The Journal of Systems & Software 146 (2018) 14–25

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Controversy Corner

How does docker affect energy consumption? Evaluating workloads in and out of Docker containers

Eddie Antonio Santos*, Carson McLean, Christopher Solinas, Abram Hindle

Department of Computing Science, University of Alberta, Edmonton, Canada

ARTICLE INFO

Keywords:
Virtualization
Docker
Containerization
Energy consumption
Cloud computing
Microservice

ABSTRACT

Context: Virtual machines provide isolation of services at the cost of hypervisors and more resource usage. This spurred the growth of systems like Docker that enable single hosts to isolate several applications, similar to VMs, within a low-overhead abstraction called *containers*.

Motivation: Although containers tout low overhead performance, how much do they increase energy use?

Methodology: This work statistically compares the energy consumption of three application workloads in Docker and on bare-metal Linux.

Results: In all cases, there was a statistically significant (*t*-test and Wilcoxon $p < .05$) increase in energy consumption when running tests in Docker, mostly due to the performance of I/O system calls. Developers worried about I/O overhead could consider baremetal deployments over Docker container deployments.

1. Introduction

Virtualization provides a number of benefits when deploying software, such as process isolation and resource control. Process *isolation* means that software developers can make strong assumptions about the state of the system, including the operating system configuration, and having the exact software dependencies needed for the system. Virtualization often allows for *resource control* such that operators can configure precisely how much CPU, memory, or access to network interfaces a particular application has. Virtualization platforms often use *images* – snapshots of the complete system needed to run an application – thus deploying an app is as easy as instantiating an image. Traditionally, virtualization has been implemented through *virtual machines*, in which one machine may host several guest operating systems. However, the intervention of the hypervisor,[1] means that applications effectively must use two kernels – directly through the guest operating system, and indirectly through the hypervisor – when accessing resources such as network and storage. This may be considered an undesirable overhead. This prompted the need for a low overhead alternative which operates similar to a virtual machine. Recently, sophisticated features in the Linux kernel – namely, namespaces and control groups – made a new form of low overhead virtualization possible: *containerization*. Containers are a lightweight alternative to virtual machines, as they offer isolation (processes, file-system, network) and resource control (CPU, memory, disk) without the overhead of an additional kernel. Container management software such as Docker (Docker Inc.), LXC (Canonical Ltd., 2017), and rkt (CoreOS, Inc., 2017), are quickly displacing virtual machines as the virtualization solution of choice (Arijs, 2016; Ferranti, 2016).

Given the blistering pace of the adoption of containerization, what is the impact of containerization on energy consumption? Changes in software have significant and measurable differences in power and energy consumption (Hindle, 2012; Zhang et al., 2010; Ellis, 1999; Vasić et al., 2009; Gupta et al., 2011). Containerization, in principle, lacks the overhead of virtual machines, however, is an abstraction layer nonetheless. From an energy consumption standpoint, *to what extent does the abstraction of containers compare to no abstraction at all?*

In this paper, we empirically test this using numerous measured workloads, run with and without containerization. In practice, container providers such as Docker *do* add additional overheads, such as the AUFS file system, and an abstracted networking layer. We seek to *quantify* the impact that these overheads have on energy efficiency. We compare the energy consumption of various scenarios run on baremetal Linux – that is, the applications are running on one kernel, without any virtualization at all – in contrast to Docker-managed containers, using “off-the-shelf” Docker images. We use total system power

* Corresponding author.
E-mail address: easantos@ualberta.ca (E.A. Santos).

[1] We use the term hypervisor for any virtual machine monitor that is hosted on top of an existing operating system, or is a module of the host operating system kernel, such as KVM (Linux Kernel Organization, 2016).

https://doi.org/10.1016/j.jss.2018.07.077
Received 5 May 2017; Received in revised form 26 April 2018; Accepted 12 July 2018
Available online 23 August 2018
0164-1212/ © 2018 Elsevier Inc. All rights reserved.

consumption (or "wall power") to estimate total energy consumption. We run several iterations of each experiment, the results of which we present and explain why we see differences in energy consumption between bare-metal Linux and Docker. To our knowledge, this is the first empirical comparison quantifying the effect that containerization (using Docker) has against using no abstraction at all; prior work has focused on comparing Docker to other forms of virtualization, without a bare-metal baseline.

This work suggests that there is no free lunch for containerization in terms of energy consumption. Containerization implies a trade-off between latency, energy and maintainability, and it is up to the individuals or teams in charge of deployment to determine which is more costly in their particular scenario. The extent of this trade-off is measured and discussed.

2. Prior work

Prior and related work addressed benchmarking the overhead of containers, virtual machines, and baremetal. In this section we discuss works relating to container performance (Section 2.1), virtual machine energy consumption performance (Section 2.2, and comparisons between container and virtual machine energy consumption (Section 2.3).

2.1. General container performance

Other work has evaluated container performance metrics such as run time, CPU usage, and network utilization. Felter et al. (2015) compared CPU, memory, I/O, and network performance of Docker and KVM against bare-metal Linux. In most cases, Docker adds little overhead, and almost always outperforms KVM. They also tried sample loads on Redis and MySQL. They found that, in some cases such as the Redis example, Docker performs comparably to bare-metal when configured appropriately. The authors found that Docker's UnionFS file system abstraction has negative performance penalties compared to a standard Linux ile system. In contrast, our work directly measures energy consumption of running similar benchmarks, both on bare-metal Linux compared to within a Docker container.

Ruan et al. (2016) conducted experiments comparing Docker and LXC. They confirmed with Felter et al. (2015) that Docker's AUFS implementation and port mapping lead to higher I/O latency, LXC did not necessarily suffer from these issues but it can be configured in such a way. They also evaluated the cost of running containers inside of virtual machines, which comes at a heavy cost. They also evaluate cloud and container offerings from Amazon and Google, concluding that containers should be offered on baremetal, rather than inside of VMs.

In general, quicker runtime is correlated with lower energy consumption; however, power must also be measured alongside with performance to observe the overall energy consumption of a task. These works demonstrate that Docker does incur some runtime overhead.

2.2. Virtual machine power use and energy consumption

Previous work has focused on *virtual machine* power and energy consumption. Xu et al. (2015) measured CPU and total power usage in both Xen and KVM hypervisors. They found that Xen generally has a greater power overhead than KVM when processing network traffic, attributed to "excessive interrupt requests". They found that as the load is more evenly distributed among virtual machines, power consumption increases. This paper elaborates on the effect of Docker on network energy consumption.

2.3. Power usage of containerization and virtual machines

Many researchers have investigated the energy/power performance of containers versus virtual machines. The consensus is that virutal machines induce far more performance overhead than containers.

Some work has compared virtual machines to containers directly. Morabito (2015) compared the power usage of traditional virtual machine hypervisors (KVM, Xen) to container based virtualization (Docker, LXC). In all cases, the container style virtualization used marginally less power, but overall neither virtualization method showed significant differences. Morabito did not consider runtime differences, hence this work cannot make conclusions about overall energy consumption. Further, there was no comparison to bare-metal Linux performance. Both of these concerns are addressed in our work.

van Kessel et al. (2016) used internal hardware sensors to determine the difference in power consumption of Xen against Docker. They found that Docker is more efficient on CPU-bound and disk bound loads. In contrast, our work compares against bare-metal Linux measuring wall power instead of internal power sensors to quantify the abstractions provided by Docker.

Shea et al. (2014) compared the power consumption of network transactions using virtualization such as KVM, Xen, and OpenVZ, in contrast to a bare-metal system. Only OpenVZ can be considered container-based virtualization. They measured both wall power and CPU power using Intel's Running Average Power Limit (RAPL). The authors found that power measured through RAPL was always a fraction of the measured wall power. They found a difference in the power overheads of network transactions on different virtualization platforms. However, they concluded that the overheads were tunable.

Our work concentrates only on Docker's container-based virtualization and baremetal. We measure wall power only, because we wanted to capture the total system power usage. Additionally, we measured more scenarios than just network transactions. Because of the consensus that is reported: VMs use more energy than containers, we opted to compare containers versus baremetal.

3. Methodology

We want to compare the energy consumption of running a workload within a Docker-managed container (the treatment) against running the same workload on "bare-metal" (the control). To estimate the energy consumption of one workload, we ran one server (the *system-under-test* or SUT) with the software of interest; we ran an external system to initiate tests on the SUT and record the power measurements (the *test runner*); and we used a *power meter* to measure the instantaneous power consumed by the SUT. We setup the systems to run the desired software – either starting the service (bare-metal Linux) or start a new container (Docker) that has already been built. We then initiated the tests on the test runner, which would induce a workload on the SUT after a two minute pause. During the test run, we collected root-mean-squared (RMS) power measurements, and recorded them. We used the power measurements to estimate the total energy consumption on the SUT in two scenarios: the software running on bare-metal Linux versus the software running within a Docker container.

Importantly, the System-Under-Test is *not* the same machine as the test runner; thus initiating the tests (test runner) is isolated from test execution (SUT). Therefore, a separate server is used as the test runner for both initiating tests and recording energy usage statistics from the power meter.

