.**

I. Introduction

In todays cloud-dominated age, we still find that dedicated Web servers are widely used, especially in many small-scale to medium-scale enterprises [1], [2]. Power consumption of such dedicated Web servers can be high. For instance, a typical tower server has a baseline power consumption of 325 W [3], which amounts to approximately 50-80% of its power consumption at peak load. However, over 90% of the time, these servers are underutilised [4].

This paper explores the feasibility of using ultra low-power systems to replace these high-power Web servers at small-to-medium scale enterprises, motivated by two observations. First, recent advancements in processor designs potentially enable new ARM-based ultra-low-power systems to replace x86 based systems in numerous applications in the near future. Second, many small-to-medium scale organisations (e.g., universities) do not have response time requirements that are as critical or strict as that of commercial online services.

In evaluating our hypothesis, we make three contributions. First, we experimentally evaluate the performance of a *lighttpd* Web server [5] implemented on an ARM-based Raspberry Pi and compare it to that on a standard x86 server. We primarily focus on the tradeoff between energy consumption and response times of these servers. Our results show that low-power computing systems provide comparable response times for serving static Web content and can sustain request rates of up to 200 requests per second (rps), which is more than adequate for small Websites. We find that the scalability of the system is reduced to 20 rps when serving more compute-intensive dynamic Web content, which makes such systems less desirable for such settings. In both scenarios, this performance is achieved at a fraction of the energy consumed by a standard server.

Second, we show that a Raspberry Pi cluster can serve, on average, 17x to 23x more requests per Watt than a typical tower server at high arrival rates. Third, motivated by these results, we consider the feasibility of using clusters of low-power servers to host more popular Websites and consider dynamic multi-server configurations in which the number of actively serving servers is adjusted based on the current workload such as to achieve a minimum desired response time. Using system parameters determined based on our experiments on real hardware, our simulations of the system evaluate the clustered servers under three basic policy classes, each designed to allow a group of servers to save energy when not in use.

The simulations show that our conclusions hold true across all policy classes. Overall, systems using multiple low-power servers consistently use much less energy, while achieving the same desired performance level as those using multiple standard servers. We also find that a policy that has a minimum number of always-on servers typically achieves a more desirable tradeoff between the number of on/off server transitions and energy usage, than a policy in which servers defer turning off after their queues empty. These policies are therefore preferable in modern systems, where there typically is a wear-and-tear cost associated with turning servers on-and-off. Overall, our study shows that low-power systems are a greener alternative to standard servers.

This paper is organised as follows. Section II discusses related work. The experimental methodology is outlined in Section III. Sections IV shows the results of the experimental evaluation and analysis. Section V presents both an average delay-based multi-server analysis and a dynamic multi-server analysis using fine grained policies for when to turn servers on-and-off such as to further save energy. Section VI presents concluding remarks, including future work directions.

This work was supported by funding from NICTA and the Australian Research Council through the ICT Centre of Excellence program, CENIIT at Linköping University, and NSF Grants CNS-1117221 and CNS-1422245.
978-1-4799-6177-1/14/$31.00 ©2014 IEEE

II. Related work

The design and performance evaluation of Web servers has attracted much attention from in the last decade. The primary focus was to achieve highest possible performance [6], [7]. Power efficiency was generally not considered. Recently, Hashemian *et al.* [8] studied the effect of scaling on performance of multi-core Web servers for both static and dynamic content. Although the impact on power consumption was not considered, their findings show that the performance of dynamic workloads scale better with number of cores than for static workloads, suggesting that multi-core servers may be better suited for serving dynamic than static workloads.

The performance of typical Web servers that involve transaction processing can be analysed using various TPC benchmarks. TPC-W was one of the earliest benchmarks used for e-commerce Websites [9] and was superseded by TPC-H. TPC-Energy specifies the rules and methodologies to be followed for measuring and reporting energy metrics in TPC Benchmarks. Another standard benchmarking suite used for server benchmarking was the SPECWeb [10]. For benchmarking the power and performance of server class computers, SPECPowerssj_2008 is used [11]. More recently, Bai *et al.* [12] benchmarked wireless Web servers for applications where the lifetime of clients are short-lived. They observed that throughput can be drastically improved by using persistent connections. They also demonstrate numerous ways in which bottlenecks are manifested in a wireless Web server.

Amza *et al.* [13] modelled and benchmarked Websites with various types of dynamic content. Their evaluation identified that bottlenecks depend on the type of content delivered and highlighted the importance of characterising the performance of a Web server depending on its application.

The use of clusters of low-power systems for computing has comparatively received less attention. Andersen *et al.* [14] developed a key-value-based storage mechanism to reduce power consumption in processing large amounts of data using a cluster of low-power systems. Although their primary focus was the design of the storage system rather than benchmarking the performance of the low-power cluster for data intensive computing, they showed that high speed data processing is possible using clusters of low-power devices.

