How Data Center Size Impacts the Effectiveness of Dynamic Power Management

Anshul Gandhi and Mor Harchol-Balter

***Abstract*—Power consumption accounts for a significant portion of a data center's operating expenses. Sadly, much of this power is wasted by servers that are left on even when there is no work to do.**

Dynamic power management aims to reduce power wastage in data centers by turning servers off when they are not needed. However, turning a server back on requires a setup time, which can adversely affect system performance. Thus, it is not obvious whether dynamic power management should be employed in a data center.

In this paper, we analyze the effectiveness of dynamic power management in data centers under an M/M/k model via Matrix-analytic methods. We find that the effectiveness of even the simplest dynamic power management policy increases with the data center size, surpassing static power management when the number of servers exceeds 50, under realistic setup costs and server utilizations. Furthermore, we find that a small enhancement to traditional dynamic power management, involving delaying the time until a server turns off, can yield benefits over static power management even for data center sizes as small as 4 servers.

I. Introduction

Energy costs for data centers continue to rise every year, already exceeding \$19 billion annually and resulting in 11 million tons of CO_2 emissions [4]. The sad part is that much of this power is *wasted*. Servers are only busy less than 30% of the time on average [1], [2], [7], but they are often left on, while idle, utilizing 60% or more of peak power when in the idle state.

For example, for the Intel Xeon E5520 servers in our lab, power is burned at a rate of 200 Watts when running at peak frequency and 140 Watts when idle. If the average utilization of a data center is 30% and the data center is provisioned for peak utilization, then 70% of servers are sitting idle on average, but are still burning power at a rate of 140 Watts.

It seems that the solution should be simple: turn off servers when they are not in use. This is known as *dynamic power management*. However, turning off servers necessitates a high *setup overhead* to get them back on again, which is prohibitive for response times, and can waste yet additional power. As an example, for the Intel Xeon E5520 servers in our lab, the setup cost required to turn on a server which is off is 260 seconds, during which time power is burned at the peak rate of 200 Watts. This setup cost is particularly formidable in light of the fact that the typical job size (service requirement) is about 1 second. Given such a high setup cost (both in time and energy), it is not at all obvious whether one should turn off servers when idle.

The standard industry approach is to simply leave all servers on all the time, even if only 30% of the servers are being used on average. We refer to this *static power management* policy as `AlwaysOn`. `AlwaysOn` has the advantage of achieving low response times. While `AlwaysOn` seems to be very wasteful of power, it is not obvious that it actually uses more power than a policy which turns off idle servers, given a setup cost that is very high relative to job size and interarrival time.

In this paper, we start by defining a policy `On/Off` which turns servers off when they are idle, and we provide analytical results comparing `AlwaysOn` and `On/Off` in server farms. Our analysis is based on Matrix-analytic methods, which are numerical methods for analyzing complex Markov chains which repeat and are unbounded in only one dimension. We find that for smaller server farms, `On/Off` is often actually *worse* than `AlwaysOn` with respect to both mean response time and mean power, even when utilization is low (30% utilization). This leads us to speculate on whether `On/Off` ever makes sense. We therefore consider larger and larger server farms, while keeping the utilization fixed at 30%. We find that as the size of the server farm grows, `On/Off` becomes more and more desirable, eventually outperforming `AlwaysOn` by a factor of 2 or more in power, while achieving response times comparable to `AlwaysOn`.

One of the problems with `On/Off` for smaller server farms is that it is too quick to turn off servers when they are idle, resulting in a (big) setup cost when new work comes in (this is less of a problem in larger server farms because there it is more likely that there is a server available when new work comes in). To mitigate this problem, we introduce an enhancement to `On/Off`, which we call `DelayedOff`. The idea is simple: when a server goes idle, instead of turning off immediately, the server waits for some time t_{wait}, in case a job arrives. We develop an approximation for choosing t_{wait}, and then show (again using Matrix-analytic methods) that, using the right t_{wait}, the `DelayedOff` policy vastly improves upon the performance of `On/Off`. As with `On/Off`, we find that `DelayedOff` is most beneficial as the size of the server farm grows, however unlike `On/Off`, we find that `DelayedOff` is already superior to `AlwaysOn` for small server farms.

This work was supported by NSF:CNS-1116282 and by Intel Research, Pittsburgh.