This section describes the hardware and instrumentation we used to run tasks and collect power samples. An overview of our full setup is provided in Fig. 1. Our hardware setup consisted of a rack-mount server as our System-Under-Test (Section 3.1), a digital power meter (Section 3.2) to collect power samples, and a test runner (Section 3.3) to initiate the workloads.

3.1. System-Under-Test

The System-Under-Test (SUT) is a Dell PowerEdge R710 rack-mount server. A summary of its hardware is listed in Table 1. Although the R710 is intended to be used with redundant power supplies, multiple

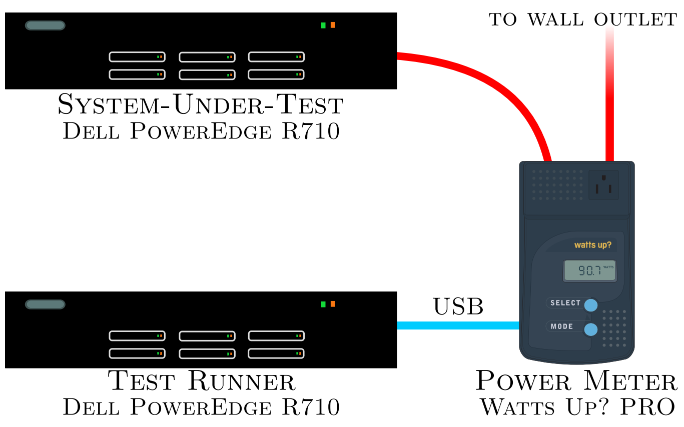


Fig. 1. Hardware test setup: one rack-mount server System-Under-Test; and one test-runner. Power measurements were collected with a WATTS UP? PRO.

Table 1
Hardware configuration of the System-Under-Test and the test runner.

CPU	2 × six-core Intel Xeon X5670 at 2.93 GHz
RAM	72 GiB ECC DDR3
Network	Gigabit Ethernet connection
Storage	146GB SAS hard drive at 15,000 RPM
Power supply	870 W (120 V ∼ 12A at 60 Hz)

network interfaces, and redundant RAID storage, we only utilized one power supply, one network interface (a gigabit Ethernet connection), and one hard drive for our tests. The two Intel Xeon X5670s contain 6 cores each, totalling 12 real cores, and with hyper-threading enabled they appear as 24 logical processors to Linux.

A summary of the software installed is listed in Table 2. Docker was installed on the System-Under-Test. For bare-metal versions of Apache, PHP, WordPress, MySQL, and PostgreSQL, we used `apt-get`. Redis was installed from source on bare-metal Ubuntu Linux. All of the Docker application software ran within Docker-managed containers. When installing software on Docker, we used the official image hosted on Docker Hub (2016). Note that the WordPress image inherits from the `php:5.6-apache` image, which installs both PHP and Apache. Hence, the only image we had to explicitly install was the one containing WordPress.

3.2. Power measurements

This paper focuses on comparing the energy required to perform several tasks. However, we cannot measure energy directly. Instead, we measured the instantaneous wall power drawn by the System-Under-Test. For this, we used a Watts Up? PRO (2016) power meter.

The WATTS UP? PRO is a device with a Type B AC power socket. It samples the voltage and current draw of the electrical appliance plugged into its socket. Since power is voltage multiplied by current, the meter can report the instantaneous power usage of an electrical appliance—in our case, a rack-mount server as our System-Under-Test. Since we are interested in the total power usage of the entire system – including the CPU, but also memory, storage, network interfaces, peripherals, internal cooling, and even overhead due to the power supply – we opted to measure wall power, instead of using onboard measurement, such as Intel's RAPL for measuring CPU power usage alone. The WATTS UP? PRO calculates the root-mean-square (RMS) of thousands of samples over the course of one second (Watts Up Pro, 2016). Previous work by McCullough et al. (2011) found that collecting RMS measurements at a frequency of one measurement per second from a WATTS UP? power meter is sufficient for accurate energy consumption estimation (McCullough et al., 2011). One sample per second is also the maximum sampling speed of the hardware (Watts Up Pro, 2016).

Table 2
Software versions used on the System-Under-Test.

Software	Version	Docker Image
Distribution	Ubuntu Server 16.04.1 LTS	
Kernel	Linux 4.4.0	
Docker	1.12.1	
Apache	2.4.10	php:5.6-apache
PHP	5.6.24	php:5.6-apache
MySQL	5.7.15	mysql:5.7.15
WordPress	4.6.0	wordpress:4.6-apache
Redis	3.2.3	redis:3.2.3
PostgreSQL	9.5.4	postgres:9.5.4

We used a modified version of `yyongpil`'s `wattsup`[2] software to retrieve the power measurements from the WATTS UP? PRO and save them on the test runner. Every second, the wattage used by the System-Under-Test is pulled from the WATTS UP? PRO, transferred over USB to the test runner, and then written to `stdout`. Collection scripts on the test runner controlled the test runs for each of our case studies and recorded measurements for each test run in order to gather power data along with timestamps. This information was saved to a local SQLite3 database on the test runner.

However, power is not energy. Energy is the integration of power over time. The WATTS UP? PRO yields RMS power samples of one second in duration – several measurements of instantaneous power averaged over one second. Given an initial timestamp (t_i) and an end timestamp (t_f), we can use the sum of power samples to estimate the energy required to complete a task. We approximated energy using a sum of power samples, taken at a regular frequency. This is analogous to using the rectangle method of approximating an integral with a duration Δt of 1 second (Eq. (1)).

$$E = \int_{t_i}^{t_f} P(t)\,dt \approx \Delta t \sum_{k=i}^{f} P_{RMS}(t_k) \tag{1}$$

We wrote Python scripts (Santos et al., 2018b) that implemented the above estimation, taking in test data from the SQLite3 databases on the test runner, which had power in watts with timestamps. Each timestamp was asserted to be about one second apart, thus making our estimation valid. The summation produces an estimate of the total energy consumed for a single run of a test. We considered each test run to be one energy sample. We ran each test 40 times, giving us 40 energy samples per case study per configuration. Before each test run, we had the machine sleep for two minutes to reset the machine to its idle run state, as Chowdhury et al. (2016) discovered that running tests in quick succession may alter the power state of the machine, artificially

[2] https://github.com/yyongpil/wattsup

skewing results. These energy summaries are then compared, grouped by case study, for bare-metal Linux versus Docker.

Our null-hypothesis is that Docker does not cause a statistically significant increase in energy consumption when compared to bare metal Linux. For all statistical tests, we use an $\alpha = 0.05$ and we consider a p-value less than 10^{-4} to be "near zero" and sufficient evidence for rejecting the null-hypothesis. Higher p-values simply indicate a lack of sufficient evidence to reject the null-hypothesis and are not a confirmation that Docker increases energy consumption.

3.3. Test runner

For initiating the tests and recording the power samples, we used a Dell PowerEdge R710 rack-mount server, identical in hardware specification and configuration as the SUT. We wrote collection scripts in Python (Santos et al., 2018b) that initiate the tests (described in Section 4) on the System-Under-Test through network requests, while simultaneously recording energy statistics from the WATTS UP? PRO via USB with yyongpil's `wattsup`. We recorded timestamps for every power sample.

For each experiment:

1. We started the service on the System-Under-Test (if applicable). In Docker, we started one or more new containers from their respective Docker images.
2. On the test runner, we initiated a batch of test runs.
3. For each test run, the test runner optionally performed a per-test initialization.
4. The test runner would then sleep for two minutes.
5. The test runner then induced a workload on the System-Under-Test via network requests.
6. During each test run, the test runner recorded the instantaneous power measurements of the SUT and the timestamp every second.
7. After all test runs from a batch have finished, we calculated the energy per each test run.

The test runner was connected to the System-Under-Test via a gigabit switch.

3.4. Reproducability

In the interests of reproducability, we have released the Python scripts that initiate tests; the database of raw results obtained from the WATTS UP? PRO; and the R scripts used to analyze the resulting data. The replication package is available either on GitHub[3] or through Zenodo (Santos et al., 2018b).

4. Case study

Three open-source software projects were selected to test the difference in energy consumption of running the app on bare-metal Linux versus within a Docker-managed container. Each of the applications stresses different hardware resources, and together provides performance and energy insights on which types of applications are most suited for Docker. WordPress with MySQL represents an extremely popular website solution that stresses CPU. Redis and PostgreSQL are common database solutions, pressuring memory and I/O respectively. Considering the popularity and breadth of applications selected as case studies, the results give relevant insight into the effect of Docker on energy consumption when compared with bare-metal installs.

4.1. Idle

As a baseline, we were interested in any possible overhead of running the Docker service without placing any load on the system. In order to estimate how much energy is expected to be used at idle, the system was left to idle for exactly 10 min, during which power usage was recorded. In order to be consistent with the methodology used for the following case studies, we inserted an additional 2 min of idle time before each test run during which power samples were not recorded. This test was performed 40 times sequentially, and can be considered a baseline for bare-metal Linux and Docker.

