Distributed, Robust Auto-Scaling Policies for Power Management in Compute Intensive Server Farms

Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan
Carnegie Mellon University

Michael A. Kozuch
Intel Labs Pittsburgh

***Abstract*—Server farms today often over-provision resources to handle peak demand, resulting in an excessive waste of power. Ideally, server farm capacity should be dynamically adjusted based on the incoming demand. However, the unpredictable and time-varying nature of customer demands makes it very difficult to efficiently scale capacity in server farms. The problem is further exacerbated by the large setup time needed to increase capacity, which can adversely impact response times as well as utilize additional power.**

In this paper, we present the design and implementation of a class of Distributed and Robust Auto-Scaling policies (DRAS policies), for power management in compute intensive server farms. Results indicate that the DRAS policies dynamically adjust server farm capacity without requiring any prediction of the future load, or any feedback control. Implementation results on a 21 server test-bed show that the DRAS policies provide near-optimal response time while lowering power consumption by about 30% when compared to static provisioning policies that employ a fixed number of servers.

I. Introduction

Motivation: While energy costs of data centers continue to double every 5 years [21], unfortunately, most of this energy is wasted. Servers are only busy 10-30% of the time on average [3, 5], but they are often left on, while idle, utilizing 60% or more of peak power. Ideally, servers should be turned off when they are not in use. However, turning servers back on incurs a high *setup time* (time to go from off to on), which is prohibitive for response times, and can waste yet additional power. For example, the Intel E5520 servers running in our lab have a setup time of about 250 seconds, during which time they consume about 213 Watts of power. Given this high setup cost, it is not at all obvious whether one should ever turn idle servers off. As setup costs continue to drop, turning servers off looks more desirable, although even under lower setup costs it is still not clear whether turning servers off makes sense.

The power management problem: We ask: At any point in time, how many servers should be on and how many should be off? Our goal is to minimize both the average response time, T_{avg}, and the average power consumption, P_{avg}. *Response time* for a job is defined as the time from when the job arrives into the system to the time when it departs the system. The problem of power management is particularly difficult in situations where we *don't know the load in advance*, which is the situation we assume throughout this paper. In the unpredictable load scenario, leaving all servers on all the time is not a good option, because it requires provisioning for (at least) the peak load and wasting a lot of power [3, 15].

Prior work: There are two common approaches to handling time-varying loads. The first is to try to *predict* the future load, based on the past arrival rate, and adapt capacity based on this predicted load, see, e.g., [8, 9, 15]. The second approach is to use control-theoretic techniques to *reactively adapt* to the load, see, e.g., [10, 16, 19, 20, 22]. While both these approaches are common in research, they have two practical disadvantages. First, they can be very complex, requiring lots of frequent calculation. Second, it is tough to be predictive *or* reactive with good accuracy when setup times are high.

Our solution: We explore a broad class of distributed and robust auto-scaling policies, which we refer to as DRAS policies, that dynamically adjust the number of servers without requiring any prediction of the future load, or any feedback control. Each of our DRAS policies comprises three important design decisions. (i) *When should idle servers be turned off? (see Section III)* An obvious solution is turn off a server as soon as it goes idle. However, we find that it is beneficial to delay turning servers off for some time, $t_{wait} > 0$. (ii) *How should incoming jobs be routed to servers? (see Section IV)* Given that some servers will be idle because of the non-zero t_{wait}, when a job arrives, it is not obvious to which idle server the job should be routed. For example, one might route the job to a random idle server, or to a server that was most recently busy or least recently busy? It turns out that the right routing policy is vital to the robustness of the DRAS policies. (iii) *When should servers be turned on? (see Section V)* When a job arrives and finds no idle servers, it is natural to turn on a new server. However, if a burst of jobs arrives into the system, this will result in a lot of servers being turned on, some of which might not be needed later. We find that it helps to delay turning on a server until enough jobs, f, accumulate. In order to assess the impact of each of the three design decisions, we compare the T_{avg} and P_{avg} of DRAS against those of a straw man approach that is quite popular in the real-world [3, 5, 15], ON/IDLE. Under ON/IDLE, a fixed number of servers are always kept on. We optimistically assume that ON/IDLE has clairvoyant knowledge of the peak demand that it is going to face, and furthermore, we assume that ON/IDLE is "smart", and provisions for the peak demand using the well accepted "square-root staffing" rule[1] from [13]. While ON/IDLE results in near-optimal response times, its P_{avg} can be very high. Figure 1, which summarizes our implementation results, shows that DRAS can reduce average power consumption by about 30% in exchange for a 4% increase in response time. The 30%