More recently, Sehgal *et al.* [15] studies how file system design affects server performance and energy efficiency. They compared the performance of various file systems on two different server class machines under various workloads. They concluded that no single file system can be used universally to achieve energy efficiency, but if the file system is matched to the workload, significant power savings could be achieved.

Mathew *et al.* [16] showed that energy reductions can be maximised while maintaining end user SLA performance by using a load balancer to reduce the number of server on/off transitions in a multi-server architecture like CDNs. They argued that minimising server transitions is an important parameter so as to improve the reliability of server hardware. In our paper, we show that we are able to achieve low server transitions while achieving significant power reductions without violating SLA thresholds.

Fig. 1. Raspberry Pi micro-computing board.

III. Experimental Methodology

A. System description

Our Web server was implemented on an ARM-based Raspberry Pi, a credit card sized micro-computing board primarily developed for schools to teach programming. The network interface is a 10/100 Ethernet card. It consumes less than 3.5W of power under peak load. We choose Raspberry Pi because of its power consumption, low cost, and availability.

In our experiments we use a high speed, 8 GB class 10 (30 MB/s), SD card for storage. This ensures that the storage device is not the system bottleneck. If additional space is required, we could use a larger SD card, an USB powered high speed SSDs, or spinning drives. Even though addition of peripheral devices would increase the total system power consumption, there are ultra low-power consuming devices available in the market today that cater for power-efficient applications [17]. We note that addition of peripherals would skew the performance analysis because bottlenecks could potentially be created due to the slow speed of USB 2.0 interface.

The Web server implementation is based on *lighttpd* $v.1.4.31$ [5] which has very low resource requirements. Some of the alternatives for Web server implementation were *Apache* and *Nginx*. The advantage of using *lighttpd* over *Apache* is that the former is an asynchronous server, which handles processes in a single thread with non-blocking I/O. The latter is a process-based server, which requires a separate thread to handle simultaneous connections. In essence, the memory footprint for *lighttpd* is much smaller than *Apache* for large workloads. We chose to use *lighttpd* for our experiments, primarily due to its ease of use and its slightly better performance than *Nginx* while handling smaller file sizes.

The Raspberry Pi that we use for our experiments is $Model$ 0002, $Type\ B$, $Rev.$ 1 with 256 MB RAM. We have a similar implementation on a more recently released $Model$ 000e, $Type\ B$, $Rev.$ 2 with 512MB RAM. Unless otherwise mentioned, the results are from the experiments run on $Model$ 0002. The operating system used is 'wheezy raspbian', which is a barebones version of Debian customised for Raspberry Pi.

The processor speed for Raspberry Pi can be over-clocked from 700MHz to 1.1GHz in turbo mode. Increasing the clock speed causes a slight increase in power consumption and temperature. For all our experiments, the Raspberry Pi was clocked at its default speed of 700MHz. The Raspberry Pi also comes with an inbuilt temperature sensor that could be used for real-time monitoring of processor temperature. This can be used to estimate whether external cooling is required. The maximum rated temperature it can handle is $85\,^{\circ}\mathrm{C}$.

TABLE I. FACTORS AND LEVELS FOR EXPERIMENTS

Factors	Levels
Arrival Rate, λ (rps)	1,2,4,8,10,20..100,200..500
Content Size, S (Kb)	1, 4, 16, 64, 256
Content Type	Static, Dynamic
Server Type	Raspberry Pi, Standard

The "standard server" used for our performance comparison is a tower server running *lighttpd* with PHP, powered by an AMD Phenom II X6 (6 Core) 1100T Processor with 16GB RAM and a clock speed of 800MHz per core.

For power measurements, we make use of a *'WattsUp? .Net'* power meter. It can be used to measure instantaneous power consumption at a resolution of 100 mW. The measurements are logged at a rate of 1 sample per second.

B. Experiment design

The factors and levels used for the experiments are highlighted in Table I. We use *httperf* [18] to send requests at varying arrival rates to the Web server to study the impact of arrival rates on server performance. The requests were modelled as a Poisson process with arrival rate λ, varied from 1 to 500 requests per second (rps). Each experiment ran for 10 minutes with the first and last minutes removed to avoid transient effects. Experiments were repeated 20 times. The results reported are the average of these trials. To minimise impact of network latency, the server and client were connected to the same wired network.

The following metrics were observed from the experiments: (i) average server response times, (ii) CPU utilisation, (iii) average CPU load, and (iv) power consumption. Unless otherwise reported, the number of errors in the system was observed to be either zero or insignificant to warrant further investigation.

The number of file descriptors on the client was increased to 65,000 (from 1,024) to avoid running out of file descriptors during experiments with high request rates. We also set the TCP timeout value to 10 seconds.

To study the impact of content size on performance, we experiment with contents at different sizes. We used server logs from a medium scale enterprise to identify the typical size of content being accesses. Figure 2 shows the distribution of content sizes being accessed over a day from an enterprise Web server. We observed that nearly 90% of content accessed is less than 100Kb. Motivated by the observed file size distribution, for our experiment design, we used the following content sizes: 1Kb, 4Kb, 16Kb, 64Kb and 256Kb.