"Idle" means the system has been operating long enough to achieve a stable state with nothing but the base operating system in operation, meaning that none of the other services under test (PostgreSQL, Redis, MySQL, Apache) were running, or were active in any way. When performing the Docker baseline, the only difference is enabling the Docker background service. *Zero* containers were running, so we measured the overhead of just the Docker daemon itself.

Since time is fixed in this test, any difference in energy *must* be due to a difference in power consumption.

4.2. WordPress

WordPress is an open-source content management system (AboutWordPress, 2016). As of February 2017, Docker Hub has had over 10 million WordPress pulls (Docker Hub, 2016) and WordPress powers over one quarter of the top 10 million websites worldwide (W3Techs.com, 2016).

We installed WordPress manually for the bare-metal Linux version, as per the WordPress official documentation (WordPress, 2016). We used Docker Compose (2016) for installing WordPress within Docker. Both methods installed the same versions of WordPress, MySQL, PHP, and Apache, as listed in Table 2. On the bare metal system, MySQL and Apache ran as services. Docker required two containers: one container held Apache, which runs WordPress with `modphp`, while another contained the MySQL database. These were automatically setup and connected using Docker Compose. We generated a blog using the WP Example Content Plugin 1.3 (Ferrara, 2016), whose database was copied both into the bare-metal installation and the Docker installation. The default MySQL Docker image mounts a volume on the host (i.e., outside of the container) to persist data. Thus, database writes do not have to access Docker's AUFS storage layer.

We used Tsung 1.6.0 (Tsung, 2016) to perform an HTTP load stress test on the WordPress server for which the test runner was monitoring energy usage. Tsung, running on the test runner, created virtual clients that simulate a large number of users visiting the WordPress front page and randomly navigate the site. Each test was exactly 15 min long. Starting from no load, the test added 100 simulated users per second. Each user loaded the WordPress homepage content, which in turn required database queries in order to retrieve the posts and other content. We performed the full test 40 times sequentially, in order to produce 40 energy samples, with 2 min of idle time between tests to ensure accuracy of the energy measurements.

We invoked Tsung with the following command,

`tsung -f` *configuration* `start`

where *configuration* is an XML file which sets the load as follows:

```
<load>
  <arrivalphase phase="1" duration="15" unit="minute">
    <users maxnumber="100000" arrivalrate="100"
          unit="second"></users>
  </arrivalphase>
</load>
```

[3] https://github.com/eddieantonio/collect-data

We used the following Docker Compose configuration on the SUT:

```
version: '2'
services:
  db:
    image: mysql:5.7.14
    volumes:
    - "./.data/db:/var/lib/mysql"
    restart: always
    ports:
    - "3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    depends_on:
    - db
    image: wordpress:4.6.0-apache
    links:
    - db
    ports:
    - "80:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_PASSWORD: wordpress
```

After which, we built and started the containers using the following commands:

```
docker-compose build.
docker-compose up -d.
```

4.3. Redis

Redis is an open-source, in-memory key store that can be used as a database, cache, or message broker (Redis, 2016). As of February 2017, Docker Hub has had over 10 million Redis pulls (Docker Hub, 2016). We chose the Redis to test the overhead of a workload that is predominantly memory, CPU, and network bound (it does minimal accesses to storage).

Redis was installed in Docker with the version specified in Table 2. On bare-metal Linux, Redis was built from source. For Docker, we used the official image to build a single container which held the Redis server. The official image downloaded from Docker Hub disables periodic persistence of the in-memory database to permanent storage, hence we disabled this on the bare-metal configuration as well.

The Redis benchmark suite, `redis-benchmark` was used to create a workload of 1000 parallel clients making a total of 1.5 million requests. This involves a great deal of network traffic from the server running the clients, as well as doing a large amount of memory accesses. We ran the full test 40 times sequentially, which produced 40 energy samples, with two minutes of idle time between each sample.

We invoked `redis-benchmark` with the following command:

On the SUT, we started the Redis container with the following command:

4.4. PostgreSQL

PostgreSQL is an open-source, object-relational database management system (DBMS) (PostgreSQL, 2016). As of February 2017, PostgreSQL has been pulled over 10 million times (Docker Hub, 2016).

PostgreSQL includes `pgbench` for performance benchmarking. PostgreSQL was installed on both the SUT and test runner servers with the version specified in Table 2. On bare-metal Linux, we ran PostgreSQL as a service, while Docker held the database processes in a single container. It is important to note that the Ubuntu 16.04 version enables SSL by default whereas the Docker install does not. In order to compare equivalent configurations, we disabled SSL in the bare-metal Linux PostgreSQL installation. We also ran a test on the bare-metal configuration with SSL enabled, to compare the overhead of Docker against the overhead of encrypting queries. In Docker, the default PostgreSQL image creates a volume mounted on the host (i.e., escaping the container) for persisting data. Thus, writes do not access Docker's AUFS storage layer.

The test consisted of running `pgbench` on the test runner with 50 clients, each performing 1000 database transactions on the SUT of "a scenario that is loosely based on TPC-B" (The PostgreSQL Global Development Group, 2016; TPC, 1990). We performed 40 sequential tests to produce 40 energy samples. Before each test, we ran `pgbench -i` to initialize the database, then waited for two minutes of idle time before starting the test proper. The entire test was performed for both bare-metal Linux and Docker.

We invoked `pgbench` with the following command:

On the SUT, we started the PostgreSQL container with the following command:

5. Results

After collecting all power samples, estimating energy per each test run, we ran some statistical analyses on the results to determine whether there is a significant difference in energy consumption to run a task on bare-metal Linux compared to a Docker container. A summary of our results is given in Table 3. Our raw data is available online (Santos et al., 2018a). The scripts we used to compute the results in this section are available in the replication package (Santos et al., 2018b).

First, we determined whether both energy samples on Linux and on Docker were normally distributed using the Shapiro–Wilk normality test. Then, we applied various tests to determine if both samples came from the same distribution. For normally-distributed data, we used a paired Student's *t*-test. Otherwise, we applied non-parametric tests: a Kruskal–Wallis rank sum test, and a pairwise Wilcoxon rank sum test. In all two sample experiments, we found that the difference in distributions of energy consumption in Docker compared with Linux was statistically significant, with a *p*-value of less than 10^{-4}, no matter which test we used. To quantify the difference, we calculated the effect size. For all tests, we used Cliff's delta, which simply compares how often samples from one distribution are greater than samples in the other distribution. As shown in Table 3, for the WordPress and Redis experiments, the distributions from Docker are all greater than the observations from Linux with a maximum Cliff's delta of 1.0. The other two experiments also had large effect size, according to Cliff's delta, with small overlaps in distributions.

Finally, we calculated the linear correlation, Pearson's correlation coefficient, of measured energy, with the WATTS UP? PRO, with run time (time to run the benchmark). Recall that energy is power × time. Thus, energy should be strongly correlated with time (an *r* value of + 1.0). In every case, we found that energy was strongly correlated with time, however, since the *r* value of each test was not exactly 1.0, we assert

Table 3
Summary of results obtained for each experiment. "Correlation" refers to the linear correlation between estimated energy with the elapsed time of the test run. Note: for the "idle" experiment, calculating correlation of energy with run time does not make sense because the elapsed time is fixed.

Case Study	Normal	Effect Size		Correlation (r_{Et})	
		Cliff's *d*	Cohen's *d*	Linux	Docker
Idle	No	0.80			
WordPress	No	1.00		0.83	0.99
Redis	Yes		11.31	0.98	0.98
PostgreSQL	Yes		1.55	0.99	0.95

that other factors must be influencing the total energy rather than energy being completely explained by run time. First and foremost the majority of these tests are busy and utilize many resources, so their power use is high and somewhat stable during the test. This runtime analysis and correlation is necessary because some tasks could be potentially parallelized whereby energy use is related to the number of cores or processes used rather than runtime. For example, some problems can be solved very quickly by using multiple cores, their total CPU time will be high but their runtime will be lower than running serially on one core. This behaviour is demonstrated clearly in the work of Lima et al. (2016), who provide examples where different concurrency monads used in Haskell result in different runtimes with differing energy effeciencies. In some cases there are clear non-linear tradeoffs between short runtime and energy consumption. Thus you might not see energy consumption gains by reducing runtime if you use more power while you do it. Thus factors that can affect the linear correlation between runtime and energy consumption could be the utilization of CPU cores, the variance in the power use across the test-run, the distribution of utilization of peripherals such as disk and network, the distribution of requests or commands across the test run, or the distribution of I/O operations across peripherals. Thus if we had more variation of demand and utilization in our benchmarks the correlations might not be so high.

The results are presented in two ways: summaries of the energy data is presented in *violin plots* (Figs. 2, 4, 8,6) which can read somewhat like box plots where each "violin" represents one distribution. The width of the violin at any given point represents the density of measurements observed at that point. To give a sense of tendency, a line is drawn at the median of the sample distribution. Summaries of the power data are given as *density plots* (Figs. 3, 5, 10, 7), with hexagonal bins. Each bin represents a cluster of observations at the given time and wattage. Darker hexagons represent a denser concentration of observations.

