Lecture 19: Datacenters

Anirudh Sivaraman

2018/12/17

Today, we'll discuss an aspect of networking that has risen in prominence over the last decade or so: *datacenters*. Datacenters are large private networks consisting of tens of thousands of servers [15, 16]. Some of these datacenters are in buildings that are the size of six football fields [3]. We'll first discuss how such datacenters came to be [8]. We'll then discuss in some detail one specific aspect of datacenters: the topology of the network interconnecting the servers within a datacenter [11].

1 The origins of datacenters

1.1 Large-scale web search

The origins of datacenters can be traced back to the rapid increase in the size of the world wide web (WWW)[1] in the early 2000s. Search engines of the time, including Google, which was incorporated in 1998, had to cope with searching an ever-growing world wide web. Typically, these search engines maintained a database of different web sites that they had obtained by *crawling* the web periodically and chasing hyperlinks from one website to another. Soon enough, this database was too large to store on a single server machine and had to be split across many different machines. Barroso's paper [8] provides an early look (ca. 2003) at Google's system for handling search queries. As early as 2003, Google had around 15K servers in a single cluster [8], an earlier term for what we call datacenters today.

1.2 Many cheap servers vs. few expensive servers

Other companies that needed to process large amounts of data converged on a similar private network with 10s of thousands of machines, across a broad variety of use cases, e.g., social networking (Facebook) and online shopping (Amazon). The defining characteristic of these networks is the use of numerous cheap commodity servers rather than a few high-end servers [8].

This reliance on cheap servers was influenced by the rapid improvements in server capability in the early 2000s enabled by Moore's law. This meant that any server machine would probably become obsolete in 3 years or so, and hence the machine's cost had to be amortized over this small 3-year period. This in turn prompted an architecture where individual machines were as cheap as possible. The observation made in Barroso's paper wasn't exactly new. Several years ago, the NOW project at Berkeley [6] had made similar observations about building a cluster out of workstation-scale machines. But, it took the scale of Google's search engine for this idea to really take off in a big way.

1.3 Cloud computing: computing as a utility

Clusters or data centers are related to cloud computing, a term many of you may have heard of it. Cloud computing is the idea of computing as a utility [13, 7] where compute, storage, memory, and network capacity can be rented out by the hour, and more recently by the second [4].

[1]Size as measured by the number of web sites on the Web.

Cloud computing in its most mainstream form arose from the idea at Amazon that large-scale private networks/clusters/datacenters that were used internally at Amazon could also be profitably rented out to third-party users because of the large compute capacity that Amazon had built up over the years. The first publicly available offering of cloud computing was Amazon's Elastic Compute Cloud (EC2) in 2006. Microsoft and Google followed with Azure and App Engine in 2008.

Today, there are two broad kinds of clouds [7]: *public clouds* such as EC2, Azure, and AppEngine that a third-party user (like me or you) can log into and work on as though it was our own machine, and *private or enterprise clouds* that are for the internal use of companies such as banks, Walmart, FedEx etc.

Hopefully, by now, you understand how datacenters came to be and what their relationship to cloud computing is. We'll now go over one specific concept relevant to the network within datacenters: the network's topology. We will draw heavily from one paper: VL2 [11], an early look at the design of Microsoft's datacenter network from 2009. The ideas in VL2 were also developed earlier in the context of an academic project at UCSD in 2008 [5], although VL2 was likely the first paper to document these ideas in the context of a real datacenter running production workloads.

Other datacenters, such as the ones at Google [15] and Facebook [14], follow a similar network topology as Microsoft's datacenter network. However, it is unclear whether the Microsoft design directly inspired these datacenters or whether these datacenters were all concurrently designed. This is because the Google and Facebook papers were published several years after their datacenter networks were developed and in production.

2 Datacenter topology

The goal when designing a datacenter network is to allow any two servers within the datacenter to communicate with each other as though they had a dedicated high-capacity link between them. Let's say we had 10K servers. This would require us to create 10K * 10K = ˜100M links to connect every server to every other server—much like one extremely large switch[2] with 10K input ports and 10K output ports and a server connected to each one of these input/output ports.[3] This is what datacenters strive for: the one-big-switch abstraction [12] connecting N servers, but with far fewer than N^2 links or a switch with N ports, because for large N (such as 10K servers) N^2 links or an N-port switch is hopeless.