For the static content, experiments on the standard server were conducted with two different settings. One with $lighttpd$ in its default configuration, and another with multithreading enabled to utilize all 6 cores of the standard server. We make use of 2 worker process per core to utilize all cores. Limiting the number of processes per core help improve utilisation [8].

To evaluate the performance of the server for serving dynamic content, we use $wview$, a weather management system as a dynamic content [19]. $Wview$ is implemented as a dynamic Website which responds with content generated dynamically for each request received.

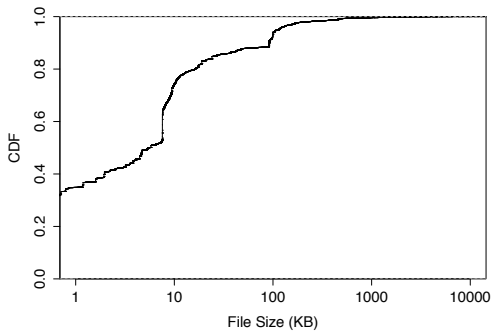


Fig. 2. Distribution of size of content accessed from an enterprise Weblog.

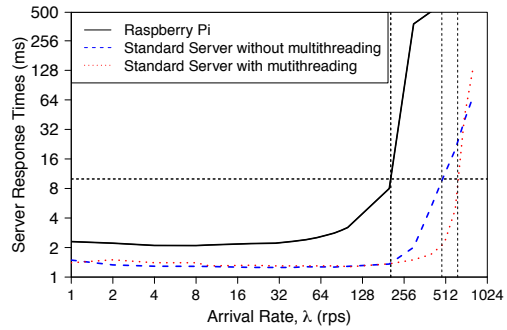


Fig. 3. Comparison of impact of request arrival rates on the response times of Raspberry Pi and a standard server for a single file (16Kb) download.

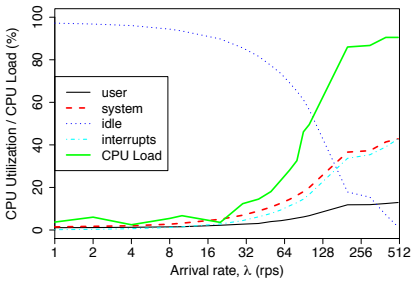
IV. EXPERIMENTAL RESULTS

A. Impact of request arrival rates

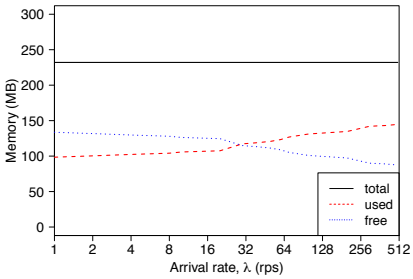
The system's ability to handle high loads is first evaluated, using response time measurements under different arrival rates. Figure 3 shows the comparison of download performance of a single 16Kb file for the Raspberry Pi server to that of a standard server configured in its default as well as multithreaded mode. It can be seen that the performance of the low-power server is comparable to that of a standard server for low to medium arrival rates (λ = 1 to 50 rps). Equivalent performance can be achieved by using multiple low-power devices and balancing its load using DNS redirection. For example, for a content size of 16Kb, to achieve a response time of under 10ms, a Raspberry Pi can handle loads of up to 200 rps while a standard server, in its default configuration without multithreading enabled, can handle approximately 450 rps. The performance advantage of the standard server is only 2.3x. With multithreading enabled, this maximum load went up to 625 rps (3.1x performance advantage). For the standard server, multi-threading has a modest performance advantage over the default (multi-threading disabled) configuration. Hence, unless otherwise mentioned, we have reported results from the default configuration with multithreading disabled.

To gain further insight into the performance of the low-power system, we considered system level factors such as CPU, and memory utilisations. We monitor the system resource utilisations for downloading a single 16Kb file with arrival rates varied between 1 rps to 800 rps. The mean CPU

The variation in response times for arrival rates from 1 to 8 rps is due to the clock granularity.

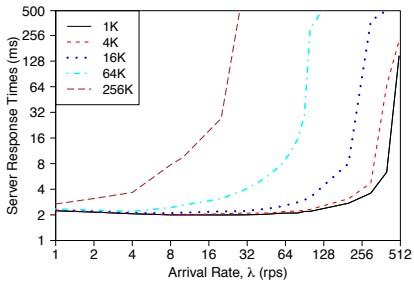


(a) CPU utilisation and Load of Raspberry Pi server.

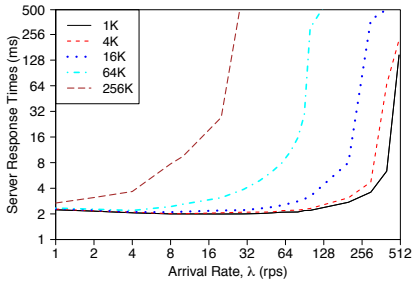


(b) Memory utilisation of Raspberry Pi server.