5.1. Idle

The distribution of energy consumption with no load for 10 minutes is given as a violin plot in Fig. 2. A density of power is provided in Fig. 3. Using the Shapiro–Wilk test, we found that neither the bare-metal Linux nor the Docker distributions are normally-distributed.

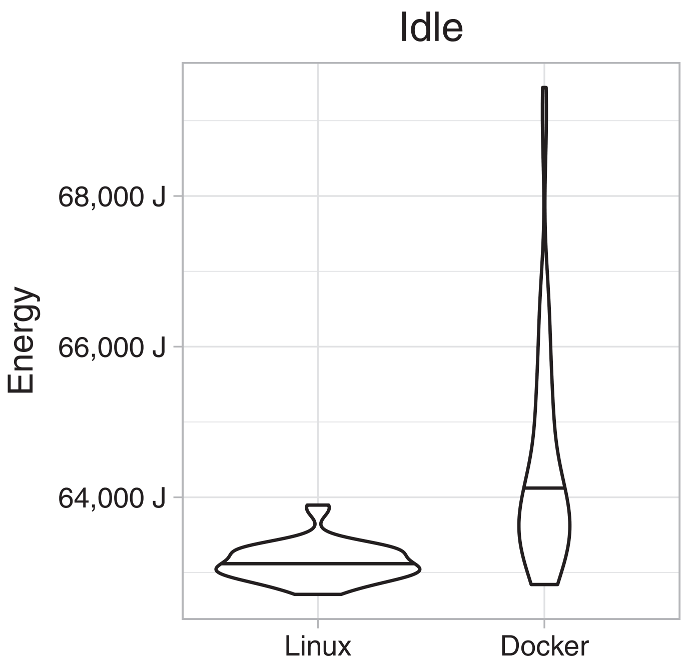


Fig. 2. Violin plot of idle energy consumption.

Using the non-parametric Kruskal–Wallis test and pairwise Wilcoxon rank sum test, we obtained a *p*-value close to zero, indicating that the distributions are indeed different. Using Cliff's delta, we got an effect size of 0.80, indicating that values in the Docker distribution are nearly 80% likely to be greater than an observation in the bare-metal Linux distribution. Another way to think about this difference, is that three-quarters of the time, we observed that running on bare-metal Linux with no load would use less than 63,380 J of energy, whereas if simply the Docker daemon was running (with no containers running), three-quarters of time we would observe the machine consuming *more than* 63,380 J of energy for doing *nothing* for ten minutes. This energy difference cannot be attributed to performance, since time is fixed to 10 minutes in both cases.

This baseline establishes that, since the Docker daemon is an unavoidable service that must run – regardless if containers are running or not – running Docker comes with a power overhead. Whether this difference in energy consumption over time is negligible is for operators to decide, however, later we describe how to make back the difference in energy consumption.

5.2. WordPress

The distribution of energy consumption for running a simulated load on a WordPress server under Linux and within Docker is shown in Fig. 4. A density of power is provided in Fig. 5. Using the Shapiro–Wilk test, only the distribution of energy consumption under bare-metal Linux was normally-distributed; hence, we used non-parametric tests for comparison and effect size. Both the Kruskal–Walis and pairwise Wilcoxon rank sum test yielded a *p*-value near zero, meaning that the distributions are significantly different. For effect size, we computed a Cliff's delta of 1.0, implying completely non-overlapping distributions. In other words, *all* samples in the Docker test runs were higher than all samples in bare-metal Linux. Finally, the linear correlation of energy and run time for bare-metal Linux and Docker were of 0.8303 and 0.9885 respectively.

5.3. PostgreSQL

For this test, we had three energy consumption distributions: bare-metal Linux, with SSL disabled; bare-metal Linux, with SSL enabled; and Docker, with SSL disabled. Not only are we testing the difference between Docker and bare-metal, but we are also introducing the difference between encrypting connections on bare-metal as well. The three energy distributions are shown in Fig. 6. A density of power is provided in Fig. 7. Using the Shapiro–Wilk test, we determined that all three samples are normally distributed, with the smallest *p*-value being 0.54 for the Docker energy distribution. Thus, we used pairwise paired Student's *t*-tests to compare each distribution to the others. The baseline (Linux with SSL disabled) is significantly different, both to Docker with SSL disabled, and with Linux with SSL enabled, with *p*-values near zero. Interestingly, Docker with SSL disabled does *not* provide sufficient evidence to claim a significant difference when compared to Linux with SSL enabled, due to having a *p*-value of 0.15. This implies that the trade-off between encrypting connections with SSL is similar to the trade-off between using Docker without encryption.

To understand the effect size, we used Cohen's *d*. Cohen's *d* compares the means of the two normally-distributed samples, taking into account their pooled standard deviation to determine the offset (Cohen, 1988). Larger results indicate a larger difference in the means. Comparing PostgreSQL with SSL disabled on bare-metal Linux versus the same configuration in Docker yields a very large Cohen's *d* of 1.55. However, simply turning on SSL on bare-metal Linux, testing against Docker with SSL disabled yields the smallest effect size obtained in this paper: 0.31. This corroborates the findings of Chowdhury et al. (2016) that simply using SSL/TLS has a significant effect on energy

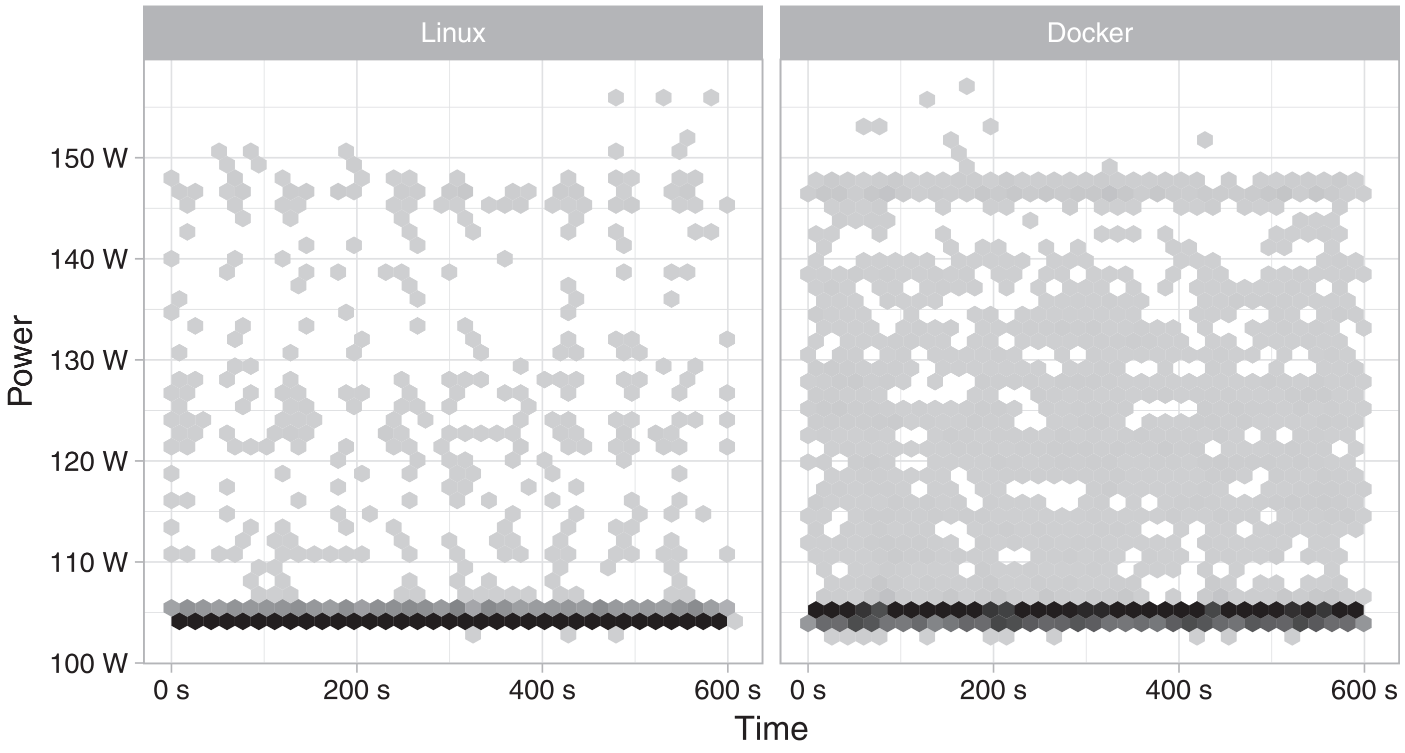


Fig. 3. Density plot of wattage measurements across all idle test runs over time.