[1]Under the square-root staffing rule, the number of servers is held fixed at $d + \sqrt{d}$, where d represents the number of servers needed to just meet the peak demand.

978-0-7695-4650-6/11 $26.00 © 2011 IEEE
DOI 10.1109/OCS.2011.6

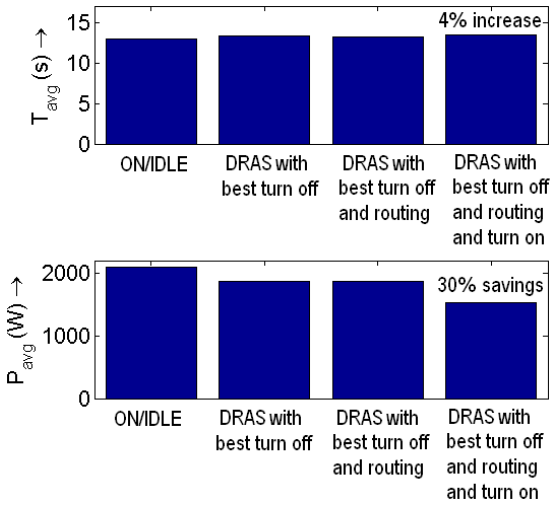


Fig. 1: *Impact of the three design decisions in DRAS based on implementation results in Figure 8.*

power savings can be attributed to using the right policy for turning servers off, which contributes 14% to power savings, and using the right policy for turning servers on, which contributes an additional 16% to power savings. While the right routing policy does not provide significant power savings, we find that it is vital for making DRAS robust.

II. Model

Figure 2 illustrates our server farm model. We assume a homogenous server farm, with CPU bound jobs, where each server only serves one job at a time, and the remaining jobs wait in a central queue. Each server can be in one of the following states: busy, idle, setup, or off. The associated power values are P_{busy}, P_{idle}, P_{setup} and P_{off}. The setup time, which we defined as the time to go from off to on (busy or idle), is denoted by t_{setup}, and is assumed to be a constant. Remaining details about turning servers off, routing jobs to servers, and turning servers on can be found in Sections III, IV and V respectively. For most of the results in this paper, we use a scaled NLANR [1] demand trace (shown later in Figure 8). We also evaluated our policies (see Table I) against other traces, such as other NLANR traces and the 1998 Soccer World Cup website trace [2], and observed similar results.

A. Implementation details

In order to experimentally evaluate the power management of compute intensive server farms via the DRAS policies, we built an implementation test bed. Experimental results based on this test bed are presented in Section VI (specifically, Figure 8 and Table I). Our test bed consists of 21 servers from the Open Cirrus cluster [4] at Intel Labs Pittsburgh. Each server has two 2.27 GHz quad-core Intel Xeon E5520 processors, with 16 GB of memory. We monitor the power consumption of individual servers by reading the power values off of the power distribution unit (PDU). We use SNMP [17] to remotely turn these servers on and off. The workload used in all our experiments is Intel's LINPACK [12] workload, which is CPU bound. Unless otherwise specified, we set LINPACK's

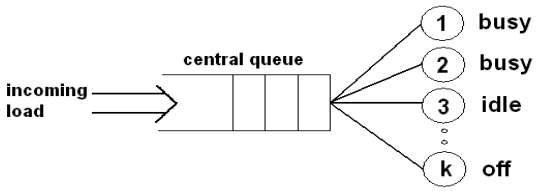


Fig. 2: *Illustration of our server farm model.*

job size to be 13 seconds. One of the servers is employed as the front-end load generator and load balancer. Each new job generates an HTTP request using the httperf [18] web server. The request is then routed to one of the 20 application servers, which then executes the LINPACK benchmark.