Anshul Gandhi is a PhD student in the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 15213. anshulg@cs.cmu.edu

Mor Harchol-Balter is an Associate Professor of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 15213. harchol@cs.cmu.edu

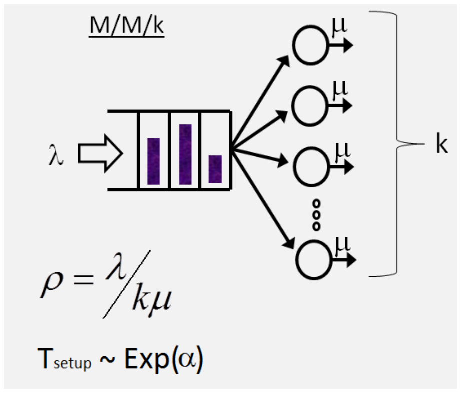


Fig. 1. Illustration of our model.

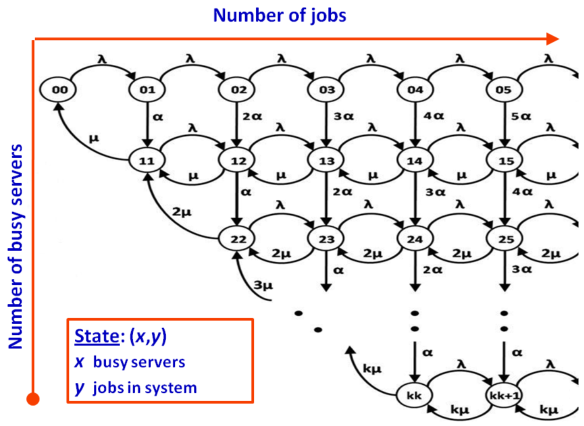


Fig. 2. Markov chain for `On/Off`. $T_{setup} \sim \text{Exp}(\alpha)$.

II. Model, Policies, and Metrics

A. Model

We assume a server farm with k servers, specifically, an M/M/k, where all servers are homogeneous, as shown in Figure 1. Job sizes (service times) are Exponentially-distributed with mean $\mathbf{E}[S] = \frac{1}{\mu}$, where μ is the speed of each server. Arrivals occur according to a Poisson Process with rate λ. The utilization of the server farm is:

$$\rho = \frac{\lambda}{k\mu}$$

Throughout the paper we set $\rho = 30\%$. The Setup time is denoted by random variable T_{setup}, which is Exponentially-distributed with rate α, where mean setup time is $t_{setup} = \frac{1}{\alpha}$. We typically assume $t_{setup} = 100$ seconds. Power is consumed at a rate of $P_{idle} = 140$ Watts when a server is idle, and consumed at a rate of $P_{max} = 200$ Watts when a server is busy or is in setup mode. We assume that zero time is required to turn off a server.

B. The policies

We consider one static power management policy and two dynamic power management policies:

- **AlwaysOn**: This policy assumes that all k servers are on all the time. A server may be either busy or idle.
- **On/Off**: Under this policy, servers are either off, in setup, or busy. When a new job arrives, if there is a server that is off, then the job puts that server into setup mode; if all k servers are already busy or in setup, then the job simply waits in the queue. Most servers do not require the full $\text{Exp}(\alpha)$ time to set up. The reason is that, whenever a busy server, b, frees up, if there is a job waiting on a server s to setup, the job will move to server b; at this point server s is given to a waiting job in the queue, or is shut off if there is no waiting job.
- **DelayedOff**: Under this policy, servers are either off, in setup, busy, or idle (in wait mode). This policy is identical to On/Off, except that when a server would normally turn off under On/Off, the server instead goes into a wait mode. The wait duration is denoted by the random variable T_{wait} which is Exponentially-distributed with rate β, where the mean wait duration is $t_{wait} = \frac{1}{\beta}$. If a job arrives during the T_{wait} period, the job takes over the waiting server, which now becomes busy; otherwise, the server is turned off after the T_{wait} period.

Figure 2 shows the Markov chain for `On/Off`. Each state takes the form (x, y), where x denotes the number of busy servers and y denotes the total number of jobs in the system. Clearly $y \geq x$. Observe that the number of servers in setup is then $\min\{y - x, k - x\}$.

Suppose for example, that we are in state $(2, 5)$, where $k = 10$. Then there are no jobs in the queue and there are 3 servers in setup. Hence with rate 3α one of the servers finishes setting up, moving us to $(3, 5)$. Alternatively, with rate 2μ, a job completes at one of the busy servers, so one of the 3 jobs in setup moves there and the server in setup shuts off, putting us in state $(2, 4)$. Finally, with rate λ a new job arrives which forces another server into setup mode, moving us to state $(2, 6)$.