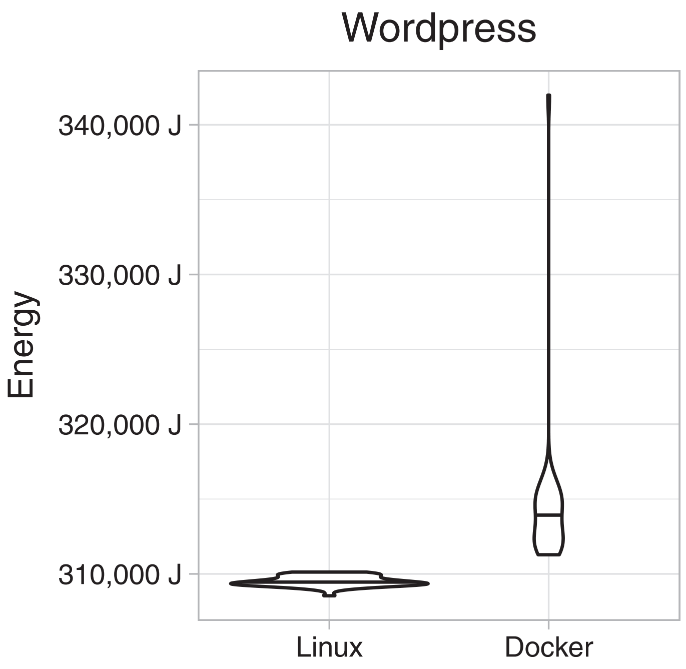


Fig. 4. Violin plot of energy consumption in the WordPress experiment.

consumption. The difference between bare-metal Linux versus enabling SSL on the same configuration also has a large effect size, with Cohen's *d* calculated to be 1.32.

5.4. Redis

The distribution of energy consumption for running `redis-bench` on Linux and within Docker is shown in Fig. 8. A density of power is provided in Fig. 10. Using the Shapiro–Wilk test, both samples are normally-distributed. We compared the distributions using a Student's *t*-test and obtained *p*-values near zero. Using Cohen's *d*, we obtained a huge effect size of 11.31. Thus, this experiment shows the greatest difference between running in Docker versus running on bare-metal Linux.

The linear correlation of energy with time yielded 0.996 and 0.995 for Linux and Docker, respectively. Given the very high correlation of energy with time, we also compared the amount of time it took to complete each test (Fig. 9). Since elapsed time is greater under Docker, energy consumption will be greater, unless the power used in Docker is drastically lower, which is not the case (Fig. 10).

6. Discussion

Fig. 2 shows that having the Docker service running consumes significantly more energy at idle than without Docker. The `dockerd` background process explains the difference in energy consumption. Recall that Docker is *not* required for containerization; rather, Docker provides a convenient infrastructure for running containerized applications in Linux. However, `dockerd`, the Docker server, written in the Go programming language periodically wakes up to do work, even if it is managing zero active containers. Using `perf top -p $(pgrep dockerd)` we found that the `dockerd` was periodically calling functions related to scheduling and garbage collection in Go (e.g., `runtime.findrunnable`, `runtime.scanobject`, `runtime.heapBitsForObject`, `runtime.greyobject`).

Table 4 quantifies the energy and financial impacts of Docker. % Difference shows the increase in average energy consumed by Docker over a bare-metal Linux implementation. Idle, Redis, and PostgreSQL tests allow for financial cost examples that were extrapolated from the benchmarks previous discussed. Tsung's testing method for WordPress does not permit for a scalable task, making financial examples unavailable. With these metrics, implementation teams can estimate the cost of Docker over a bare-metal solution. In all cases, Docker costs more, however, in cases such as PostgreSQL, the difference – while statistically significant – may be judged as negligible. This estimation does not account for external factors and secondary effects such as

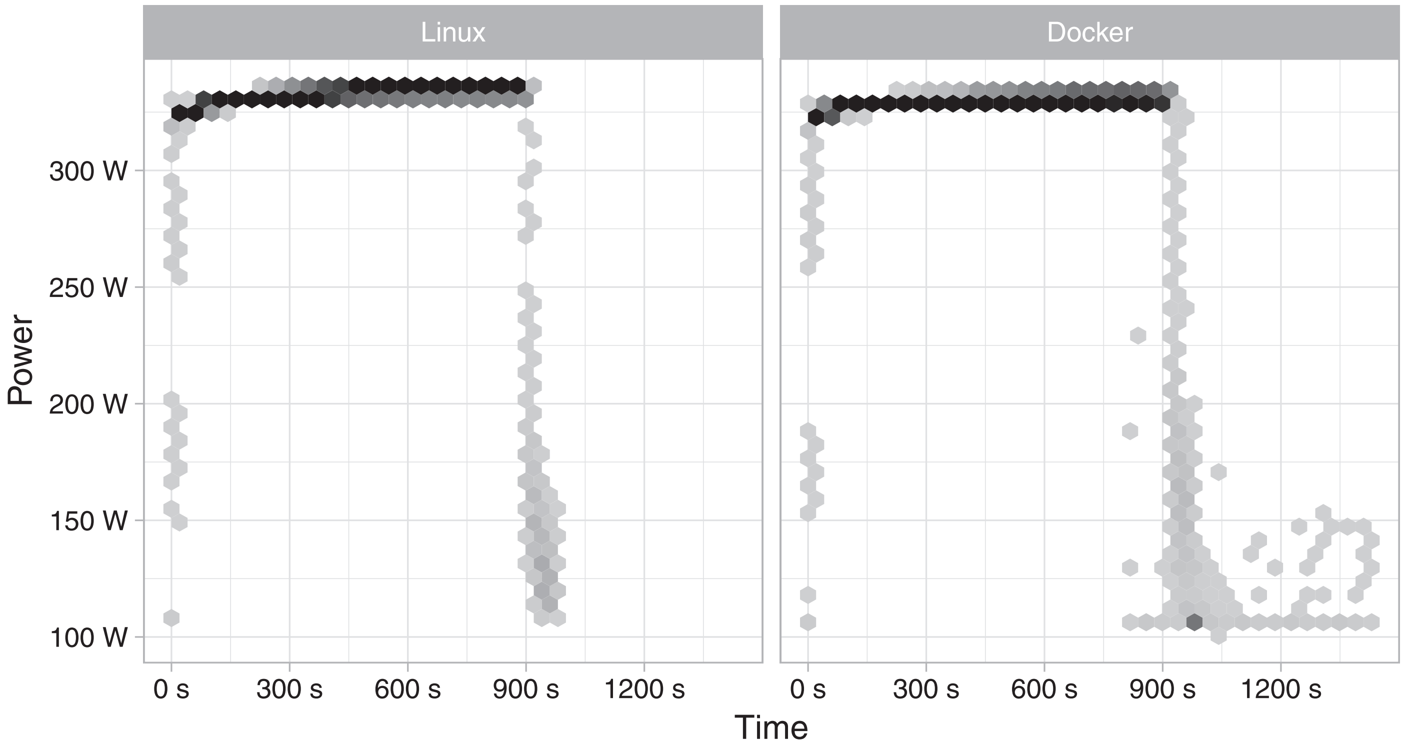


Fig. 5. Density plot of wattage measurements across all WordPress runs over time.

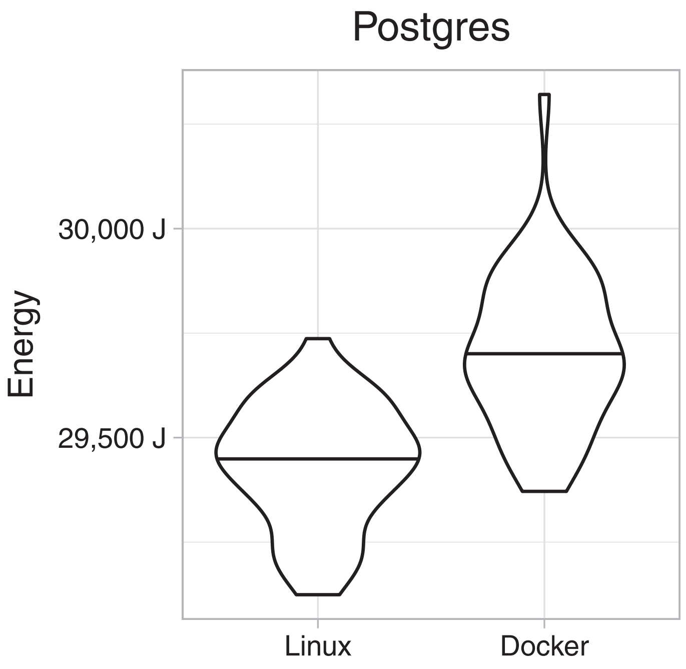


Fig. 6. Violin plot of energy consumption in the PostgreSQL test.

supporting network equipment, and air conditioning the server environment to address the induced load.

6.1. Explaining WordPress performance

A possible service deployment strategy is to create virtual networks wherein each microservice is in its own container. Only public-facing services (of which there should be few) will be required to use any kind of per-connection encryption, as provided by SSL/TLS. Our results show that, while PostgreSQL in Docker uses more energy compared to the same configuration in Linux, the effect is not very large compared with running PostgreSQL on Linux with encryption turned on. In that case, running private PostgreSQL instances within containers, with unencrypted inter-container communication may actually be a more energy efficient option.

