Abstract: We introduce and evaluate an automated technique for dynamically provisioning server capacity in a data center that caters to on-line services. Short-term load prediction is used to realize an effective energy-aware data center load balancing technique that achieves significant energy savings without compromising the delivered performance. In our heterogeneous datacenter consisting of 150 Linux servers, the load balancer is able to achieve a 30.8% reduction in overall server energy consumption on average across differently-sized configurations and across bursty workloads with minimal degradation in both response time and throughput.

1. Introduction

Data centers have become a ubiquitous part of the modern society. Data centers are behind web services, streaming media distribution, e-commerce, social networking and cloud systems that provide infrastructures/platforms as a services (IaaS, PaaS). Data centers, collectively, expend a significant amount of energy. In 2010, US data center energy expenditures were estimated at 67 to 85 billion kWh, amounting to 1.7 to 2.2% of the nation’s total electricity consumption. Worldwide, annual data center energy consumption exceeded 200 billion kWh in 2010 [7]. This problem is exacerbated by the fact that the number of data centers is growing at double digit percentage rates while available electrical generation facilities have remained fairly stagnant [6].

Data center operators generally overprovision the IT capacity to deal with unanticipated load increases and servers remain thoroughly underutilized on the average. In 2014, the industry average for data centers providing online services was reported as 12% [1]. Thus, most of the energy spent in data centers, as much as 90% in some cases [4], is wasted on grossly overprovisioned IT capacity. This energy wastage stems from the fact that idling servers dissipate a significant amount of power to remain active and this idle power can typically be 40% and upwards of the peak power dissipated by the server at maximum utilization. Idling and underutilized servers thus prevent a data center from being energy-proportional - that is, the performance delivered is not directly proportional to the energy consumed.

Ideally, if servers could be turned off or kept in a very low power state and had the ability to revert instantly to a fully active state capable of handling incoming requests, idling servers and performance losses due to dynamic changes in the workload can both be avoided, resulting in a data center that is more energy proportional. In reality, with one notable exception [10], server vendors have not offered products that come close to this ideal server. While, deep-sleep, low power dissipation states are well-implemented in laptops and portable devices at a smaller scale, server vendors have not offered similar products. The challenge is thus in provisioning IT capacity to match the dynamics of the workload and yet fully meet the workload SLAs, that is, to deliver the same quality of service as a data center that idles all its servers. In this paper, we describe a solution that addresses this challenge.

2. Related Work

Existing work on automated, energy-aware workload scheduling include [2, 3, 12] and others. Multiple sleep states and DVFS, along with a regression based workload prediction approach is used in [5] to optimize the number of spare servers in each state and realize energy savings in a cluster of servers providing multi-tiered services. Prediction errors are corrected with actual data in a feedback loop. SLB differs from these in its use of both short-term load prediction for server provisioning and the incorporation of thermal considerations. In [11], requests for static and dynamic pages were directed to servers with apriori knowledge of the energy needs for serving the request and as a function of the number of pending requests. PEGASUS [9] improves the energy proportionality of the data center through the explicit control of CPU power states using requests latency statistics in a feedback loop to just meet the upper limit specified for request latencies. PEGASUS makes no attempt to adjust the number of online servers based on the demand. Sleepscale [8] reduces data center energy consumption by selecting from a number of policies that control the power settings and sleep states of servers. The policies are
based on a characterization of the service quality and power settings of each server and take into account the latencies of reverting a server to the active state from a sleep state.

3. Dynamically Provisioning Server Capacity

We present SLB (Smart Load Balancer), a fully-automated solution for provisioning IT capacity to match the instantaneous offered workload that results in no performance loss against a baseline design where all servers remain active. A key aspect of our solution is in the use of a short-term load predictor that uses the immediate history of actual server loading to predict the imminent workload a few minutes in advance to permit turned off servers and servers in deep-sleep states to be activated in a timely fashion when the offered workload grows abruptly. Server activations in our scheme are relatively more aggressive than compared to the deactivations that are needed when the active server capacity exceeds the instantaneous demand. SLB thus contrasts well to other solutions that rely on known daily workload variations over time or systems that predict workload several hours to days in advance.

