Ram Raghunathan
Andrew ID: rraghuna

2011 Senior Research Thesis Extended Abstract

Design and Implementation of a Power-Aware Load Balancer

Abstract

Energy costs for data centers are doubling every 5 years, and are already more than $19 billion. Unfortunately, much of this energy is wasted. Servers are busy for about 5% to 30% of the time, but are left on. An idle server consumes as much as 60% of its peak energy demand.

The primary goal for a data center is to meet its response time Service Level Agreements (SLA’s). These typically take the form of a percentile guarantee. For example, a 95th percentile guarantee would mean that the data center serves at least 95% of the requests in at most the target response time. More recently, the secondary goal of reducing power consumption has been added. I introduce a load-oblivious and distributed load balancer that optimizes for power consumption, while meeting all SLA's.

Problem

Energy costs for data centers are doubling every 5 years, and are already more than $19 billion. [1] Unfortunately, much of this energy is wasted. Servers are busy for about 5% to 30% of the time, but are left on. An idle server consumes as much as 60% of its peak energy demand. [1, 2, 3]

The primary goal for a data center is to meet its response time Service Level Agreements (SLA’s). These typically take the form of a percentile guarantee. For example, a 95th percentile guarantee would mean that the data center serves at least 95% of the requests in at most the target response time. More recently, the secondary goal of reducing power consumption has been added.

Prior Work

There has been much research recently about reducing energy usage. There are a wide variety of techniques utilized. Some approaches are: optimizing the server hardware architecture [4, 5], using dynamic voltage and frequency scaling for processors [6, 7], virtualization [8, 9]. As our research focuses on reducing power demand by dynamic provisioning of data center capacity, I focus on prior work that uses this approach. Most of these solutions use a control of all servers being on, which is what industry employs today. This shall be here forth be referred to as the AlwaysOn technique.

Solutions in this area typically fall into one of two different categories. The first category is *reactive* solutions where the technique reactively adapts to the load coming into the data center. These techniques observe the response time and/or request rate to gauge whether servers need to be turned on or can be turned off to meet SLA. However, these techniques are usually computationally expensive. In addition, they are difficult to deploy when server set-up times are high, and may cause SLA violations.

The second category of techniques are *predictive* in which the technique attempts to predict the future load given past load information. One example of this technique is Moving Window Average (MWA), which extrapolates a average in a window to determine future load. Another example is Linear Regression, which fits a straight-line to the window of past request rates to determine future request rate. Given the future load information, the data center can then be provisioned for the increased or decreased load. However, unpredictable demand surges tend to result in severe SLA violations. Further-more, when set-up times are high, the future load that is calculated tends to be so far in the future, that it is meaningless.

Our Solution

Our solution, called AutoScale, is a distributed system that dynamically provisions data center capacity. Each system can independently turn itself on and off, without requiring a central control. The technique is also load-oblivious. AutoScale does not require previous information about the incoming load, nor does it attempt to predict future load.

AutoScale is combined with a load balancing algorithm that seeks to maximize power savings, rather than equitable load distribution. Each server has a threshold value of number of jobs that can be serviced at once, and the load balancer seeks to utilize a server to the maximum before distributing work to other servers. Clearly, this minimizes the number of servers being utilized. More servers can be turned off, as they are not busy, and energy consumption is reduced. If all possible servers are loaded to their thresholds, the load balancer falls back onto a Join Shortest Queue (JSQ) algorithm, which is optimal.

I now talk about turning servers off and on. When a server has been idle (not servicing a job) for a given target number of seconds, it is turned off. This delay is useful in allowing the load balancer to keep idle servers available in order to service any unprecedented increase in load. Once all usable servers are filled, AutoScale searches for an off server to turn back on. However, servers take considerable set up time before they are ready to service requests. As such, AutoScale turns on enough servers to service the number of jobs currently in the system.