6.2. Explaining Redis performance

Using `strace -c`, we measured the time spent in system calls running the `redis-bench` application. We found that in both bare-metal Linux and in Docker, the Redis server was mostly calling `write()` (about 82% of all system calls). A 32–39 s benchmark induced around 1.7 million `write()` system calls. The notable difference is that the Redis server running within a Docker container spent more than twice as long doing writes (93.94 ms) versus running the server on bare-metal Linux (44.08 ms spent in `write()`). This explains a small part of the longer runtime on Docker (and thus higher energy consumption), though it does not come close to explaining the large gap in run time.

6.3. Composition and orchestration

Modern cloud deployments are favoring containers as an alternative to virtual machines. The result is having more than one microservice per physical machine where orchestration frameworks such as Kubernetes (2018), Apache Mesos, or Docker Inc. Swarm mode (2018) allow for better management of multiple services sharing a host and improved utilization of resources. Unfortunately the availability of comprehensive benchmarking tools for such implementations is lacking. Such configurations are extremely complex to run with high degrees of variability in their implementation, resulting in the management of their energy use to be an active research topic (Piraghaj, 2016).

This work serves as a stepping stone to enable further research in the performance of composed and orchestrated containers. Three obvious immediate avenues of future work that remain open are:

What do orchestrated container deployments look like? By mining configurations from GitHub, from industry, and from others, can we answer what do Kubernetes and Docker container deployments

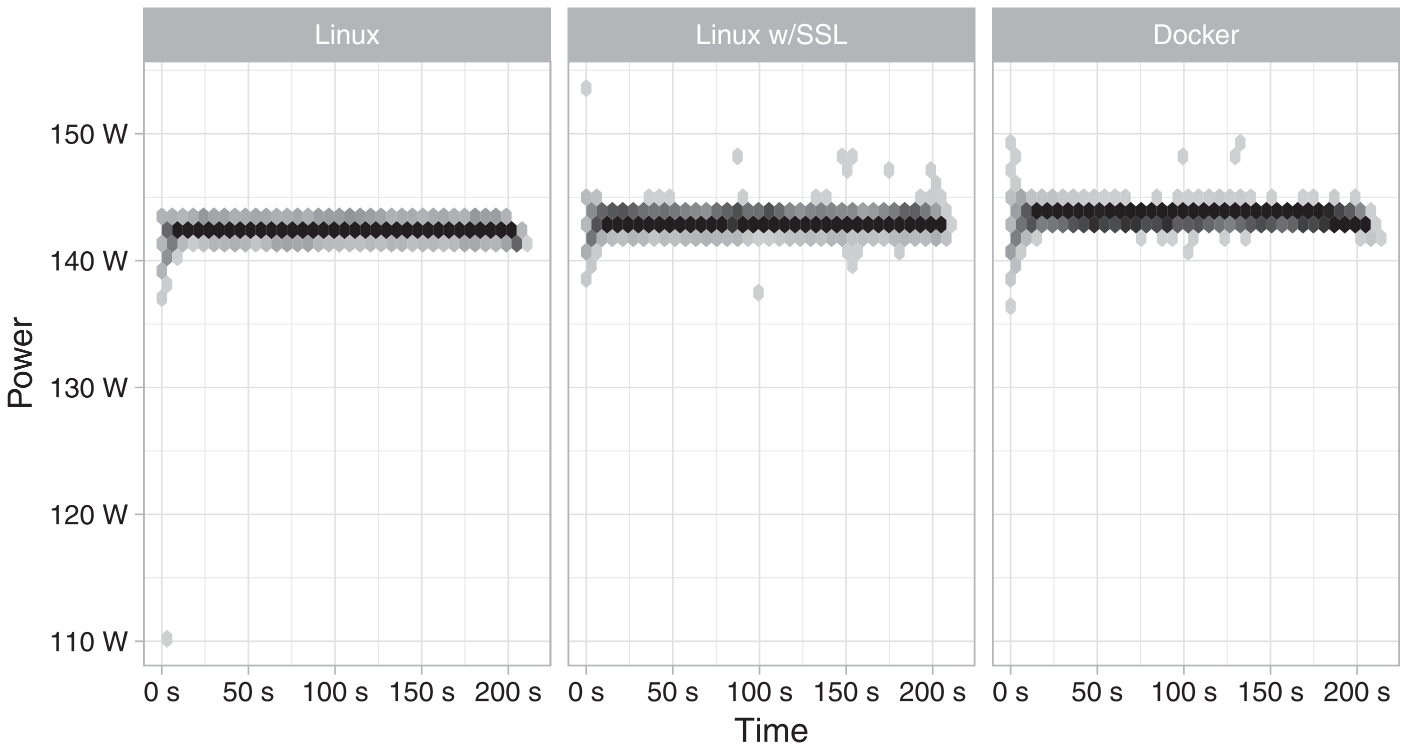


Fig. 7. Density plot of wattage measurements across all `pgbench` runs over time.

actually look like? What are the container loads expected? What are the contexts of host machines that run containers? What makes up multi-host orchestrations?

What does an orchestration benchmark look like? What should be tested? What kinds of containers are typically orchestrated? Can we use existing benchmarks from science, industry, or business?

What are the different performance and development costs of orchestrating on bare-metal, containers, and VMs? Performance in terms of throughput, energy consumption, and resource utilization can be compared across different solutions after orchestration.

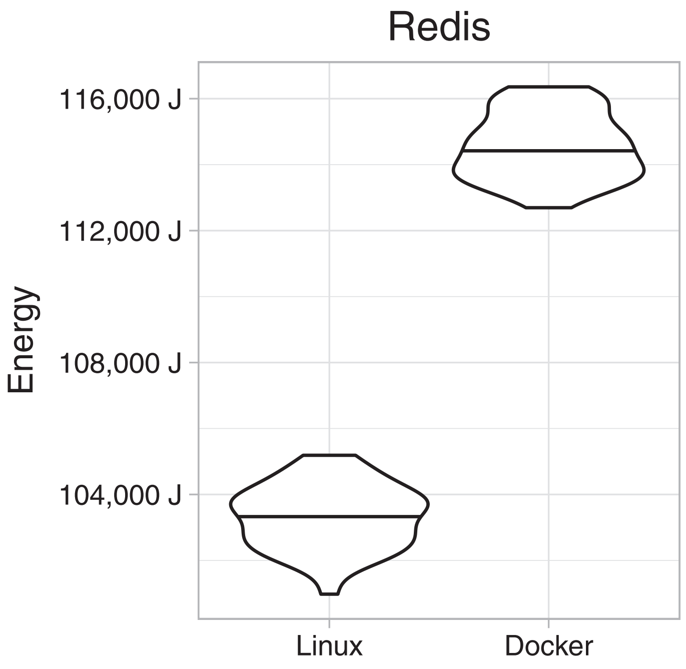


Fig. 8. Violin plot of energy consumption running the Redis benchmark.

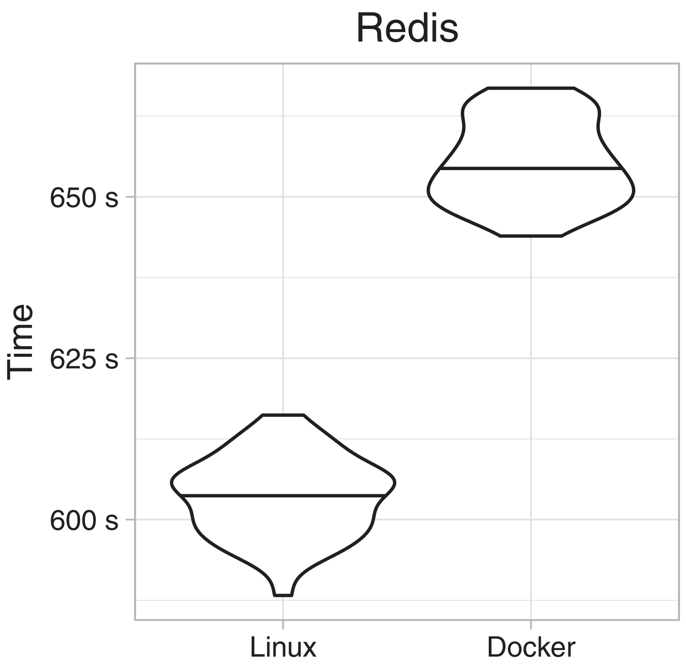


Fig. 9. Violin plot of elapsed time running the Redis benchmark. Elapsed time may explain the difference in energy (Fig. 8).

6.4. Recommendations to practitioners

In general the energy consumption differences between bare-metal and containers are different but not huge. For instance Table 4 shows a 14 cent USD difference between Redis on bare-metal and Redis in a Docker container on a large workload.

Fundamentally the sustainability costs are not high, and developers will probably make decisions regarding performance issues or developer time.