Fig. 4. Impact of arrival rate on system resource utilisations for serving static content of size 16 Kb.



(a) Raspberry Pi server.



(b) Standard server.

Fig. 5. Impact of content size on server response times.

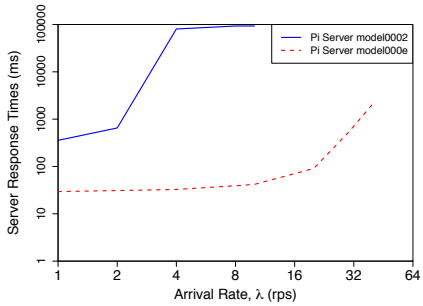


Fig. 6. Impact of request arrival rates on the response times for dynamic content on Raspberry server.

utilisations and CPU load for serving a static file of size 16Kb is shown in Figure 4a. The total CPU utilisation can be calculated as the inverse of the *idle* parameter, or as a sum of three parameters, *system*, *user*, and *interrupts*. The amount of CPU utilised by the operating system level processes is shown by the *system* parameter, the *user* parameter shows the utilisation by user generated processes, and *interrupts* shows utilisation due to software interrupts. Notice that CPU utilisation increases with increase in request arrival rate. This is because of the higher number of interrupts that need to be handled by the CPU. Note that CPU load is a measure of the size of the queue build up at the CPU. At high arrival rates, the performance of the server is drastically reduced because the queue at the CPU is always full. Results for memory utilisation, shown in Figure 4b, depicts only a minor increase in memory utilisation at high arrival rates. This shows that the Raspberry Pi is limited by the CPU, rather than the memory, for serving static content.

B. Impact of (static) content size

We conducted experiments where the content size was varied from 1Kb to 256Kb. Figures 5a and 5b show the response times for serving contents of varying file sizes as seen at the Raspberry Pi and standard server, respectively. It was observed that larger files cause the response times to go up at a lower request arrival rate; this performance degradation is due to large queue buildup at the server side associated with the service of a large file.

C. Dynamic content

Generally, the performance of Web servers are dependent on how the dynamic content is being generated and also on the performance of the database used to store the content. In most cases, the database is stored remotely. If the database is very small, then it is hosted in the same machine as the server.

For our experiments, we ran $wview$, a weather station management system which generates weather report archives based on sensor data from weather stations. We issue requests to the Website which responds with content generated dynamically for each request. This experiment was performed on two versions of the Raspberry Pi: $Model$ 0002, and a more recently released $Model$ 000e. It was observed that the performance of older version of the Pi was poor, even at 1 rps, but the newer version was able to perform reasonably well for loads upto 20 rps as shown in Figure 6. This shows that low-power servers are capable of handling dynamic content as well, but at reduced load. Having a higher RAM also help with improving the performance for serving dynamic content.

D. Power consumption

The baseline power consumption of typical tower servers are of the order of 80-300 W [3] and would increase by 20-50% when the server is loaded. The baseline power consumption for our standard tower server was measured to be 81W.

The baseline power consumption of Raspberry Pi was 1.8W without any peripherals. Under high load, the power consumption variation was minimal as it increased to a maximum of 2.2W. It was also observed that the power profile for Raspberry

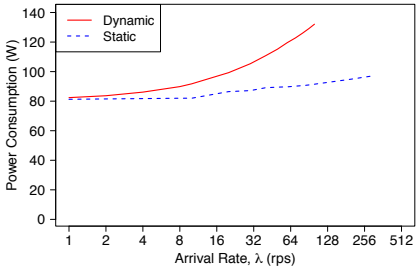
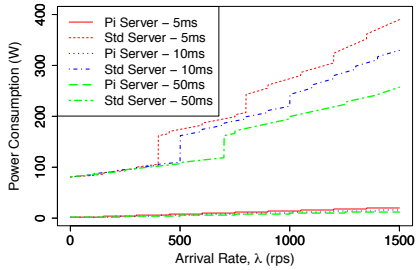
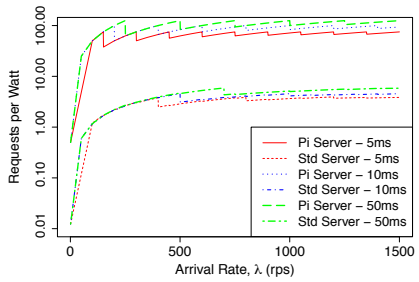


Fig. 7. Power consumption profile of a standard server for serving static and dynamic contents.



(a) Comparison of power consumptions.



(b) Comparison of requests served per Watt.

Fig. 8. Power to performance tradeoff comparison between Raspberry Pi and a standard server when scaling while adhering to the chosen response time thresholds (5ms, 10ms, 50ms) and serving a 16Kb file.

Pi is relatively independent of content type. In contrast, for a standard server there is a significant difference in power consumption depending on content type as well as server load.