B. Simulation details

Ideally, we would like to explore the individual design decisions involved in the DRAS policies via implementation. However, each evaluation would require 24 hours of experimentation time (see Figure 8). Thus, in Sections III, IV and V, we resort to careful simulations to explore the design decisions, and then validate all our findings using our implementation test bed in Section VI. Our discrete event simulator, written in C++, uses the above described model of a compute intensive server farm. Using the simulator, we can experiment with various arrival traces, job size distributions, setup times, as well as arbitrarily large server farms. We use the following server characteristics for our simulation: $P_{busy} = P_{setup} = 213W$, $P_{idle} = 140W$ and $P_{off} = 9W$ (when the server is off, some components are kept powered on, such as the NIC). These values are based on measurements we obtained for our Intel Xeon E5520 servers running the CPU-bound LINPACK [12] workload. While for our test bed we measured $t_{setup} = 250s$, we also run simulations with lower t_{setup} values, such as $25s$ and $2.5s$.

III. When to turn servers off?

The first design decision we address for our DRAS policies is when should idle servers be turned off. In particular, should idle servers be immediately turned off to save power? Or should we leave them idle for some time, say t_{wait} seconds, in anticipation of new arrivals? We assume that an incoming job that does not find any idle server upon arrival, turns on an off server. The job then waits in the central queue, and once it reaches the head of the queue, it is routed to one of the idle servers, at random. If there is no idle server, the job waits until a server becomes idle, or until a server is turned on (whichever happens first). In Section IV, we relax the random routing assumption, and explore other routing policies and their impact on T_{avg} and P_{avg}. Then, in Section V, we relax the assumption that every arrival turns on a server if it does not find an idle server, and explore other alternatives for when to turn servers on.

Figure 3 shows our simulation results for average response time (T_{avg}) and average power consumption (P_{avg}) as a function of t_{wait} for different setup times (t_{setup}), on a 20 server system using the NLANR [1] demand trace (shown later in Figure 8) *We find that it is best to wait for some time before turning idle servers off.* Each server independently

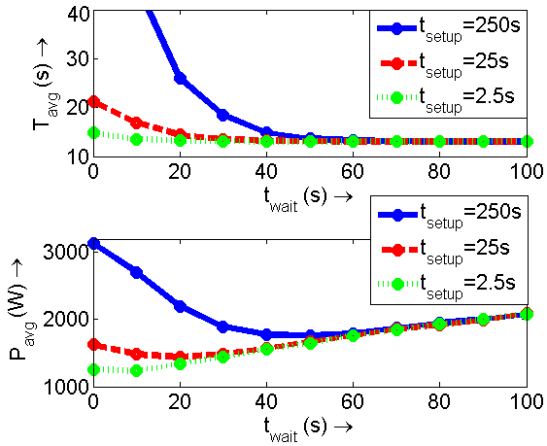


Fig. 3: *Waiting for some time before turning idle servers off can reduce response times (top graph) by as much as 83% and power consumption (bottom graph) by as much as 40%.*

sets a t_{wait} timer when it goes idle (no central control is needed). In particular, for $t_{setup} = 250s$, the best t_{wait} setting reduces response time by as much as 83% (from $80s$ when $t_{wait} = 0$ to $13.5s$ under the optimal t_{wait}) and reduces power consumption by as much as 40% (from $3000W$ to $1800W$) when compared to the case where $t_{wait} = 0$. The reason behind this is that if we immediately turn idle servers off, then new arrivals won't find an idle server. Thus, new arrivals will have to spend a lot of time waiting for a busy server to become idle or for a server to turn on (the latter is proportional to t_{setup}). Note that the best t_{wait} setting increases with t_{setup}. This result was also observed in [11], where the authors used analytical modeling to determine whether idle servers should be immediately turned off or not.

The idea of leaving a server purposely idle for some time before shutting it off has been used before in the mobile computing community (see [6, 7, 14]), however, *only for a single device*. For a multi-server system such as the one we consider, we find that the right t_{wait} setting is greatly affected by the routing policy used to dispatch jobs to servers, as discussed in the next section.