We test this solution in implementation on a model data center, as explained below. This is substantially different from many prior research, which focus on simulations that tend to be more predictable than real-world experiments.

Experimental Setup

The experimental setup consists of many tiers, modelling a real data center setup. At the outermost tier, we have the client, who generates requests. We use a modified version of httperf [10] to generate requests. The second layer contains the load balancer which implements the AutoScale algorithm. This uses Apache with a modified version of the mod_proxy_balancer Apache extension. This extension allows incoming requests to be routed to one of a set of servers as per a selected routing algorithm. The third layer contains application servers, which service the job. Our AutoScale only operates on these application servers. These servers are also running an unmodified Apache webserver. The request takes form of a php script. The fourth layer is the cache layer. As data is requested by the application servers, it goes first to the cache. We use memcache on this layer. Memcache is a distributed (as opposed to replicated) cache. Finally, the innermost fifth layer contains the database which is accessed if the cache misses. This uses memcachedb for simplicity and easy integration with memcache.

We have one client and load machine. However, we have up to ten application servers, up to three cache servers, and two database servers.

These machines are located in the Intel Research Pittsburgh center in the Collaborative Innovation Center building.

Software Modifications

As AutoScale is tested in implementation, much software had to be adapted or created. In the Fall, I helped create a load creator and balancer that worked on a much simpler system. This system had only application servers, and ran a long-running (13 seconds) LINPACK benchmark as the workload. The load was taken from the 1998 World Cup Internet activity trace [11] as well as a generated periodic load. This proved successful and also matched up with simulation, so we decided to test on a larger, more realistic system, as described above.

On the client side, httperf was used to generate load parametrically. However, httperf can only accept one interarrival time parameter. To accurately simulate real-world scenarios, we needed to generate load with variable arrival rate. I modified httperf to accept many interarrival rates and number of connections on that arrival rate. Thus, a variable arrival rate load generator was created.

On the server side, the load balancer server ran Apache as it has a mod_proxy_balancer extension that performs load balancing. I added more load balancing policies, such as the threshold balancing described before to this extension. I also added a configurable thread which would perform the AutoScale power aware balacing. This thread would poll the application servers to turn them on and off as and when necessary.

Testing

Right now, we are working with the three tier system described before. A job involves querying the memcache for a value given the key. The value is a set of keys which are then queried for again. The application server does this for a given depth before returning the request. Thus, by changing the depth, we can increase or decrease the size of a job. This is quite similar to real systems. The work is ongoing, but preliminary results are promising, with better power consumption and performance over prior work.

References

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

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

[3] 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.

[4] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: eliminating server idle power. In ASPLOS ’09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems, pages 205–216, New York, NY, USA, 2009. ACM.

[5] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. Fawn: a fast array of wimpy nodes. In SOSP ’09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles,

pages 1–14, New York, NY, USA, 2009. ACM.

[6] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. Optimal power allocation in server farms. In SIGMETRICS ’09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems, pages 157–168, New York, NY, USA, 2009. ACM.

[7] Qiang Wu, Philo Juang, Margaret Martonosi, and Douglas W. Clark. Voltage and Frequency Control With Adaptive Reaction Time in Multiple-Clock-Domain Processors. In HPCA ’05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture, pages 178–189, Washington, DC, 2005.

[8] Xiaoying Wang, Zhihui Du, Yinong Chen, and Sanli Li. Virtualization-based autonomic resource management for multi-tier web applications in shared data center. Journal of Systems and Software, 81(9):1591 – 1608, 2008.

[9] P. Ranganathan R. Nathuji S. Kumar, V. Talwar and K. Schwan. M-channels and m-brokers: Coordinated management in virtualized systems. In Workshop on Managed Multi-Core Systems, 2008.

[10] David Mosberger and Tai Jin. httperf—A Tool for Measuring Web Server Performance. ACM Sigmetrics: Performance Evaluation Review, 26:31–37, 1998.

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