Now suppose that we are in state $(2, 5)$, where $k = 4$. Then there is a job in the queue, and there are 2 servers in setup and 2 busy servers. Now, with rate 2α we move to $(3, 5)$; with rate 2μ, we move to $(2, 4)$; and with rate λ we move to state $(2, 6)$.

Figure 3 shows the Markov chain for `DelayedOff`. Here the state takes the form (x, y), where x denotes the number of servers that are busy or waiting to turn off, and y denotes the total number of jobs in the system. Observe that there are two distinct regions to the Markov chain. In the region where $y > x$, there are x servers which are busy, $y - x$ servers in setup, and no servers waiting to turn off. In the region where $y < x$, there are y servers busy, $x - y$ servers waiting to turn off, and no servers in setup.

The Markov chains in Figures 2 and 3 both have a finite number of rows, k, and an infinite number of columns. Importantly, both Markov chains *repeat* after the kth column. This repeating property allows us to analyze both of these chains via Matrix-analytic methods, which are numerical methods tailored to such chains (see [5]). Once we obtain

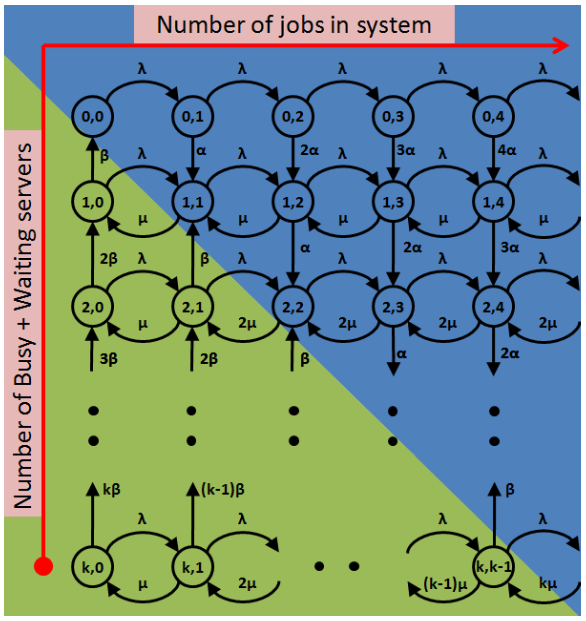


Fig. 3. Markov chain for `DelayedOff`. $T_{setup} \sim \text{Exp}(\alpha)$ and $T_{wait} \sim \text{Exp}(\beta)$.

the limiting probability for each state, $\pi_{x,y}$, we can easily compute mean response time and mean power consumption, as described in Section II-C.

C. Performance metrics

There are two performance metrics we care about: (i) mean response time, T_{avg}, defined as the time from when a job arrives until it completes service, and (ii) mean power consumption, P_{avg}, which takes into account that power is consumed at a different rate depending on what mode the server is in. These are obtained from the limiting probabilities of the above Markov chains as follows:

$$
\begin{aligned}
T_{avg}^{On/Off} &= \frac{1}{\lambda}\sum_{x,y} \pi_{x,y} \cdot y \\
P_{avg}^{On/Off} &= \sum_{x,y} \pi_{x,y} \cdot \min\{y,k\} \cdot P_{max} \\
T_{avg}^{DelayedOff} &= \frac{1}{\lambda}\sum_{x,y} \pi_{x,y} \cdot y \\
P_{avg}^{DelayedOff} &= \sum_{x,y} \pi_{x,y} \cdot (\min\{y,k\} \cdot P_{max} \\
&\qquad + \max\{x-y,0\} \cdot P_{idle})
\end{aligned}
$$

Clearly, if one only wants to minimize T_{avg}, then one should leave on as many servers as possible. On the other hand, if one only wants to minimize P_{avg}, then one should turn all servers off. A common metric which takes both these metrics into account is the Performance-per-Watt metric, PPW, defined as:

$$PPW = \frac{1}{T_{avg} \cdot P_{avg}}$$

Higher PPW is better since it says that either T_{avg}, or P_{avg}, or both are lower.