A comparison of instantaneous power profiles of a standard server for serving static and dynamic contents with arrival rates varying from 1 to 100 requests per second is shown in Figure 7. For static content, the variation of power consumption was found to be 16.5% with the maximum power consumed at 300 requests per second being 97.5W. The variation is much more when it comes to dynamic content due to the additional processing required by the CPU and memory utilisation.

V. Multi-server Analysis

A. Simple average delay model

We study the power savings that could potentially be achieved by using multiple low-power devices instead of using a single (or multiple) high-power server as frontend for serving static content. Our model assumes that we make use of multiple low-power Raspberry Pi systems as the frontend to achieve a desired quality of service. We measure the quality of service in terms of response time thresholds.

Figure 8a shows the power consumption profile of Raspberry Pi and standard server configurations while downloading a static 16Kb file for three different response time thresholds: 5ms, 10ms, and 50ms, respectively. To generate this model, we have used the average power consumption of the Raspberry Pi at peak load which is 2W. For the standard server, we have used two threshold values, the baseline power consumption at 1 rps (81W) and the maximum power consumption at 300 rps (97.5W). Here, we have assumed that multiple low-power servers can be utilised to achieve the equivalent performance of that of a standard server. Similarly, to cater for higher arrival rates, we cascaded multiple standard servers as well. This cascading can be seen as step changes in Figure 8a. It is assumed here that load balancing while cascading is achieved using DNS redirection.

In addition to providing great energy savings for a given workload, when utilising multiple servers, low-power servers allow an opportunity for finer grained optimisation of the number of active servers that are on at each point in time. This can be valuable when turning servers on/off based on the expected workload requirements, such as to best serve the current workload. Such policies have been widely proposed to leverage the fact that many systems have long periods with lower server loads, to save energy by switching off unused server resources. For example, based on our experiments, a single standard server can serve anywhere between 1 to 400 requests per second (rps), consuming a similar energy for the full request rate range. On the other hand, by using multiple low-power servers, we can adjust, with relatively fine granularity, the number of low-power servers required to serve each of the request rates in this range. This allows for even greater energy savings during periods with very low load.

To compare the power efficiencies of the Raspberry Pi and standard server, we also calculated the number of requests that could be served per unit of power consumed. Figure 8b shows that even at high arrival rates, the Raspberry Pi cluster would outperform and is more energy efficient than a standard server for static content. At arrival rates of $\lambda > 300$ rps, we were able to serve 17x to 23x more requests per Watt using a Raspberry Pi than a standard server, at various response time thresholds.

The results show that at higher arrival rates, significant power savings can be realised by using multiple low-power devices. For example, at 1000 rps, to achieve a response time threshold of 5 ms, we need to use three standard servers resulting in a power consumption of 275 W. The same performance can be achieved at a cost of only 14W using seven Raspberry Pi servers. This is a power reduction of approx. 20x.

B. Dynamic policies

The experiment results and average scaling model showed that significant power savings could potentially be achieved by using low-power servers. This encouraged us to compare the impact of some simple multi-server policies on energy consumption of low-power servers and compare it to that of standard servers. This is achieved using simulations whose system parameters were determined based on the results obtained from our experiments using real hardware shown in Section IV. We define one baseline policy and two multi-server policies which provide a more fine-grained tradeoff between power and

overhead costs. The overhead cost captures the complexity and wear-and-tear associated with turning servers ON/OFF, and is measured here by server transitions metric. A server transition is either a Turn ON or a Turn OFF event.

1) Baseline policy: In this policy, the servers are turned ON and OFF without any delay, and an already active server is loaded with requests only if placing them at these servers would not violate the SLA response time constraint. Unless otherwise mentioned, we have used an SLA response time threshold of 10 ms. Assuming that we can turn servers ON/OFF instantly, the baseline policy provides a lower bound for the energy consumption. Here, we would be able to save maximum amount of energy because a resource (server) is utilised if and only if it is required, otherwise it is released (turned OFF). The major drawback of this type of system is that we would have to turn ON/OFF servers very frequently to cater to varying loads to achieve our SLA threshold.

2) Always ON policy: This is a more practical policy in which we keep a minimum of N servers $always\ ON$. This simple policy provides a tradeoff between energy consumption and number of ON/OFF transitions, as determined by the number of $AlwaysON$ servers.

3) Delayed turn OFF policy: Similar to the $always\ ON$ policy, the $delayed\ turn\ OFF$ policy is designed to provide a good tradeoff between power consumption and number of server transitions. In this policy, the server, once it is turned ON, remains in the ON state for an additional time period of δ after it has become idle. This would help the server to absorb, to certain degree, the bursty nature of the request traffic.

We note that the $delayed\ turn\ OFF$ policy using $\delta = 0$ and $always\ ON$ policy with $N = 0$, correspond to the baseline policy. Furthermore, at the other extreme, when the N and δ values are large, and no transitions are needed, then the two policies are similar in nature.

For our simulations, we modelled a system with multiple parallel queues, each with a finite queue size and fixed service times. The servers are turned ON/OFF adaptively, based on the policies above.