IV. How to route jobs to servers?

In the last section, we saw that leaving servers idle for some time (t_{wait}) was beneficial to response time and power. Given that an arrival finds some idle servers, it is not obvious which idle server the arrival should go to. One possibility is to send the arrival to a random idle server (RANDOM), as we did in the previous section. We now introduce two other routing policies. Under the Most-Recently-Busy (MRB) routing, the job at the head of the queue is routed to the idle server that was most recently busy. That is, the server that has been idle for the shortest amount of time. We also consider the Least-Recently-Busy (LRB) policy which sends the job at the head of the queue to the idle server that has been idle for the longest amount of time. It is not clear which routing policy is the best, and further, how the routing policy impacts response time and power. Figure 4 compares T_{avg} and P_{avg} for the MRB, RANDOM and the LRB policies. We find that the routing policy does not significantly affect response time. For power consumption, each policy, when run at its optimal t_{wait} setting, results in approximately the same power consumption. However, different policies have different optimal t_{wait} settings ($120s$ for MRB, $80s$ for RANDOM and $40s$ for LRB), and some policies (like LRB) are more sensitive to t_{wait}, whereas MRB is largely insensitive beyond a t_{wait} of $120s$. The lowering of the optimal t_{wait} setting as we go from MRB to LRB can be reasoned as follows. Under MRB routing, servers that have been idle for a long time are very likely to complete their t_{wait} timeout and be turned off, without being interrupted for servicing a job. However, under the LRB routing, an idle server can only turn off if no new jobs arrive for at least t_{wait} seconds. Now, if the t_{wait} setting is large, the LRB routing policy will waste a lot of power by not allowing idle servers to turn off, and thus, LRB favors a lower t_{wait} setting.

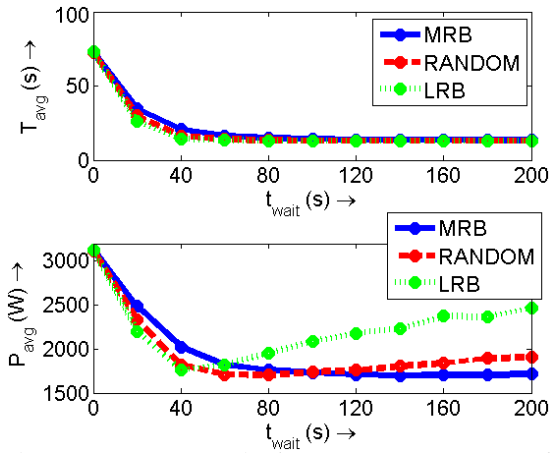


Fig. 4: *MRB is insensitive for large t_{wait}. RANDOM and LRB are very sensitive to t_{wait}.*

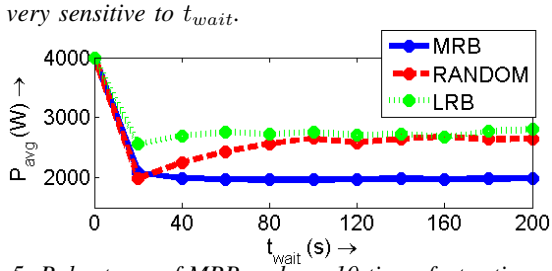


Fig. 5: *Robustness of MRB under a 10 times faster time scale.*

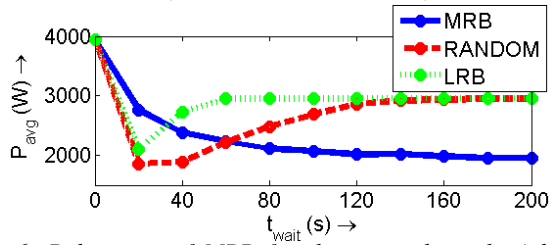


Fig. 6: *Robustness of MRB for the case where the job size distribution is as observed in the NLANR trace.*

Observe that operating at the optimal t_{wait} setting is crucial for power savings under the RANDOM and LRB routing policies. Any deviation from the optimal t_{wait} setting can