To compare our dynamic power management algorithms with the static `AlwaysOn` algorithm, we look at the Normalized Performance-per-Watt, $NPPW$, defined as the PPW for the dynamic algorithm in question, divided by PPW for `AlwaysOn`:

$$NPPW = \frac{PPW^{Dynamic}}{PPW^{\texttt{AlwaysOn}}}$$

Here "Dynamic" can denote either `On/Off` or `DelayedOff`. When NPPW exceeds 1, we say that the dynamic power management algorithm is superior to `AlwaysOn`. An $NPPW$ of 1.2 says that the dynamic algorithm improves upon `AlwaysOn` by 20%, while an $NPPW$ of 0.2 says that the `AlwaysOn` algorithm is 5 times better than the dynamic algorithm.

III. `AlwaysOn` VERSUS `On/Off`

This section compares `AlwaysOn` with `On/Off` for server farms of different sizes, where utilization is held fixed at $\rho = 30\%$.

A. A simple example

It should be obvious that `AlwaysOn` is superior to `On/Off` with respect to T_{avg}. What is far less obvious is what happens to P_{avg}. We start with a simple example to illustrate this point. Consider a small server farm with $k = 10$ servers, each of which serve jobs at a rate of $\mu = 1$ job/sec. Arrivals enter according to a Poisson Process with rate $\lambda = 3$ jobs/sec, so the server farm utilization is 30%. The setup time is Exponentially-distributed with mean $t_{setup} = \frac{1}{\alpha} = 100$ seconds.

Under the above configuration parameters, we find that `On/Off` performs *worse* than `AlwaysOn` both with respect to mean response time, T_{avg} and mean power, P_{avg}. The results of our analysis are shown in Figure 4.

B. The effect of job size and setup time

We now repeat the analysis from Figure 4, on a wider range of job sizes, ranging from $\mathbf{E}[S] = 1$ second to $\mathbf{E}[S] = 10$ seconds and a wider range of setup times, ranging from $t_{setup} = 20$ seconds to $t_{setup} = 100$ seconds. As before,

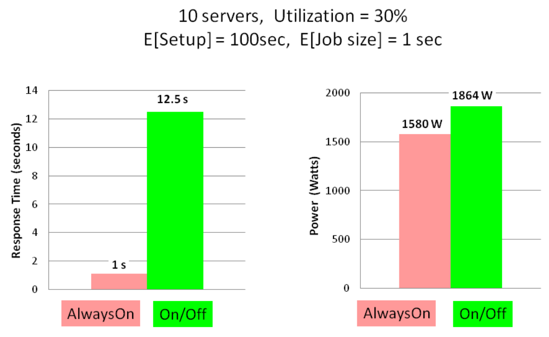


Fig. 4. Comparison of `AlwaysOn` and `On/Off` for $k = 10$, $t_{setup} = 100s$ and $\mathbf{E}[S] = 1s$. `On/Off` is worse for both T_{avg} and P_{avg}.

t_{setup} (secs) ↑	1	3	5	7	9
100	12.5	15.5	18.6	21.6	24.6
80	10.3	13.3	16.4	19.4	22.3
60	8.1	11.1	14.1	17.1	20.0
40	5.9	8.9	11.9	14.7	17.5
20	3.7	6.7	9.4	12.1	14.6

E[S] (secs) →

(a) $T_{avg}^{On/Off}$ (in s)

t_{setup} (secs) ↑	1	3	5	7	9
100	1864	1673	1542	1447	1374
80	1835	1618	1480	1382	1309
60	1791	1542	1396	1298	1227
40	1713	1427	1279	1185	1119
20	1542	1227	1093	1016	964

E[S] (secs) →

(b) $P_{avg}^{On/Off}$ (in watts)

t_{setup} (secs) ↑	1	3	5	7	9
100	4.10^{-5}	4.10^{-5}	3.10^{-5}	3.10^{-5}	3.10^{-5}
80	5.10^{-5}	5.10^{-5}	4.10^{-5}	4.10^{-5}	3.10^{-5}
60	7.10^{-5}	6.10^{-5}	5.10^{-5}	5.10^{-5}	4.10^{-5}
40	10.10^{-5}	8.10^{-5}	7.10^{-5}	6.10^{-5}	5.10^{-5}
20	17.10^{-5}	12.10^{-5}	10.10^{-5}	8.10^{-5}	7.10^{-5}

E[S] (secs) →

(c) $PPW^{On/Off}$ (in $(\text{s} \cdot \text{watts})^{-1}$)