To capture the unique characteristics of the low-power and standard servers, we determined queue size thresholds and service rates based on measurements from our experiments. The service rate, for both the low-power as well as the standard server was obtained by using a least squares fitting of the data points while assuming an M/D/1 queue. We used this service rate values to calculate the queue sizes that would replicate the performance of the low-power and standard servers used in our experiments, and to achieve desirable service time guarantees.

For our request workload, requests are generated using a Poisson process. To capture how the multi-server system performs with increasing request rates, we generate request sequences with λ = 300, 600, 900, 1200 and 1500 rps respectively. The simulations were done for 1,000,000 arrivals for these high arrival rates. To avoid transient effects, the initial 10,000 and the last 10,000 responses from the simulations were omitted from our results.

In the simulation for $always\ ON$ policy, the number of minimum always ON servers were varied from 1 to 10 to capture its impact on energy consumed as well as overhead costs. The delay durations used in simulating $delayed\ turn\ OFF$ policy are δ = 0.01, 0.1, 1, 10, and 100 seconds.

To capture the cost of scaling (overhead cost) on the life of multi-server system, we look at the number of servers transitions that are required to serve these high load. We analyse the instantaneous power consumption (W) as well as the total power consumed over time, Energy (Wh), to gain insight into the energy savings that could potentially be achieved by using low-power multi-server systems over standard multi-server systems. We make use of the baseline power consumption values from our experiment, 2W for a single low-power server and 80W for a single standard server, to obtain the energy consumption over the day.

C. Simulation results

Figure 9 shows the average number of servers required for serving requests at varying arrival rates while maintaining and SLA threshold of 10ms. While these results show that more low-power servers are needed to serve any given request load, it is important to remember that each low-power server consumes much less energy. This is illustrated in Figure 10, which shows the comparison of aggregate daily power consumption for various arrival rates for the low-power and high-power servers.

Figure 11 shows the total number of transitions observed using the baseline policy, where we have only one server that is maintained in the ON state at all times. The rest of the servers are turned ON/OFF as and when required ensuring that the SLA is not violated. The queue sizes for the low-power and high-power servers were determined using an SLA threshold of 10ms. A bigger slack in the SLA threshold would potentially yield lower number of transitions.

An interesting observation from our simulations is that, at high arrival rates, the number of transitions for both the low-power and high-power servers are almost the same. This is an encouraging observation as we have a finer grained control over the number of transitions while using a low-power server than using a high-power server. Hence, we look at the impact of keeping certain number of servers always ON.

Figure 12 shows the impact of number of always ON servers on the daily power consumption for a low-power server. It can be seen that, for the workload considered here, and with a minimum of 7 $Always\ ON$ servers, we would get a constant power profile. The policy simply trades additional energy usage, due to always ON servers, against less server transitions overhead. To gain further insight into the impact of this policy on overhead cost, we look at the number of server transitions that correspond to keeping a minimum of N servers $Always\ ON$. Figure 13 shows the impact of these always ON servers on number of server transitions for a low-power server.

One of the potential drawbacks of having an $Always\ ON$ policy is that we miss out on the ability to save power by turning them OFF when the load becomes low. This is addressed by the simple $Delayed\ turn\ OFF$ policy, which turns the servers OFF after being idle for δ seconds. This deferral period allows the policy to absorb some of the variability in the inter-arrival times of the request process. Figure 14 shows the impact of delaying the turning OFF of the servers on the daily power

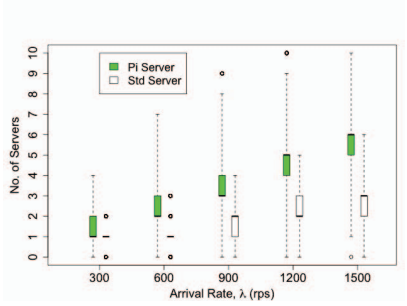


Fig. 9. Number of servers that is ON using baseline policy.

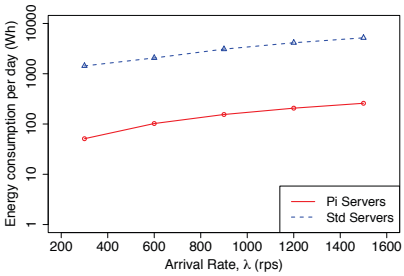


Fig. 10. Baseline comparison of power consumption under dynamic scenarios.

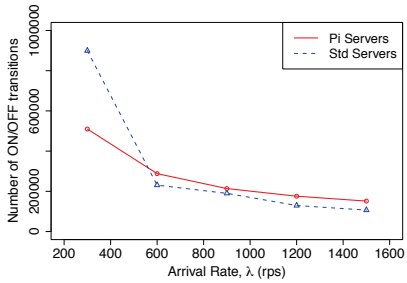


Fig. 11. Baseline comparison of server ON/OFF transitions under dynamic scenarios.