SLB assumes the typical structure of an online data center that accepts incoming requests through one or more tiers of load balancing switches and then, depending on the request, send it to one or more back-end servers that provide the needed service. In our technique, servers are broken down into groups and each server group is managed to provide just enough IT capacity to handle the workload assigned to the group. Grouping is used to provide scalability and to also permit the system to be thermally conscious - permitting the thermal load to be distributed evenly or to be localized to aggressively cooled spots, racks or servers within the data center. A server in within a group can be in one of the following states:

**Active (or online):** Servers in this state are serving requests and capable of serving additional, incoming requests. The load balancing switch maintains a list of the active servers and schedules incoming requests to these servers using a weighted least-connection approach.

**Deactivated (or offline):** Servers in this state do not receive any new requests, but are allowed to finish processing previously-assigned requests. Servers are migrated to this state when they are either unneeded or when they are about to cross a pre-designated temperature threshold.

**Suspended:** Servers in this state are in a deep sleep mode and must be awakened to the active state before they can serve requests. For servers that support the ACPI S3 state, all state is suspended in DRAM to permit fast revival (within 7.5 seconds in our implementation). For the remaining servers, the suspended state corresponds to the ACPI S5 state, in which a server must be powered on before use (which takes between 3 to 5 minutes in our setup).

**Waking:** Servers in this state are making the transition from a suspended state to an active state. Requests cannot be sent to a server in this state.

We now describe the control strategy in our solution at the level of each group.

4. SLB: Control and Scheduling

SLB’s goal is to always provision just enough server capacity to handle the instantaneous workload demands. Two issues have to be addressed in realizing this goal. First, SLB has to deal with the heterogeneous set of servers that are found in a typical data center, that is with servers that have diverse performance, efficiency, and controllability of power states. Secondly, it has to meet SLAs in the presence of sudden load surges. To address these issues, SLB does the following:

- SLB leverages the DVFS and turbo boost mechanisms in the servers’ processor cores to handle very short spikes in the offered load.
- SLB uses a short-term load-predictor to bring additional servers online in advance of larger growths in the offered load over a long duration.

Note that, while data center load prediction has certainly been a significant area of research, most of these approaches focus on predicting the offered load several hours to days in advance and are thus incapable of dealing with load transients that occur over a span of tens of seconds to a few minutes. As SLB completely powers off servers with poor or non-existent sleep states to realize additional power savings, and noting that it takes three to five minutes to typically bring such servers back online, this necessitates a load predictor that is capable of predicting the workload beyond a much shorter interval of time than what current research/techniques provide.

SLB’s load prediction technique is based on the observation that load balancing switches tend to direct requests preferentially to servers that have a lower utilization using some appropriate metric that is indicative of the load. Thus, one can make the assumption that the load trends of the most recently activated servers are indicative of the instantaneous load trend. SLB manages each group of servers independently and tracks this load trend by monitoring the utilization of the set of N most recently activated servers within each group, called GNRAS. The load trends of the GNRAS, in reality, are generally more
exaggerated than the load trends of the entire data center and thus its use in deciding whether or not to activate additional servers are often conservative, but this avoids any performance loss that would otherwise stem from under-provisioning the number of servers.

Within a group, the collective utilization of the GNRAS is computed as the harmonic mean of the utilization of the members of GNRAS. This is done to reduce the contribution from outliers; especially that of the very most recently activated server. As the membership of the set of the N most recently activated servers change over time, the collective utilization data is also updated based on server activations and deactivations at each sampling point.

To detect the trend in the load levels of recently activated servers, SLB periodically samples the load of the GNRAS at regular intervals, called scheduling intervals and calculates the collective utilization of the GNRAS at each of the last K sampling points, where K is a small number. A line is then fitted to indicate the utilization trend shown by the K sampling points. The slope of this line is used as the predicted rate of change in the utilization for the GNRAS. This rate can then be scaled up to the entire set of active servers.

To detect how quickly load is collectively growing on the GNRAS, SLB uses two operator-defined load thresholds, L1 and L2, where both are expressed as system utilization and where L1 < L2. The implications of these thresholds are as follows:

1. Till the collective utilization of GNRAS has crossed L1, the servers in GNRAS are considered to provide enough headroom to deal with load growths, including sudden load growths.