1	3	5	7	9
1.0	3.0	5.0	7.0	9.0

E[S] (secs) →

(d) $T_{avg}^{AlwaysOn}$ (in s)

1	3	5	7	9
1580	1580	1580	1580	1580

E[S] (secs) →

(e) $P_{avg}^{AlwaysOn}$ (in watts)

1	3	5	7	9
63.10^{-5}	21.10^{-5}	13.10^{-5}	9.10^{-5}	7.10^{-5}

E[S] (secs) →

(f) $PPW^{AlwaysOn}$ (in $(\text{s} \cdot \text{watts})^{-1}$)

Fig. 5. Results for On/Off and AlwaysOn for $k = 10$ servers and $\rho = 30\%$.

t_{setup} (secs) ↑	1	2	3	4	5	6	7	8	9	10
100	0.07	0.13	0.18	0.23	0.28	0.32	0.35	0.39	0.42	0.45
90	0.07	0.14	0.20	0.25	0.30	0.34	0.38	0.42	0.45	0.48
80	0.08	0.16	0.22	0.28	0.33	0.37	0.41	0.45	0.49	0.52
70	0.09	0.18	0.24	0.31	0.36	0.41	0.45	0.49	0.53	0.56
60	0.11	0.20	0.28	0.34	0.40	0.45	0.50	0.54	0.58	0.62
50	0.13	0.23	0.32	0.39	0.45	0.51	0.56	0.60	0.64	0.68
40	0.16	0.28	0.37	0.45	0.52	0.58	0.63	0.68	0.73	0.77
30	0.20	0.34	0.45	0.54	0.62	0.68	0.74	0.79	0.84	0.88
20	0.28	0.45	0.58	0.68	0.77	0.84	0.90	0.96	1.01	1.05
10	0.45	0.68	0.84	0.96	1.05	1.14	1.20	1.26	1.32	1.37

E[S] (secs) →

(a) $k = 10$

t_{setup} (secs) ↑	1	2	3	4	5	6	7	8	9	10
100	0.15	0.26	0.36	0.43	0.50	0.56	0.61	0.65	0.69	0.73
90	0.16	0.29	0.38	0.46	0.53	0.59	0.64	0.69	0.73	0.77
80	0.18	0.31	0.42	0.50	0.57	0.63	0.68	0.73	0.78	0.82
70	0.20	0.34	0.45	0.54	0.61	0.68	0.73	0.78	0.83	0.87
60	0.23	0.38	0.50	0.59	0.67	0.73	0.79	0.84	0.89	0.93
50	0.26	0.43	0.56	0.65	0.73	0.80	0.86	0.91	0.96	1.00
40	0.31	0.50	0.63	0.73	0.82	0.89	0.95	1.00	1.05	1.09
30	0.38	0.59	0.73	0.84	0.93	1.00	1.06	1.12	1.17	1.21
20	0.50	0.73	0.89	1.00	1.09	1.17	1.23	1.28	1.33	1.38
10	0.73	1.00	1.17	1.28	1.38	1.45	1.51	1.57	1.61	1.65

E[S] (secs) →

(b) $k = 25$

t_{setup} (secs) ↑	1	2	3	4	5	6	7	8	9	10
100	0.26	0.43	0.55	0.64	0.72	0.79	0.84	0.90	0.94	0.98
90	0.28	0.46	0.58	0.68	0.76	0.83	0.88	0.94	0.98	1.02
80	0.31	0.49	0.62	0.72	0.80	0.87	0.93	0.98	1.03	1.07
70	0.34	0.53	0.67	0.77	0.85	0.92	0.98	1.03	1.08	1.12
60	0.38	0.58	0.72	0.83	0.91	0.98	1.04	1.10	1.14	1.19
50	0.43	0.64	0.79	0.90	0.98	1.05	1.12	1.17	1.22	1.26
40	0.49	0.72	0.87	0.98	1.07	1.14	1.21	1.26	1.31	1.35
30	0.58	0.83	0.98	1.10	1.19	1.26	1.32	1.37	1.42	1.46
20	0.72	0.98	1.14	1.26	1.35	1.42	1.48	1.53	1.58	1.61
10	0.98	1.26	1.42	1.53	1.61	1.68	1.74	1.78	1.82	1.85

E[S] (secs) →

(c) $k = 50$

Fig. 6. $NPPW$ of AlwaysOn vs. On/Off for a range server farm sizes, where $\rho = 30\%$. Lighter regions indicate the superiority of On/Off over AlwaysOn.