How do we provide the one-big-switch abstraction for $N = 10K$ servers without building an N-port switch? The idea is to use switches with smaller port counts and arrange them intelligently in a topology of switches to provide this illusion. The idea goes back to Charles Clos in 1952 [9], who dealt with a similar problem when building extremely large telephone exchanges. Clos' problem was to build a telephone exchange supporting many users that allowed any user to talk to another without being blocked by the conversation of another user pair, by using smaller telephone exchanges[4] to emulate a larger one.

Let's start with a simple example. Let's assume a k-port switch that can connect to k servers. Typically, k is on the order of a few tens to a few hundreds of ports today.[5] One way to build a one-big-switch abstraction using a k-port switch is given below and shown in Figure 1. Figure 5 of the VL2 paper [11] contains a similar, but slightly more involved, example.

1. We create a *leaf layer* of k k-port switches. For each switch in the leaf layer, we connect $\frac{k}{2}$ ports (i.e., half the ports) to $\frac{k}{2}$ servers.

2. We create a *spine layer* of $\frac{k}{2}$ k-port switches. We connect each of the remaining $\frac{k}{2}$ ports on each leaf-layer switch to all $\frac{k}{2}$ switches in the spine layer. To check that this works, let's first compute the total number of ports at the leaf layer that are not connected to server machines: k leaf layer switches each with $\frac{k}{2}$ unconnected ports, which totals up to $\frac{k^2}{2}$ ports. At the spine layer, there are $\frac{k}{2}$ switches each with k

[2]We are using the term switch interchangeably with the term router.

[3]Recall that the input and output ports are really two wires on the same physical port.

[4]These telephone exchanges were called switches. We realize this is confusing, but the term switch originated in the context of mechanical and electronic switches in the telephone network.

[5]There is a range in the number of ports supported by a switch because the same switch can be used to support (say) 250 10-Gbit/s ports or 25 100-Gbit/s ports because the internal switching mechanism is the same in both cases.

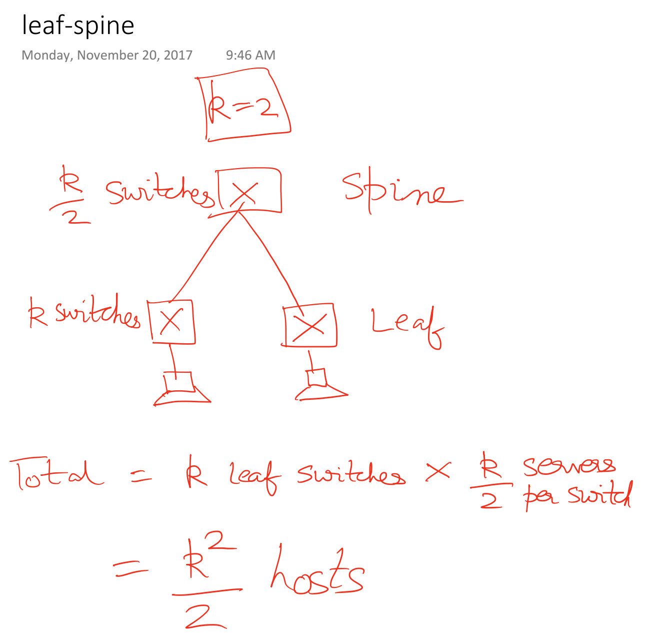
leaf-spine
Monday, November 20, 2017 9:46 AM
k=2
k/2 switches
Spine
k switches
Leaf
Total = k leaf switches × k/2 servers per switch
= k²/2 hosts

Figure 1: 2 layer leaf-spine topology with k=2 port switches

unconnected ports, which again totals up to $\frac{k^2}{2}$ ports. Hence, there is a port in the spine layer for every unconnected leaf layer port.