Table 4
Quantified difference in energy used under Docker compared to native Linux bare metal. Dollar estimates are the cost to perform the example task, using the US 2016 average industrial electricity rate of USD $.0676/kWh (U.S. Energy Information Administration).

Case Study	Example Task	Linux	Docker	% Difference	$ Difference
Idle	One week idle	$1.20	$1.22	2.08%	+$0.02
WordPress				1.56%	
Redis	10^9 queries	$1.29	$1.43	10.85%	+$0.14
PostgreSQL	10^6 transactions	$0.55	$0.55	0.97%	+$0.00

CPU and Memory bound containers: these tasks should incur very little cost or overhead being run within a Docker container. Developers deploying CPU and memory bound containers need not worry much about performance issues or sustainability related issues.

I/O bound containers: these tasks do incur some overhead even when mounting directories from the host filesystem. If 5% runtime overhead for I/O is significant for your I/O bound task then perhaps bare-metal is more appropriate. In terms of sustainability impact, the impact is relatively small. Table 4 shows for PostgreSQL the energy cost is not noticeable for 10^6 transactions. Thus what should be emphasized is the need for addressing I/O latency.

Isolation: the cost of bare-metal deployment is that processes are far less isolated than if they were containerized. Furthermore the development environment is more shared on bare-metal deployments than on container deployments. This isolation of dependencies can result in many developer hours saved not worrying about cohabitating their processes with other developer's processes. If development cost is more of a concern than performance then we recommend that developers employ containers and accept that I/O and network performance may suffer mildly.

7. Threats to validity

Construct validity In general, using benchmark frameworks does not necessarily model real usage of the applications. This is especially true when there has been no investigation in to what a realistic typical usage of these applications would be, as was the case here. Future work should start by discovering what is representative of typical usage for each of the test cases (a profile), or benchmark using real world data and actions, if at all possible.

Docker has a number of configuration options concerning networking and the file system. Likely, any administrator deploying Docker in production would tweak these settings extensively. As such, our usage of "off-the-shelf" defaults (deploying straight from the Docker Hub image using a command like `docker run postgres:latest`) is not representative of true deployments using Docker. Most notably, Docker's default networking option is `bridge` which delivers poor performance under high packet rate tasks such as the Redis benchmark due to overhead in creating a separate network stack and NAT packet rewrites (Antoan Milkov, 2013).

Each of the studied applications was only serving a single host. Each benchmarking tool provided support for simulating multiple clients, and these features were used in all tests. The quality of the multiple simulated clients from a single client when compared to real-world users is unknown and so may not realistically stress the applications. Furthermore, the servers were only using a single gigabit Ethernet connection, where real deployments may see multiple network connections sharing the load of requests.

Internal validity One may call into question the precision and reliability of the power measurements obtained from the WATTS UP? PRO. Another threat to validity is that we left services such as `OpenSSH` and `OpenVPN` running on the System-Under-Test, whose power usage is also included in all of the power measurements. Thus, the *exact* numbers may not be indicative of real loads, but, after taking several energy samples, the comparisons can give an idea of the differences. SSH and VPN were only used for configuring the machines before the tests were run; none of the traffic in any of the tests used SSH or used the same network interface as the VPN.

External validity The applications selected as test cases do not necessarily generalize or represent the performance of other applications, even of similar type. Generalizations are hard to draw from such a small set of applications. Even different versions of the same application

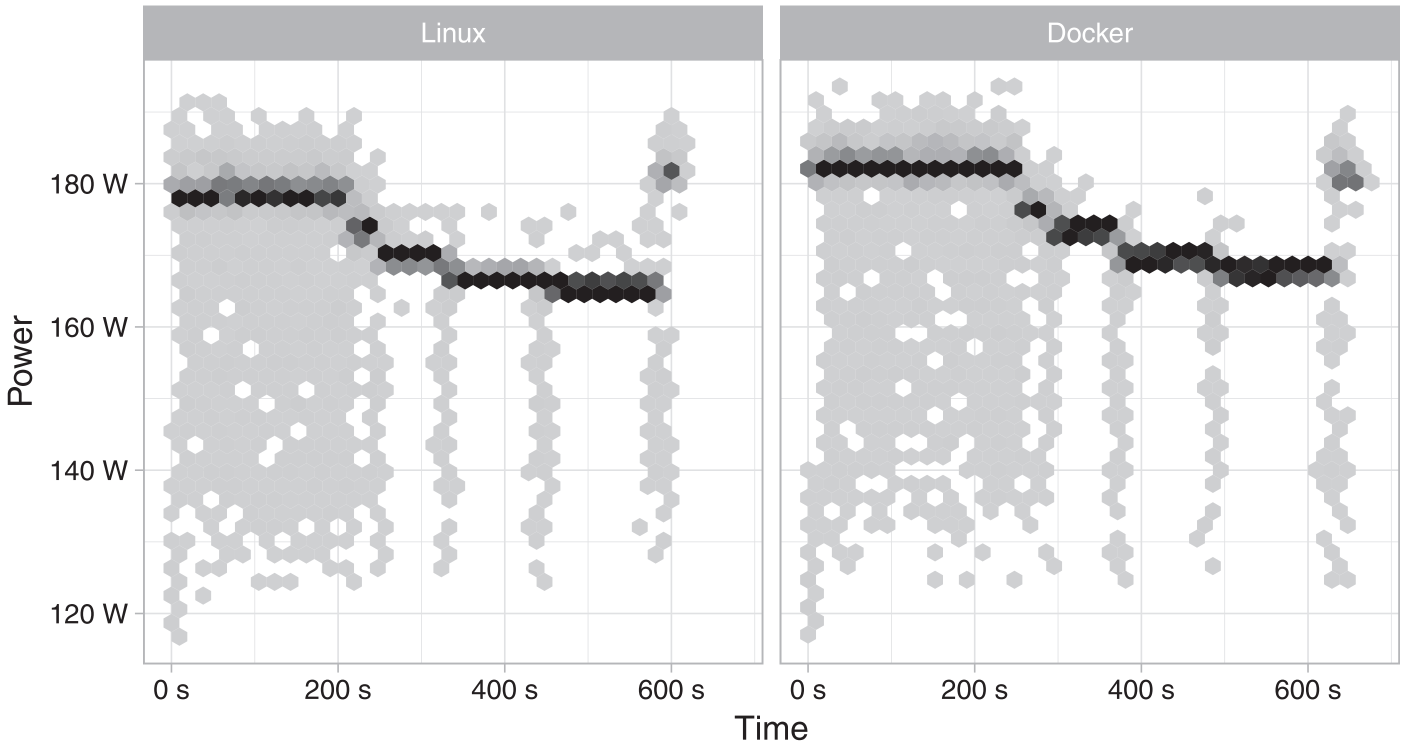


Fig. 10. Density plot of wattage measurements across all Redis Benchmark runs over time.

have different energy profiles (Romansky and Hindle, 2014; Chowdhury and Hindle, 2016) – especially when the load makes different operating system calls. External parties need to consider the resources required by their application in order to best evaluate the consequences of using Docker.

Finally, the System-Under-Test that we used only represents a single machine configuration. Having multiple test platforms that differ in performance and architecture would allow for more generalized findings.

8. Conclusion

In this paper, we compared the energy consumption of various workloads running within Docker-managed containers and on "bare-metal" Linux. After almost 2 days and 20 h of total time collecting power measurements, we found that, in all cases, workloads running in Docker have a measurable energy overhead. Simply running `dockerd` idle induces a 2 W difference in average power, and thus an increase in energy over time. However, the increase in energy consumption may mostly be attributed to runtime performance. In the case of Redis and WordPress, the increase in energy can be attributed to increase in runtime—thus the decrease in performance explains the increase in total energy consumption.

Operations teams must decide which is more important: sustainability, energy consumption, and run-time performance of reduced resource usage by employing bare-metal Linux, or the process isolation and maintainability of containerized applications of Docker. Saving on heat and energy is important for some scenarios, yet the human cost of maintenance can far exceed run time, energy, and cooling/air-conditioning costs of Docker's minor inefficiency. An aspect that must be investigated further is the impact of container orchestration, such as Kubernetes (2018), that encourages operation teams to deploy many containers on a single machine. At the end of the day, for most developers to use Docker or not will depend less on energy and more on how much I/O operations affect their product and if they can afford a 5% decrease in efficiency – which for a database server could be an unacceptable cost.

This work's main contribution is that, while many works have compared containerization performance, no work to date has clearly compared the energy consumption of Docker containers versus bare-metal in a variety of settings. There is still much left to evaluate such as orchestration and different container configurations.

Acknowledgment

We are indebted to the support of Steve Sutphen. Funding: Abram Hindle is supported by an NSERC Discovery Grant.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.jss.2018.07.077.