2. When the collective utilization of servers in GNRAS is between L1 and L2, and the utilization trend of the GNRAS indicates that collective utilization of GNRAS may cross L2 very soon (clarified later), servers are activated incrementally. Incremental activation is used in this utilization range as there is still adequate headroom left to deal with load surges.

3. When the collective utilization of GNRAS has crossed L2, a small number of servers (called a bundle) are activated simultaneously. A bundled allocation is used in this case to deal with load surges, as not enough headroom is left at the point. The number of servers within a bundle also change depending on the overall system utilization to limit the maximum utilization of each type of server within a pre-specified limit. This limit is chosen to guarantee the performance delivered by each type of server when it is heavily loaded.

In general, with each server activation - one server or a bundle, the impact of the allocation is assessed to revise the trend projection and to decide if further activations are needed. A reduction of the collective utilization of GNRAS, sustained over a number of consecutive intervals prompts server deallocation, which is always done less aggressively than the activations.

5. SLB Implementation

SLB uses a simple load reporting module that runs within each server and reports various aspects of the server load (CPU, memory, IO etc.) as well as thermal parameters. A front-end controller (called the Load Manager, LM) collects this load information and implements all of the SLB functions. The LM updates the list of back-end servers targeted by the load balancing switches when servers make state transitions under its directives. The LM also directs servers to make appropriate state transitions and also implements the thermal-awareness functions (which are not described here); it also implements server wear leveling to reduce thermal stress on servers that would otherwise be caused by uncontrolled state transitions.

6. Experimental Assessment

We evaluated our prototype implement on a cold aisle consisting of 16 racks populated with a heterogeneous mix of 150 servers (enumerated later). All of the servers PXE/NFS boot off the network into a Debian “Jessie” environment running Linux 3.16. Two Dell PowerEdge R610s were used exclusively for generating traffic, measuring latency, and server and SNMP data collection.

The aisle’s networking equipment consists of a F5 Networks BIG-IP 4000s LTM load-balancer, four Cisco Nexus 2448 switches, and two Cisco Nexus 3064 switches. Each Nexus switch is connected to a Cisco Nexus 5548 core switch, except for one of the Nexus
3064 switches, which connects to the neighboring Nexus 3064 switch instead. The load-balancer has two 10Gbit interfaces: one configured as internally facing and connected to the first Nexus 3064 and one configured as externally facing and connected to the second Nexus 3064. Figure 1 illustrates a scaled-down version of this network topology. We observed that the cross-sectional bandwidth was not exceeded during any of our experiments.

To permit power metering, each rack is fitted with a pair of Server Tech 24V2C415A1 rack-mount PDUs. To report total server power consumption, we poll each PDU in parallel at the maximum possible rate using the SNMP protocol and sum up the power consumption of each server. We do not measure the power consumption of the switches and the CRAC; SLB reduces energy consumption of these as well.

6.1 Workloads
As there are no standard workloads or input sets for a system of this nature, we developed two unique classes of workloads that are representative of online requests to evaluate SLB. The first workload consists of batch processing environment, in which we launch one of 56 tasks. Here, a task number and thread count are submitted using a HTTP GET request, which spawns one of the following tasks: all or one of four synthetic PHP tasks, a SPEC CPU2006 benchmark with test inputs, a PARSEC/SPLASH-2 benchmark with simlarge inputs, or a "cachegrind" task. Note that the thread count argument is ignored for all tasks other than the PARSEC/SPLASH-2 benchmarks. The four synthetic PHP tasks concatenate a string together several times in a tight loop, perform arithmetic in a tight loop, generate images, or calculate MD5 hashes of random binary data, respectively. For the SPEC and PARSEC/SPLASH-2 benchmarks, we used test inputs and simlarge inputs, respectively, to simulate short, bursty jobs similar to batch queries in an online environment. Lastly, the "cachegrind" task performs pointer-chasing in a manner designed to thrash the caches and create a lot of memory traffic (while minimizing the number of TLB misses).