Does this idea actually provide the one-big-switch abstraction? It does so under two conditions [11]. We won't explain why, but both conditions are fairly natural: (1) no server sends or receives more than the capacity of the link connecting it to the switch (in our telephone exchange analogy each user is communicating to at most one other user), and (2) traffic is spread out uniformly over all the leaf-to-spine links.

VL2 [11] achieves the first condition using TCP congestion control, which automatically adjusts the window size to the link capacity (more precisely the bandwidth-delay product), and the second through an idea called Valiant Load Balancing (VLB), which is essentially just a routing algorithm that randomly spreads traffic equally across a set of candidate links when routing traffic from one server under a particular switch to another server under a different switch.

What does VLB do? For every new packet that arrives at a particular leaf switch S destined for a particular leaf switch D, there are k different paths to take for this packet: each path "bounces off" one of the k spine layer switches. The goal of VLB is to spread out traffic so that for each combination of S and D, every one of the k spine layer switches is chosen with the same probability $\frac{1}{k}$.

Ideally, VLB would be implemented by picking a spine layer switch at random for every packet that arrives at a leaf layer switch. The problem with this is that two packets belonging to the same TCP flow may take different paths, i.e., different spine layer switches, because of how the random choice worked out. If two packets from the same TCP flow take different paths, there is the risk that the packets arrive at the TCP receiver out of order.

TCP interprets this reordering as a sign of packet loss because acknowledgements in TCP are cumulative by default. Packet loss, in turn, leads to a decrease in the congestion window due to the AIMD algorithm. Furthermore, the networking code in most kernels is optimized for the common case that packets arrive in order without reordering, and reordered packets trip up TCP processing in the kernel leading to an increased computational overhead from packet processing [10]. For both reasons, it is preferable to retain packet order within a TCP flow.

Hence, instead of making a random choice on each packet, VL2 makes a single random choice for all packets within a flow, using the Equal Cost Multipathing (ECMP) algorithm present in many switches today. ECMP is essentially just a load balancing algorithm that assigns a new flow randomly to one of several candidate outgoing links—in this case leaf-to-spine links. Note that there is no choice in the spine-to-leaf link because this is dictated by the destination leaf switch. VL2 makes the observation that because many flows last a very short duration (1 second) [11] in Microsoft's datacenter, a random choice at the level of flows instead of packets still provides ample load-balancing opportunities to spread out the load evenly.

2.1 Many cheap switches vs. few expensive switches

VL2 applies the idea from Google's cluster architecture (and the Berkeley NOW project) of interconnecting a large number of servers using many small port-count switches, instead of one gigantic port-count switch. In fact, prior to embarking on their design, the VL2 authors considered the predominant network design for dataceners at the time (Figure 1 of the VL2 paper [11].). This design involved higher port-count and higher link-capacity switches at higher layers, which were more expensive to procure and did not provide the one-big-switch abstraction of uniform connectivity between the servers. In a way, VL2 (and more explicitly the earlier work from UCSD [5]), applied the idea of a large number of cheap components from computing to networking. This was probably not an accident; the faculty member on the UCSD paper, Amin Vahdat, was a graduate student at UC Berkeley during the Berkeley NOW project [6].

2.2 Comparing the leaf-spine approach to a naive switch

The leaf-spine approach allows us to connect $\frac{k*k}{2}$ servers to each other using a total of $\frac{k*k}{2}$ links between the leaves and the spines. We are not counting the links from the servers to the leaf switches because this is unavoidable even if we were using a naive switch. For a naive switch interconnecting N servers to each other, we would need at least $\frac{N.(N-1)}{2}$ links—even if we assumed bidirectional links and did not connect a server port back to itself. In other words, with the naive switch, the number of links would scale quadratically with the number of servers, while with the leaf-spine approach, it would scale linearly. More specifically, for $\frac{k*k}{2}$ servers, we need $\frac{k*k}{2}$ links. In other words, for N servers, we need N links in the leaf-spine approach, which is a linear relationship.