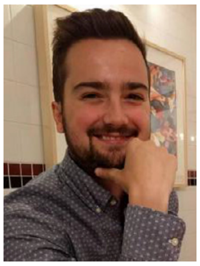
References

About —PostgreSQL, 2016. https://www.postgresql.org/about/ (Accessed: 09-09-16).
About-WordPress, 2016. https://wordpress.org/about/ (Accessed: 05-09-16).
Antoan Milkov, 2013. Running redis with docker (performance issue). https://stackoverflow.com/a/26559466, October (Accessed: 15-01-18).
Apache Mesos, 2018. http://mesos.apache.org/(Accessed: 18-04-18).
Arijs, P., 2016. Docker usage statistics: Increased adoption by enterprises and for production use. http://www.coscale.com/blog/docker-usage-statistics-increased-adoption-by-enterprises-and-for-production-use July (Accessed: 21-12-16).
Chowdhury, S.A., Hindle, A., 2016. GreenOracle: estimating software energy consumption with energy measurement corpora. Proceedings of the 13th International Conference on Mining Software Repositories. MSR '16, ACM, pp. 49–60.
Chowdhury, S.A., Sapra, V., Hindle, A., 2016. Client-side energy efficiency of HTTP/2 for web and mobile app developers. Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). 1. pp. 529–540. March
Cohen, J., 1988. Statistical power analysis for the behavioural sciences. NJ: Lawrence Earlbaum Associates, Hillside.
Canonical Ltd. Linux containers, 2017. https://linuxcontainers.org/, February (Accessed: 02-06-17).
CoreOS, Inc. rkt, a security-minded, standards-based container engine., 2017. https://coreos.com/rkt, February (Accessed: 02-06-17).
Docker Hub. 2016https://hub.docker.com/explore/, September (Accessed: 02-09-16).
Docker Inc. What is docker?, 2016. https://www.docker.com/what-docker#/VM, November (Accessed: 21-12-16).
Docker Inc. Swarm mode overview | Docker documentation. 2018. https://docs.docker.com/engine/swarm/ (Accessed: 18-04-18).
Ellis, C.S., 1999. The case for higher-level power management. Hot topics in operating systems. Proceedings of the Seventh Workshop on 1999. IEEE, pp. 162–167.
Felter, W., Ferreira, A., Rajamony, R., Rubio, J., 2015. An updated performance comparison of virtual machines and linux containers. Proceedings of the 2015 IEEE International Symposium On Performance Analysis of Systems and Software (ISPASS). IEEE, pp. 171–172.
Ferranti, M., 2016. Survey: 96% increase in container production usage over past year A clusterhq. https://clusterhq.com/2016/06/16/container-survey/, June (Accessed: 30-01-17).
Ferrara, J., 2016. WP example content. https://wordpress.org/plugins/wp-example-content/, September.
Gupta, A., Zimmermann, T., Bird, C., Nagappan, N., Bhat, T., Emran, S., 2011. Detecting energy patterns in software development. Microsoft Research Microsoft Corporation One Microsoft Way Redmond, WA 98052.
Hindle, A., 2012. Green mining: a methodology of relating software change to power consumption. Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR). IEEE, pp. 78–87.
van Kessel, J., Taal, A., Grosso, P., 2016. Power efficiency of hypervisor-based virtualization versus container-based virtualization. University of Amsterdam.
Lima, L.G., Soares-Neto, F., Lieuthier, P., Castor, F., Melfe, G., Fernandes, J.P., 2016. Haskell in Green Land: analyzing the energy behavior of a purely functional language. Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). 1. pp. 517–528.
Linux Kernel Organization Inc. KVM, 2016. http://www.linux-kvm.org/page/Main_Page, November (Accessed: 06-02-17).
McCullough, J.C., Agarwal, Y., Chandrashekar, J., Kuppuswamy, S., Snoeren, A.C., Gupta, R.K., 2011. Evaluating the effectiveness of model-based power characterization. Proceedings of the USENIX Annual Technical Conference. 20.
Morabito, R., 2015. Power consumption of virtualization technologies: an empirical investigation. arXiv:1511.01232.
Kubernetes | production-grade container orchestration, https://kubernetes.io/ (Accessed: 18-04-18), 2018.
Overview of Docker Compose, 2016. https://docs.docker.com/compose/overview/, September (Accessed: 02-09-16).
Piraghaj, S., 2016. Energy-efficient management of resources in enterprise and container-based clouds. http://www.buyya.com/gridbus/students/SarehPhDThesis2016.pdf, March (Accessed: 16-01-18).
Watts Up Pro, 2016. http://www.vernier.com/products/sensors/wu-pro/(Accessed: 21-12-16).
Watts Up? Plug load meters, 2016. https://www.wattsupmeters.com/secure/products.php?pn = 0&wai = 322&more = 4, (Accessed: 09-09-16).
Redis, 2016. http://redis.io/ (Accessed:02-09-16).
Romansky, S., Hindle, A., 2014. On improving green mining for energy-aware software analysis. Proceedings of 24th Annual International Conference on Computer Science and Software Engineering. CASCON '14, IBM Corp, pp. 234–245.
Ruan, B., Huang, H., Wu, S., Jin, H., 2016. A performance study of containers in cloud environment. Springer International Publishing, Cham. 343–356
Santos, E. A., Mc Lean, C., Solinas, C., 2018a. Replication data for: "How does Docker affect energy consumption? Evaluating workloads in and out of Docker containers". https://doi.org/10.5281/zenodo.1227250.
Santos, E. A., McLean, C., Solinas, C., 2018b. eddieantonio/collect-data: replication package for "How Does docker affect energy consumption? Evaluating workloads in and out of Docker containers". https://doi.org/10.5281/zenodo.1227253.
Shea, R., Wang, H., Liu, J., 2014. Power consumption of virtual machines with network transactions: measurement and improvements. Proceedings of the IEEE INFOCOM 2014–Conference on Computer Communications. pp. 1051–1059.
The PostgreSQL Global Development Group, 2016. pgbench(1), postgreSQL. 9.5.4.
TPC, 1990. TPC-B. http://www.tpc.org/tpcb/, (Accessed: 25-01-17).
Tsung, 2016. http://tsung.erlang-projects.org/ (Accessed:07-09-16).
U.S. Energy Information Administration, 2017. Average retail price of electricity, annual. https://www.eia.gov/electricity/data/browser/#/topic/7?agg = 0,1&geo = vvvvvvvvvvvvo&endsec = vg&linechart = ELEC.PRICE.US-ALL.A&columnchart = ELEC.PRICE.US-ALL.A&map = ELEC.PRICE.US-ALL.A&freq = A&start = 2015&end = 2016&ctype = linechart&ltype = pin&rtype = s&pin = &rse = 0&maptype = 0(Accessed 11-01-18).
Vasić, N., Barisits, M., Salzgeber, V., Kostic, D., 2009. Making Cluster Applications Energy-Aware. Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds. ACDC 09, ACM, pp. 37–42.
W3Techs.com. Usage of content management systems for websites, 2016. https://w3techs.com/technologies/overview/content_management/all/, September (Accessed:02-09-16).
Installing WordPress-WordPress Codex, 2016. https://codex.wordpress.org/Installing_WordPress(Accessed: 22-12-16).
Xu, C., Zhao, Z., Wang, H., Shea, R., Liu, J., 2015. Energy efficiency of cloud virtual

machines: from traffic pattern and cpu affinity perspectives. IEEE Syst. J. PP 99, 1–11.
Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R.P., Mao, Z.M., Yang, L., 2010. Accurate online power estimation and automatic battery behavior based power model generation for Smartphones. Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardwaresoftware codesign and system synthesis. ACM, pp. 105–114.

Eddie Antonio Santos is a graduate student in the Department of Computing Science at the University of Alberta. Eddie's research focuses on how natural language processing tools can be leveraged to work on source code and software engineering artifacts. In his spare time, Eddie enjoys teaching beginners how to code, adventuring in the D&D multiverse, and jamming on bass guitar. Eddie received a BSc. in Computing Science from the University of Alberta. Contact him at easantos@ualberta.ca or https://www.eddieantonio.ca/.

Carson McLean is an undergraduate student in the Department of Computing Science at the University of Alberta. He is interested in the environmental and financial costs of computing. Carson intends to focus his Master's on machine learning, specifically the areas of representation and multimodal learning.



Christopher Solinas is a graduate student specializing in artificial intelligence and games at the University of Alberta. His main focus is building agents that can play games in which players have limited information, but he has been interested in software systems since his undergraduate studies. Prior to graduate school, Christopher spent some time working in industry as a software developer. In his spare time, Christopher enjoys playing soccer and trying to recreate authentic Chinese dishes in his home kitchen.

Abram Hindle is an associate professor of Computing Science at the University of Alberta. His research focuses on problems relating to mining software repositories, improving software engineering-oriented information retrieval with contextual information, the impact of software maintenance on software energy consumption, and how software engineering informs computer music. He likes applying machine learning in music, art, and science. Sadly, Abram has no taste in music and produces reprehensible sounding noise using his software development abilities. Abram received a PhD in computer science from the University of Waterloo, and Masters and Bachelors in Computer Science from the University of Victoria. Contact him at abram.hindle@ualberta.ca http://softwareprocess.ca.