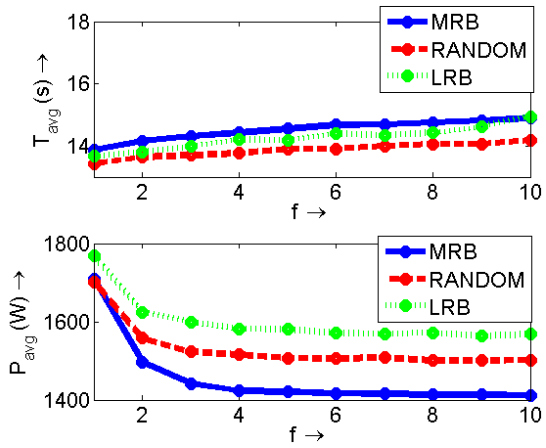


Fig. 7: *Dampening helps reduce P_{avg} at the expense of a slight increase in T_{avg}.*

result in a significant increase in power consumption. On the other hand, MRB routing is largely insensitive to the t_{wait} setting, as long as $t_{wait} > 120s$. This suggests that the MRB routing is quite robust. We find that MRB is robust even under different time scales. For example, consider Figure 5, where the arrival rate is scaled up by a factor of 10, and the job sizes are correspondingly scaled down by a factor of 10, to maintain the same load. While the routing policies again do not affect response time significantly (not shown here), we find that the power consumption is affected. In particular, for the RANDOM and LRB policies, the optimal t_{wait} is now around $20s$, which is different from what we had in Figure 4. Also, operating at the optimal t_{wait} setting is again crucial for power savings under the RANDOM and LRB routing policies. However, the MRB policy is again largely insensitive to the t_{wait} setting, as long as $t_{wait} > 120s$. We find similar results when we use a different job size distribution as well. For example, consider Figure 6, where we use the job size distribution observed in the NLANR trace as opposed to a constant job size. Again, the MRB policy is largely insensitive to the t_{wait} setting, as long as $t_{wait} > 120s$. Thus, if the arrival rate or job size distribution is not known beforehand (which is commonly the case), it is much better to use the robust MRB routing policy. We arrived at a similar conclusion when we evaluated the other traces mentioned in Section II-B as well.

V. When to turn servers on?

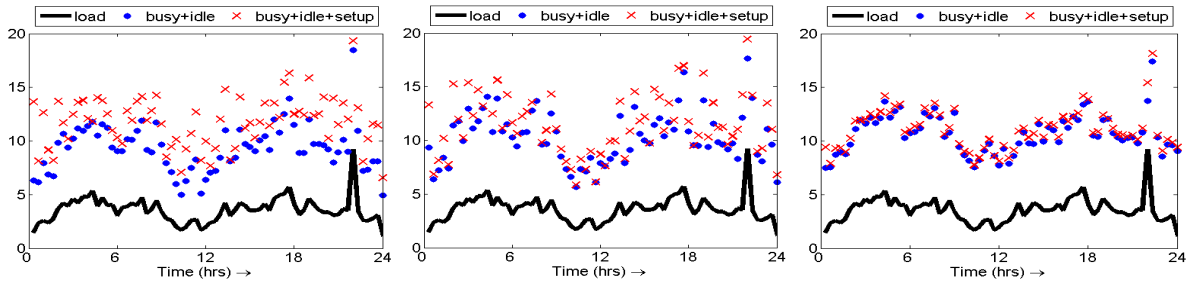
The third design decision we address for our DRAS policies is when servers should be turned on. In the previous sections, we assumed that each arrival into the system that does not find an idle server will turn on an off server. However, if a burst of jobs arrives into the system and finds no idle server, then a lot of servers will be turned on, resulting in a huge power draw, with most of these servers not being needed later. Thus, it might help to delay turning a server on until enough jobs have accumulated in the queue. We refer to this idea as "dampening", and use f to denote the number of jobs that need to accumulate before a server is turned on. Note that once a server has been turned on (after f jobs have accumulated), another server will not be turned on until an additional f jobs have accumulated. Figure 7 shows the effect of f on T_{avg} and P_{avg} for the different routing policies, which were run at their respective optimal t_{wait} settings. We see that for all routing policies, *dampening helps reduce P_{avg} at the expense of a slight increase in T_{avg}*. In particular, for MRB, setting $f = 4$ reduces power consumption by 16% at the expense of only a 4% increase in response time, for our workload. Also, for MRB, when $f = 1$, there were 394 instances of servers being turned on during the entire 24 hour trace, out of which 243 servers (62%) were turned off without serving any jobs. However, when $f = 4$, there were only 131 instances of servers being turned on, out of which only 36 servers (27%) were turned off without serving any jobs. Thus, with dampening, fewer servers are turned on needlessly.