The second workload emulates an online, content management service environment, in which we simulate hundreds of users concurrently requesting articles on a local mirror of the English version of the Wikipedia website. This workload tests SLB’s ability to remain energy-proportional in a bursty, real-world environment consisting of a large, replicated MySQL database, scripting languages such as PHP and Lua, and many other common web-based tools and libraries. For both workloads, we generated and measured traffic statistics using a harness tool that issues and measures the latency of each HTTP GET request. The latency measured is the latency as perceived by the client — that is, from the point the point when the socket is connected to the point when the socket is closed. The harness tool also consists of a trace generation mechanism that generates random inputs for both workloads and records both the time offset since the start of the evaluation and the HTTP GET parameters for each request. To ensure that our inputs are highly dynamic, the trace generator emits controllably-random waveforms spaced at controllably-random intervals. We then playback the same HTTP GET queries at the exact same time, in the exact same order as they were recorded. For high-traffic traces, we split the request playback and accounting across two dual-socket systems to minimize as much variation from the client end as possible.

6.2 Impact of Workload Prediction
To evaluate SLB’s prediction algorithm, we generated a very bursty trace with large, sudden load spikes followed by stagnant periods of various lengths as shown in Figure 2. Although a load pattern this extreme may be considered as unrealistic, our intent is to demonstrate how SLB’s prediction algorithm enables it to adjust to a wide array of environments without a priori knowledge of those environments. For the same reason, we disabled grouping for this particular study and defer the evaluation of grouping until later in this section. For the sake of comparison, we consider the performance and energy consumption of this trace under three configurations: a typical datacenter management scheme ("Always On"), SLB, and a variant of SLB without prediction (SLB-NP).

To create a reactive version of SLB, we stripped it of its prediction algorithm and left the temperature awareness and wear-leveling components intact. In lieu of the prediction algorithm, we designed the reactive scheduler such that it activates or wakes up one server per scheduling cycle as long as the average server load is above L2 and deactivates one server per scheduling cycle as long as the average server load is below L1. As in SLB, servers were allowed to suspend the moment they did not violate the constraints in place for wear-leveling, such as minimum time between power cycles. Lastly, we constrained the size of the waking list for SLB-NP to three servers. Without this addition, the reactive scheduler would likely power on all servers in the suspended list when the load levels remain high and there are no S3-capable servers available for wakeup due to the length of time it takes for the non-S3-capable servers to become ready.
As seen in Figure 2, the reactive solution, expectedly, activates and deactivates servers very aggressively. As such, the average server utilization quickly shifts to the ideal operating range during stagnant periods, and excess servers are either quickly deactivated or suspended altogether.

However, at the same time, the reactive solution is entirely incapable of dealing with large, sudden load spikes. This is partially due to the fact that it only reacts to load increasing above a threshold value, and partially due to the fact that it deactivates, and subsequently suspends, servers too aggressively. This can be seen in the active server count during sudden spikes, particularly where the active server count resembles a step function and the average CPU utilization showing a large fraction of servers utilizing turbo boost. It is at that point where the effect of the reactive solution exhausting the S3-capable pool of servers early on in the trace and subsequently resorting to slowly powering on bundles of non-S3-capable servers becomes apparent. In fact, during that time, performance suffers so much so that numerous requests are dropped altogether as the load balancer and server queues become saturated. All in all, the reactive solution dropped well over 1% of the total connections for the trace – something that is entirely unacceptable from an SLA point of view.

On the other hand, SLB’s predictor prevents it from powering off servers too aggressively and, as a result, SLB is able to provision the correct number of servers for each load spike. This is made apparent not only by how closely SLB tracks the utilization levels of the typical data center management scheme, but also by the average latency realized by SLB. SLB did not drop a single connection and experienced an average request latency that was within 3.75% of Always On.

In fact, during that time, performance suffers so much so that numerous requests are dropped altogether as the load balancer and server queues become saturated. All in all, the reactive solution dropped well over 1% of the total connections for the trace – something that is entirely unacceptable from an SLA point of view.