we still assume a small server farm of $k = 10$ servers only. Figures 5(a), 5(b) and 5(c) show the performance of On/Off under all these settings with respect to mean response time, mean power, and PPW respectively. Observe that the response time numbers of On/Off are much lower than the average setup time, indicating that servers typically do not wait anywhere near their full setup time, and yet jobs are clearly delayed, with slowdown factors as high as 12.5. Figures 5(d), 5(e) and 5(f) show the corresponding performance of AlwaysOn (note that AlwaysOn is not affected by t_{setup}). We see that only when setup time is very low ($\leq$ 20 seconds) and job size is relatively high ($\geq$ 9 seconds) does the PPW for On/Off approach that of AlwaysOn (and actually surpasses AlwaysOn). This is a little surprising because the benefits with respect to power savings are apparent throughout most of the space, for larger job sizes. The problem is that response time is heavily penalized under On/Off, due to the setup overhead.

Low setup times clearly favor On/Off because there is less overhead involved in turning the server on. High job sizes also favor On/Off, because when utilization is fixed and job sizes increase, then interarrival times also increase, making it more advantageous to turn off servers.

C. The effect of increasing server farm size

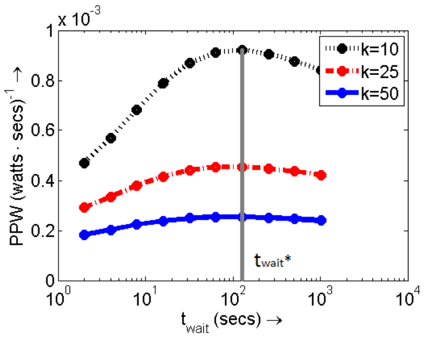
We now show the effect of increasing the size of the server farm. Figure 6 considers server farms of size (a) $k = 10$, (b) $k = 25$, and (c) $k = 50$. Throughout, utilization is held constant at $\rho = 30\%$. A range of values of job size and setup times is shown. The metric shown is:

$$NPPW = \frac{PPW^{\texttt{On/Off}}}{PPW^{\texttt{AlwaysOn}}}$$

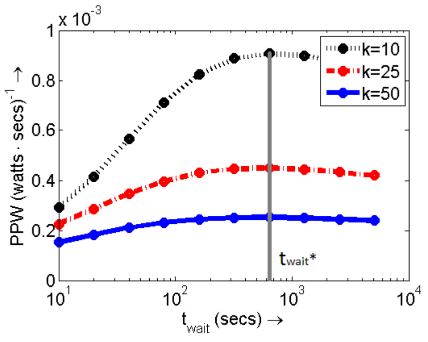
Hence a value greater than 1 implies that On/Off is superior to AlwaysOn.

While in the case of $k = 10$ servers NPPW is almost always below 1, in the case of $k = 50$, NPPW is almost always above 1. The superiority of On/Off clearly increases as the size of the server farm goes up. In fact, for the case of 50 servers, for about a third of the region, On/Off improves upon AlwaysOn by more than 25%.

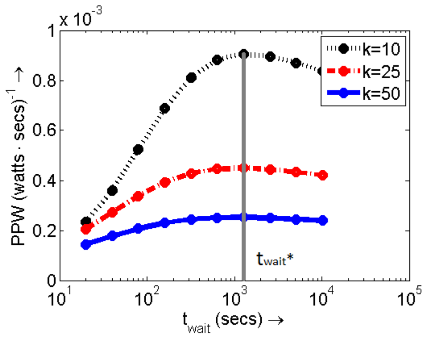
It may not be obvious why On/Off suddenly appears superior to AlwaysOn when the size of the server farm increases, given that utilization is constant, and hence the arrival rate has increased proportionately to the increase in server farm size. The answer lies in response time. Recall that the problem for On/Off in the case of $k = 10$ servers was not its power usage, which was often superior to that of AlwaysOn, but rather its response time, which was heavily influenced by the setup time. In a larger server farm, even with the proportional increase in the arrival rate, an arrival is much more likely to quickly end up being served (with many more busy servers, it is more likely that one will quickly free up quickly). Hence, while On/Off will never have as low a response time as AlwaysOn, the response time of On/Off approaches that of AlwaysOn as k increases, and, coupled with superior power consumption, the result is a higher PPW for On/Off.