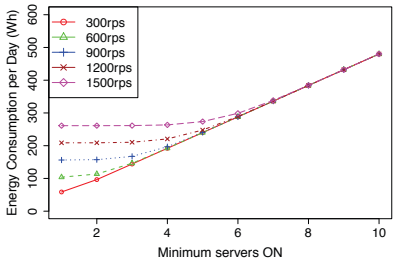


Fig. 12. Impact of Always ON servers on Energy Consumption

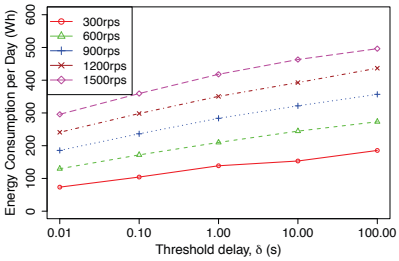


Fig. 14. Impact of delayed Turn-OFF policy on Energy Consumption

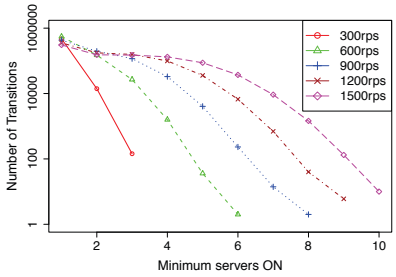


Fig. 13. Impact of Always ON servers on number of transitions

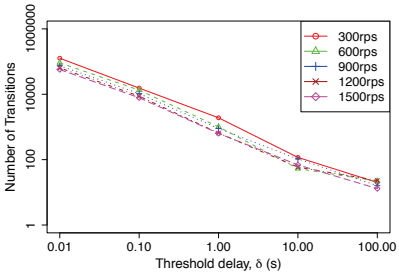


Fig. 15. Impact of delayed Turn-OFF policy on number of transitions for Low-Power Server

consumption for low-power servers. Similarly, the impact of delayed server turn OFF on the total number of transitions is shown in Figure 15. As expected, longer delays would help keep the number of server transitions to a minimum.

To compare performance of above policies for a low-power multi-server system, we look at the tradeoff between energy consumption and number of server transitions for both the policies as shown in Figure 16. As both these costs are outputs of our simulations, for clarity, we show the policy parameter values that correspond to each point. Interestingly, our simulation results show that the $Always\ ON$ policy is marginally better than the $Delayed\ turn\ OFF$ policy for reducing both the energy as well as the number of server transitions. These results show that the delayed turn OFF policy often end up wasting valuable resources after periods with many active servers. More intelligent hybrid policies, for example, are possible, but is outside the scope of this paper.

For practical implementation, a proper tradeoff for the operating point for a multi-server system should be a case which would minimise the energy consumption as well as number of server transitions. Depending on the service requirement, this point can be selected from Figure 16 and the corresponding policy parameters could be used.

In general, these energy-overhead tradeoffs for the low-power servers appear beneficial in relationship to the corresponding cost tradeoffs when using regular servers. For example, Figures 17 and 18 show the tradeoff curves for the $Delayed\ turn\ OFF$ and $Always\ ON$ policies respectively, for both low-power and standard multi-server systems. As we can see, the energy savings are considerable with the use of low-power multi-server systems over standard ones, confirming that our conclusions also holds in the context of dynamic policies and workloads.

VI. Concluding Remarks and Future Work

In this paper, we studied the power to performance tradeoff obtained while using a low-power micro-computing device like a Raspberry Pi as a Web server as opposed to a standard tower server. For serving static content, the low-power server

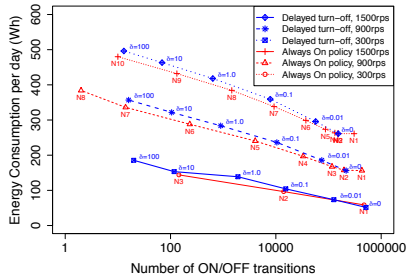


Fig. 16. Comparison of impact various policies to achieve tradeoff between power and scaling cost for low-power multi-server system

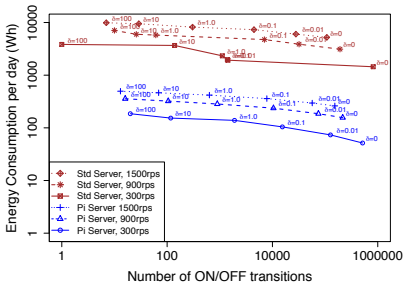


Fig. 17. Comparison of energy-transition tradeoff for low-power and standard multi-server systems under $delayed\ turn\ OFF$ policy

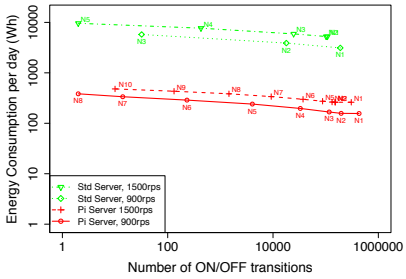


Fig. 18. Comparison of energy-transition tradeoff for low-power and standard multi-server systems under $always\ ON$ policy

achieved performance equivalent to that of a standard server for request rates up to 200 rps. For an architecture where multiple low-power servers are used instead of standard tower servers, we were able to observe reduction in power consumption by a factor of 17x to 23x. The use of multiple low-power servers would also enable finer-grained control over when to turn servers ON/OFF for a workload with variable arrival rates thereby achieving higher power savings.