6.3 Scalability
One of SLB’s many contributions is its ability to allow an operator to partition servers into logical groups. As such, it is essential that SLB remain scalable both vertically, allowing for large group sizes, as well as horizontally, allowing for several groups. Thus, we evaluate the scalability of SLB under three different configurations and across two inherently unique traces. For the sake of comparison, we fix the composition of each group with respect to heterogeneity and only change the group size as shown in Figure 3. As the group size shrinks, so too must the SLB scheduling parameters for each group. Instead of re-determining ideal parameters for SLB-2G/4G we simply scale down GNRAS and scale up Q linearly with the group size.

In addition to scaling-down the group sizes, we must also scale-down each trace while retaining the character of each trace. As the group size shrinks, so too must the SLB scheduling parameters for each group. Instead of re-determining ideal parameters for SLB-2G/4G we simply scale down GNRAS and scale up Q linearly with the group size.

For the first trace we analyze is a mix of both single and multi-threaded synthetic and real-world workloads. An important observation that stands out is that the energy saved by SLB (over Always On) is inversely related to the group size – 27.6%, 24.6%, and 15.2% for SLB-1G, SLB-2G, and SLB-4G, respectively. This makes sense, as SLB’s scheduling
algorithm linearly interpolates the load growth rate. However, recall that we decreased GNRAS linearly with each decrease in group size: this has the unfortunate effect of creating more noise within the average utilization reported on the recently activated subset of servers. As a result, when the predicted load approaches L2, especially with sharp increases in load, smaller group sizes are more likely to activate servers out of caution. Although this results in higher energy consumption, it helps SLB retain service quality during spikes, even for small groups. Similarly, we can also observe that as GNRAS decreases, so too does the rate at which SLB deactivates servers, even though we scaled up Q with respect to the group size. That is to say, a decrease in GNRAS increases the likelihood that all GNRAS are below L1 at any given time. Thus, SLB is less apt to either activate or deactivate servers as GNRAS increases. This is a desirable feature due to the fact that, as a group increases in size, so too does its ability to absorb a fixed change in load.

Figure 4 - Multi-Threaded Trace

Next, we analyze SLB’s behavior with a workload comprised of an entirely different nature of requests: those from the MediaWiki framework. For this trace, we withheld 16 of the Dell PowerEdge R520 servers from each of the configurations and used them as MySQL database and memcached servers. The nature of these stateful services precludes us from being able to activate, deactivate, and suspend these servers as would typically be done with SLB and as such we do not include their power consumption, utilizations, or other statistics in this section of the results. The most interesting observation from this set of results is uniformity in the power consumption across all three grouping schemes, as seen in the power consumption of Figure 5. Notice that in this trace, since the load remained well below L2 and the increase in load was gradual, we did not observe varying degrees of energy savings across the three SLB grouping schemes as seen in the prior two traces. In addition to a more uniform power consumption across different grouping schemes, one can also observe SLB’s ability to save additional power as the average CPU utilization decreases.

Notice that, as in the previous trace, the average CPU utilization for the Always On scenario was about 30%, as shown in Figure 4. For the MediaWiki workloads, we generated a trace that was more indicative of the load profiles mentioned in the Gartner Group report (quoted in [1]). As shown in Figure 5, the average CPU utilization was closer to 20% for this trace. As a result, SLB powered off servers more aggressively than in the prior two workloads and, consequently, realized at least 10% additional energy savings for each grouping scheme when compared to the prior trace. In total, SLB saved 41.1%, 37.6%, and 38.6% for SLB-1G, SLB-2G, and SLB-4G, respectively, over Always On for this trace.

Figure 5 - Wikipedia Trace

7. Conclusions
Short-term load prediction as used in SLB is effective in mitigating performance losses in techniques that automatically provision server capacity to match the dynamics of the offered workload. Specifically, this prediction permits turned-off servers and servers in deep sleep state to be activated in a timely fashion before workload growth inundates the system.

8. Acknowledgements
This work was supported in part by the NSF through award Nos. 0958501 and 1040666 and separate NSF awards to the NSF IUCRC, Center for Energy-Smart Electronic Systems.
References


[12] WU, Q. Making Facebook’s software infrastructure more energy efficient with autoscale, August 2014. [Online; posted 8-August-2014].