2.3 Adding more layers to the topology.

The leaf-spine approach can be scaled to support a larger number of servers. Figure 2 (based on Figure 3 of the UCSD paper [5]) contains an illustration of a three-layer network with an edge, aggregation, and core layer, built out of $k = 4$ port switches. This network can support the one-big-switch abstraction on $\frac{k^3}{4}$ servers. With $k = 64$, which is a common port count for switches today, this topology can support 65K servers.

This approach can be extended to more layers, but supporting more than 100K servers within the same datacenter is quite rare. So 3 or 4 layers should suffice for most practical purposes, especially with port counts exceeding 100 in many switches today [2, 1].

References

[1] Barefoot: The World's Fastest and Most Programmable Networks. https://barefootnetworks.com/media/white_papers/Barefoot-Worlds-Fastest-Most-Programmable-Networks.pdf.

[2] High Capacity StrataXGS®Trident II Ethernet Switch Series. http://www.broadcom.com/products/Switching/Data-Center/BCM56850-Series.

[3] Mark Zuckerberg Just Shared Rare Photos of Facebooks Data Center In The Arctic. https://www.fastcompany.com/3064187/mark-zuckerberg-is-sharing-stunning-photos-of-facebooks-arctic-dat.

[4] New Per-Second Billing for EC2 Instances and EBS Volumes. https://aws.amazon.com/blogs/aws/new-per-second-billing-for-ec2-instances-and-ebs-volumes/.

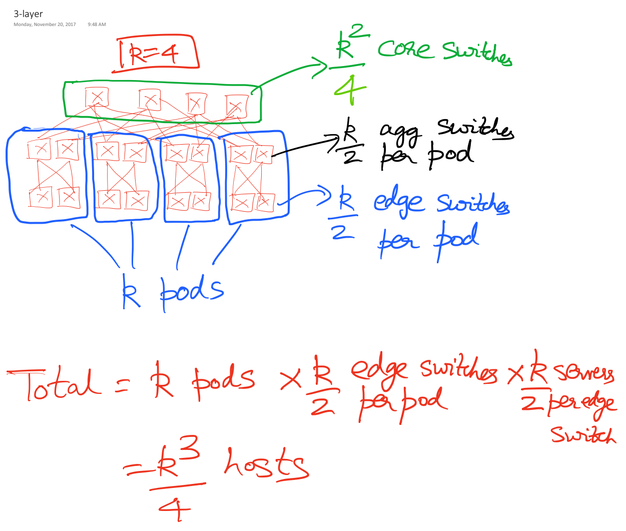
3-layer
Monday, November 20, 2017 9:46 AM
k=4
$\frac{k^2}{4}$ core switches
$\frac{k}{2}$ agg switches per pod
$\frac{k}{2}$ edge switches per pod
k pods
Total = k pods × $\frac{k}{2}$ edge switches per pod × $\frac{k}{2}$ servers per edge switch
= $\frac{k^3}{4}$ hosts

Figure 2: 3 layer topology with k=4 port switches. Figure based on Figure 3 of the UCSD paper [5].

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM*, 2008.

[6] T. E. Anderson, D. E. Culler, and D. Patterson. A case for NOW (Networks of Workstations). *IEEE Micro*, Feb 1995.

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 2010.

[8] Luiz Andre Barroso, Jeffrey Dean, and Urs Holzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 2003.

[9] Charles Clos. A Study of Non-Blocking Switching Networks. *Bell System Technical Journal*, 1953.

[10] Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, and Mohammad Alizadeh. Juggler: A Practical Reordering Resilient Network Stack for Datacenters. In *EuroSys*, 2016.

[11] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM*, 2009.

[12] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. Optimizing the "One Big Switch" Abstraction in Software-defined Networks. In *CoNEXT*, 2013.

[13] Peter M. Mell and Timothy Grance. SP 800-145. The NIST Definition of Cloud Computing. Technical report, Gaithersburg, MD, United States, 2011.

[14] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the Social Network's (Datacenter) Network. In *SIGCOMM*, 2015.

[15] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *SIGCOMM*, 2015.

[16] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *EuroSys*, 2015.