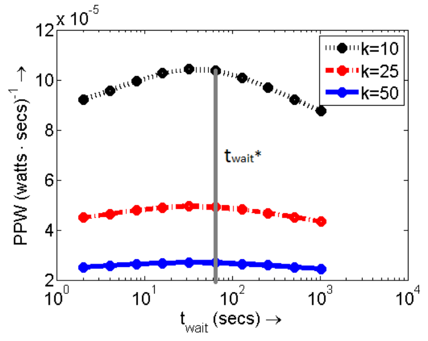
(a) $t_{setup} = 10s$, $\mathbf{E}[S] = 1s$



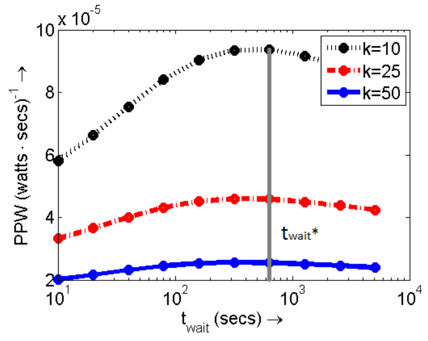
(b) $t_{setup} = 50s$, $\mathbf{E}[S] = 1s$



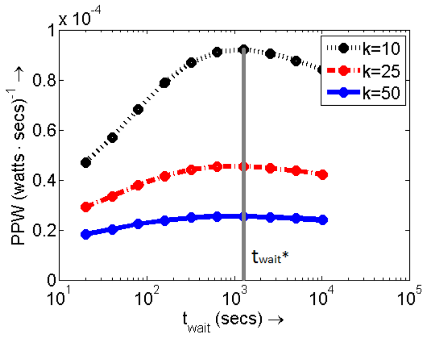
(c) $t_{setup} = 100s$, $\mathbf{E}[S] = 1s$



(d) $t_{setup} = 10s$, $\mathbf{E}[S] = 10s$



(e) $t_{setup} = 50s$, $\mathbf{E}[S] = 10s$



(f) $t_{setup} = 100s$, $\mathbf{E}[S] = 10s$

Fig. 7. In all configurations, $t^*_{wait} \approx 10 \cdot t_{setup}$, regardless of k or $\mathbf{E}[S]$.

IV. The benefits of DelayedOff

The idea behind DelayedOff is to try to mitigate the damage caused by large setup times by simply not turning the servers off in the first place. Of course if we never turn the servers off, we are left with AlwaysOn. The key is to turn servers off less frequently. Whenever a server would normally shut off, it now waits for some T_{wait} time (with mean t_{wait}) before shutting off. The DelayedOff algorithm is described in Section II-B. Surprisingly, as we'll show, this very simple idea has huge impact in the performance of dynamic power management.

We start in Section IV-A with a discussion of how long a timer to use and derive a heuristic for the optimal t_{wait}. We then derive the performance of DelayedOff using this t_{wait} value and show its performance in Section IV-B.

A. How long to wait: optimizing t_{wait}

The performance of DelayedOff is clearly affected by t_{wait}. When t_{wait} is very small, the behavior of DelayedOff is like that of On/Off. When t_{wait} is huge, the behavior of DelayedOff becomes that of AlwaysOn. However DelayedOff can actually be far better than either of these policies if t_{wait} is chosen carefully.

Figure 7 shows t^*_{wait}, the optimal value of t_{wait} for a range of values of server farm sizes, k, mean job size, $\mathbf{E}[S]$, and mean setup time, t_{setup}. Surprisingly, of all these parameters, the only one with a strong effect on t^*_{wait} is the setup time. We find that a heuristic rule of thumb that appears to work under our full range of parameters is:

$$t^*_{wait} \quad = \quad 10 \cdot t_{setup} \tag{1}$$

Some justification for rule of thumb (1) comes from the observation that t_{wait} should be chosen so that the extra energy consumed by leaving the server on for t_{wait} extra seconds should equal the energy spent in setting up a new server. That is:

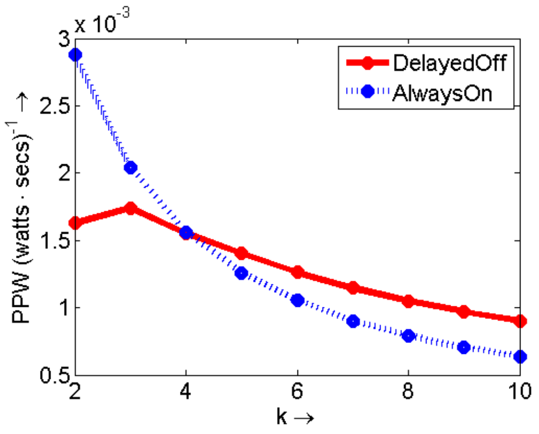
$$140 \cdot t_{wait} = 200 \cdot t_{setup}$$

This says that t_{wait} should be directly proportional to t_{setup}, although it does not explain the factor of 10. We believe that the heuristic rule of thumb will change based on the server farm utilization. As the server farm utilization goes up, we might want to have more spare servers to absorb occasional spikes in request rate. Thus, the t^*_{wait} should go up. When we increased the server farm utilization to about 70%, we found that the desired ratio of $\frac{t^*_{wait}}{t_{setup}}$ went up from 10 to about 100, indicating that it was beneficial to have more spare servers.

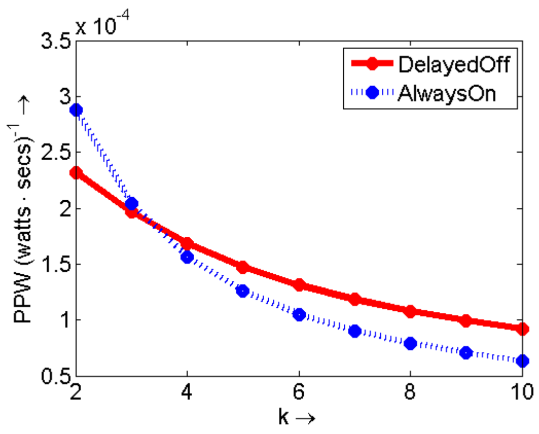
B. The effect of increasing server farm size

We now assume that t_{wait} is set according to Eq. (1) and we derive the PPW for DelayedOff. Figure 8 shows PPW for both DelayedOff and AlwaysOn as a function of server farm size, k.

As in the case of On/Off, we find that as k increases, DelayedOff improves upon AlwaysOn by bigger and bigger factors. What is significant however, is that the cross-over point where DelayedOff first surpasses AlwaysOn



(a) $\mathbf{E}[S] = 1$



(b) $\mathbf{E}[S] = 10$

Fig. 8. `AlwaysOn` vs. `DelayedOff` for different server farm sizes for $t_{setup} = 100$.

occurs already when k is very low. In fact, for a mean job size of $\mathbf{E}[S] = 1$ second, the cross-over point is $k = 4$, and for a mean job size of $\mathbf{E}[S] = 10$ seconds, the cross-over point is $k = 3$! For relatively small server farms with $k = 10$, we already see that `DelayedOff` outperforms `AlwaysOn` by more than 50%, even with a high setup time of $t_{setup} = 100$ seconds. This type of improvement wasn't visible under `On/Off` until we got to $k = 50$ servers, and even there we required that the setup time was much lower and job sizes were large.

V. Conclusion and Future work

In this paper we examine the effectiveness of dynamic power management policies in data centers. We find that dynamic power management policies are very effective when the setup time is small or when the job size is large. Further, we find that the effectiveness of dynamic power management policies increases with the size of the data center. In fact, while the dynamic power management policies considered in this paper seem ineffective for small data centers, these same policies outclass widely employed static policies when the data center size is large.

While there has been some recent work (see, for example, [3], [6]) in the area of analyzing dynamic power management policies, there is still a long way to go. In particular, while the authors in [3] were able to derive closed-form approximations for T_{avg} and P_{avg} for `On/Off`, no closed-form solutions or approximations have yet been found for `DelayedOff`.

References

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[2] L. A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[3] Anshul Gandhi, Mor Harchol-Balter, and Ivo Adan. Server farms with setup costs. *Perform. Eval.*, 67:1123–1138, November 2010.

[4] Green Grid. Unused Servers Survey Results Analysis. http://www.thegreengrid.org/en/Global/Content/white-papers/UnusedServersSurveyResultsAnalysis, 2010.

[5] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, Philadelphia, 1999.

[6] Isi Mitrani. Managing performance and power consumption in a server farm. *3rd Madrid Conference on Queueing Theory*, 2010.

[7] Neil MacDonald. Addressing the most common security risks in data center virtualization projects, 2010.