Trace	Job Size	ON/IDLE		DRAS	
		T_{avg}	P_{avg}	T_{avg}	P_{avg}
WC'98 [2]	13s	13s	2034W	13.8s	1486W
WC'98 [2]	5s	5s	1988W	5.4s	1648W
nlanr1 [1]	13s	13.1s	2040W	13.6s	1256W
nlanr2 [1]	13s	13.1s	2124W	13.5s	1621W
nlanr3 [1]	5s	5.1s	1696W	6s	953W

TABLE I: Additional implementation results.

VI. Implementation results

We now present our implementation results for DRAS based on our 20-server test bed described in Section II-A. As discussed in Section I, we compare against a clairvoyant and "smart" static provisioning policy, ON/IDLE. Figure 8 shows the 24-hour long time-series results, along with T_{avg} and P_{avg} numbers, for (a) DRAS with the random routing and best t_{wait}, (b) DRAS with MRB routing and best t_{wait}, and (c) DRAS with MRB routing, best t_{wait} and the best dampening. Each plot shows the measured load (arrival rate times the job size) or incoming work (solid line), number of servers busy or idle (dots) and the number of servers busy or idle or in setup (crosses). The "smart" ON/IDLE policy (not shown here) provides excellent response times ($T_{avg} = 13.1s$), but wastes a lot of power ($P_{avg} = 2098W$) by over provisioning server capacity, especially during periods of low load. On the other hand, the DRAS policies provide near-optimal response times ($T_{avg} < 14s$) and great power-savings when compared to ON/IDLE. This is because the DRAS policies *automatically* adjust the server capacity (servers busy or idle) based on the incoming load. For example, we see that all the DRAS policies shown in Figure 8 lower the server capacity around the 12 hour mark, when the load decreases, and increase server capacity around the 18 hour mark, when the load increases. DRAS with the best dampening (best turn on) provides additional power savings by reducing the number of servers in setup (the difference between the crosses and the dots in the plots). Overall, DRAS provides nearly 30% power savings at the expense of a 4% increase in response time. We also evaluated DRAS (with the best settings) for other arrival traces and/or job sizes, as shown in Table I, and found similar improvements, with power savings in the range of 17%-44% at the expense of a 3%-17% increase in response time.



(a) DRAS with best turn off. $T_{avg} = 13.4s$, $P_{avg} = 1876W$. (b) DRAS with best turn off and routing. $T_{avg} = 13.2s$, $P_{avg} = 1873W$. (c) DRAS with best turn off, routing and turn on. $T_{avg} = 13.5s$, $P_{avg} = 1532W$.

Fig. 8: *Implementation results showing DRAS.*

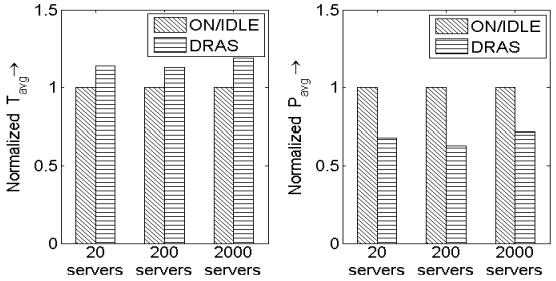


Fig. 9: *Simulation results for ON/IDLE vs. DRAS under different server farm sizes.*

Thus, our implementation results successfully demonstrate the superiority of DRAS over ON/IDLE. While our experimental test bed is limited to 20 servers, we can simulate much larger server farms. Figure 9 compares ON/IDLE against DRAS for various server farm sizes in simulation using the trace shown in Figure 8. We find that even for larger server farms (200 or 2000 servers), DRAS provides significant power savings without compromising much on response times.