Using simulations we validate the observations from our experiments that low-power multi-server systems provide better energy efficiency than standard multi-server systems. We also show that, with simple ON/OFF policies, we can achieve a good tradeoff between energy consumption and number of server transitions.

Although the performance of serving dynamic content is poor, the promising performance for static content is encourages us to consider them as a viable alternative to approaches for small-to-medium scale enterprises and universities. A promising solution may use one or more low-power frontend servers for static content and a standard server at backend for dynamic content. The attractiveness of deploying low-power servers is likely only going to increase with future advancements in low-power architectures and their capabilities.

As part of future work, we would factor in a couple of parameters that were not considered in our experiments and simulation models. First, we would be looking at the impact of peripherals on the server performance and power consumption. For example, a large peripheral terabyte storage would cause additional power consumption and could potentially result in performance bottlenecks as well, due to limitations in interface and read/write speeds. Second, we will look at the aspects of reliability and redundancy. After an initial disk failure (due to a sudden high fluctuation in power supply voltage), our device under test remained operational without any failure for over 3 months. The system operated under standard load conditions, serving static content with a load of $\lambda < 100$ rps. Typical enterprise servers would have built-in redundancy and the 'time to recover' from failure is very low. This 'time to recover' and 'availability' figures need to be quantified for these low-power devices if they are to be commercially deployed. Also, the cost of maintenance and redundancy need to be factored in.

Finally, we would consider as part of future work, the use of these low-power systems as proxy servers for HTTP-based Adaptive Streaming (HAS) [20]. HAS is a protocol where streaming is done by splitting the video into HTTP chunks and a range of these chunks are requested depending on the required video quality and available bandwidth. Use of low-power proxy servers along with a carefully designed prefetching policy could potentially achieve better streaming performance. Moreover, HTTP chunks can be considered as static content. Hence, based on our results, we expect these low-power proxy servers to be able to handle the streaming workload with minimal loss in performance.

REFERENCES

[1] "Netcraft web-server survey july 2013," http://news.netcraft.com/archives/2013/07/02/july-2013-web-server-survey.html.

[2] W. Vogels, "Beyond server consolidation," *Queue*, vol. 6, no. 1, pp. 20–26, 2008.

[3] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the internet with nano data centers," in *Proc. ACM CoNEXT '09*, pp. 37–48, 2009.

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

[5] A. Bogus, *Lighttpd*. Packt Publishing, 2008.

[6] J. Almeida, V. Almeida, and D. Yates, "Measuring the behavior of a world-wide web server," Boston, MA, Tech. Rep., 1996.

[7] M. F. Arlitt and C. L. Williamson, "Internet web servers: workload characterization and performance implications," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 631–645, 1997.

[8] R. Hashemian, D. Krishnamurthy, M. Arlitt, and N. Carlsson, "Improving the scalability of a multi-core web server," in *Proc. 4th ACM/SPEC ICPE*, pp. 161–172, 2013.

[9] D. Menasce, "TPC-W: A benchmark for e-commerce," *Internet Computing, IEEE*, vol. 6, no. 3, pp. 83–87, 2002.

[10] "Specweb2009," http://www.spec.org/web2009/.

[11] "Specpowerssj_2008," http://www.spec.org/power_ssj2008/.

[12] G. Bai, K. Oladosu, and C. Williamson, "Performance benchmarking of wireless web servers," *Ad Hoc Netw.*, vol. 5, no. 3, pp. 392–412, 2007.

[13] C. Amza, A. Chanda, A. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite, "Specification and implementation of dynamic web site benchmarks," in *Proc. IEEE WWC-5*, pp. 3–13, 2002.

[14] 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 SOSP*, pp. 1–14, 2009.

[15] P. Sehgal, V. Tarasov, and E. Zadok, "Optimizing energy and performance for server-class file system workloads," *Trans. Storage*, vol. 6, no. 3, pp. 10:1–10:31, 2010.

[16] V. Mathew, R. Sitaraman, and P. Shenoy, "Energy-aware load balancing in content delivery networks," in *Proc. IEEE INFOCOM*, pp. 954–962, 2012.

[17] "Calxeda," http://www.calxeda.com/.

[18] D. Mosberger and T. Jin, "httperf - a tool for measuring web server performance," *SIGMETRICS PER.*, 1998.

[19] M. Teel, "Using wview," *Linux J.*, vol. 2010, no. 196, . 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1883498.1883500

[20] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri, "Helping hand or hidden hurdle: Proxy-assisted http-based adaptive streaming performance," in *Proc. IEEE MASCOTS*, pp. 182–191, 2013.