VII. Conclusion

In this paper we presented the design and implementation of the DRAS policies, a class of distributed and robust auto-scaling policies, for power management in compute intensive server farms. The DRAS policies can provide near-optimal response times while lowering power consumption by 30% when compared to existing server farm policies. Via simulations and implementation on a 20 server test bed, we show that the DRAS policies dynamically adjust server farm capacity without requiring any prediction of the future load or any feedback control, and are robust to variations in job size distribution and time scale. While we only experimented with the off state in this paper, we plan to experiment with the various sleep states that might soon be available in commercial servers as part of future work. We also plan to extend our current work to address power management problems in multi-tier web server farms, involving a mix of CPU and I/O bound jobs, and time-sharing servers.

References

[1] National Laboratory for Applied Network Research. Anonymized access logs. Available at ftp://ftp.ircache.net/Traces/.

[2] The internet traffic archives: WorldCup98. Available at http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, University of California, Berkeley, Feb 2009.

[4] A. I. Avetisyan, R. Campbell, I. Gupta, M. T. Heath, S. Y. Ko, G. R. Ganger, M. A. Kozuch, D. O'Hallaron, M. Kunze, T. T. Kwan, K. Lai, M. Lyons, D. S. Milojicic, H. Y. Lee, Y. C. Soh, N. K. Ming, J. Y. Luke, and H. Namgoong. Open Cirrus: A Global Cloud Computing Testbed. *IEEE Computer*, April 2010.

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

[6] L. Benini, A. Bogliolo, and G. De Micheli. System-level Dynamic Power Management. In *IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, pages 23 – 31, 1999.

[7] F. Chen, S. Jiang, W. Shi, and W. Yu. FlexFetch: A History-Aware Scheme for I/O Energy Saving in Mobile Computing. In *IEEE International Conference on Parallel Processing (ICPP)*, 2007.

[8] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI'08*, Berkeley, CA, USA, 2008.

[9] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing Server Energy and Operational Costs in Hosting Centers. In *Sigmetrics 05*, Banff, Canada, 2005.

[10] X. Fan, W.D. Weber, and L.A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07*, San Diego, CA, USA, 2007.

[11] A. Gandhi, V. Gupta, M. Harchol-Balter, and M.A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. In *PERFORMANCE 2010*, Namur, Belgium, 2010.

[12] Intel Corp. Intel Math Kernel Library 10.0 - LINPACK. http://www.intel.com/cd/software/products/asmo-na/eng/266857.htm.

[13] O.B. Jennings, A. Mandelbaum, W. A. Massey, and W. Whitt. Server staffing to meet time-varying demand. *Management Science*, 42:1383–1394, 1996.

[14] J. Kim and T. S. Rosing. Power-aware resource management techniques for low-power embedded systems. In S. H. Son, I. Lee, and J. Y-T Leung, editors, *Handbook of Real-Time and Embedded Systems*. Taylor-Francis Group LLC, 2006.

[15] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. Katz. Napsac: Design and implementation of a power-proportional web cluster. In *First ACM SIGCOMM Workshop on Green Networking*, August 2010.

[16] J.C.B. Leite, D.M. Kusic, and D. Mossé. Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster. In *ICAC '10*, Washington, DC, USA, 2010.

[17] D. R. Mauro and K. J. Schmidt. *Essential SNMP*. O'Reilly.

[18] D. Mosberger and T. Jin. httperf—A Tool for Measuring Web Server Performance. *ACM Sigmetrics: Performance Evaluation Review*, 26(3):31–37, 1998.

[19] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: Managing performance interference effects for qos-aware clouds. In *EuroSys '10*, Paris, France, 2010.

[20] B. Urgaonkar and A. Chandra. Dynamic provisioning of multi-tier internet applications. In *ICAC '05*, 2005.

[21] U.S. Environmental Protection Agency. EPA Report on server and data center energy efficiency. 2007.

[22] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. In *HPCA '08*, Salt Lake City, UT, USA, 2008.