Mega Data Center for Elastic Internet Applications

Hangwei Qian
VMWare
qianhangwei@gmail.com

Michael Rabinovich
Case Western Reserve University
misha@case.edu

***Abstract*—This paper outlines a scalable architecture that supports datacenter-wide resource management for elastic Internet applications in a mega data center. Our architecture includes a scalable load-balancing fabric and provides effective knobs to balance load among the applications, servers, access links, as well as the load-balancing components themselves – the low-level resource managers and switches in the load-balancing fabric.**

I. Introduction

Modern cloud providers are building data centers with hundreds of thousands of servers, knowns as *mega data centers*. For example, Microsoft's data center in Chicago can potentially pack up to 300,000 servers [1]. Mega data centers are especially attractive to host elastic clouds, which dynamically adjust resources allocated to applications based on the demand and which are particularly valuable for Internet applications where the demand is often hard to predict in advance. In this environment, mega data centers not only reduce the overall operating costs because of the economy of scale, but also promise better resource utilization through the statistical multiplexing of resource usage among the hosted applications.

However, to fully utilize this promise requires global resource management across the entire data center. While efficient dynamic resource management is relatively well understood for traditional data centers, the scale of mega data centers brings new challenges. This position paper describes these challenges and sketches a novel data center architecture capable of meeting them. In particular, we concentrate on supporting elastic Internet applications where demand is driven by external client requests.

A. Challenges

With qualitative increase in scale of mega data centers, many traditional solutions in data center management and architectures no longer apply. We formulate some of these new challenges below.

Scalability of resource provisioning algorithms The resource management in the data center aims to balance load among servers and other elements, minimize application placement changes, and increase utilization of the servers (e.g., to conserve energy). As shown in [23], this is an NP hard problem. A number of heuristic algorithms have been proposed but they do not work at the mega data center scale. For example, in the scheme in [23], the algorithm execution time increases exponentially with the increase of the number of managed machines and needs about half minute to create provisioning decisions for only about 7,000 servers and 17,500 applications. Also, in [25], it takes about 30s to manage 1500 virtual machines.

One way to address this problem is to distribute these tasks among multiple agents, either at the host level or application cluster level, as proposed in [10], [26]. However, distributed approaches improve scalability at the expense of the quality of their solutions (except for some restricted settings, e.g., in the case of content delivery networks that cache static files [24]).

Limitations of load-balancing elements Traditional data centers commonly deploy load-balancing switches (called LB switches here) to achieve fine-grained load balancing and fail-over among replicated servers. According to [6], 10% of devices in the studied data centers were load-balancing switches. However, these switches do not scale to the demands of a mega data center. For example, the maximum number of external IP addresses for applications (known as virtual IP addresses, or VIPs) and real internal IP addresses for replicated servers (known as real IP addresses or RIPs; these can be taken from a private address space such as the 10.0.0.0/8 block) supported by Cisco Catalyst switches are 4,000 and 16,000 respectively [12]. A naive way to support a greater number of applications or their replicated instances is to partition the applications among multiple switches; this however, compartmentalizes the data center resources and diminishes the benefits of statistical multiplexing.

Variety of load-balanced resources A traditional data center typically has a small number of access links to the Internet, often a single default link and a backup, and a fairly streamlined internal network organization. Thus, resource provisioning traditionally has concentrated on server load. However, a mega data center typically has multiple Internet access links and border routers as well as complex internal network topology, necessitating load balancing among these components. In particular, in this paper, given recent advances in intra-data center network organization [2], [8], [17], we address the issues of load balancing among Internet access links as well as among LB switches that received less attention.

B. Approach

In this position paper, we introduce a novel data center architecture capable of addressing these challenges. First, to address the scalability limitations of research provisioning algorithms, servers in the data center are divided into groups, which we call *pods*. Existing solutions [23], [28] are applied to efficiently allocate server resources to applications within each pod. Meanwhile, to avoid compartmentalization of the data center resources, we introduce new control knobs to dynamically reallocate resources among the pods. This results

in a hierarchical approach that is scalable and yet we believe capable of approaching global optimality.

Second, we present control knobs to balance the load of the access links and LB switches, which in addition to limited numbers of VIPs and RIPs have a finite throughput capacity. Finally, we sketch an architecture that ties together all the control knobs in a coherent platform.

II. System Environment

We consider a mega data center that houses a multi-tenant cloud platform. We assume applications that implement client-facing Internet services and that each hosted application runs in its own virtual machine (VM); in other words, our applications roughly correspond to websites. A popular application can be represented by multiple VM instances, in which case the application instances are fronted by a load balancing (LB) switch that distributes client demand among the application instances. Each application is represented by an external virtual IP address (VIP) that maps to the LB switch and a set of real IP addresses (RIPs) representing all VM instances of the application. Incoming client communication sessions are addressed to the application's VIP, and these sessions therefore arrive at the switch on which the application's VIP is configured; the switch then chooses a RIP to service this session. For specificity, we assume the following load-balancing switch parameters: 4,000 VIPs, 16,000 RIPs and 4 Gbps throughout the discussion. These parameters are characteristic of Cisco Catalyst switches [12], but our approach equally applies to switches with other parameters. We target the mega data centers with around 300,000 servers and 300,000 applications, and we assume on average 20 VM instances per application. Thus, the entire data center requires at least 300,000 VIPs and 6M RIPs. Websites are typically structured in a multi-tier fashion, where client-facing application servers (which could be application servers such as Websphere or JBoss, or HTTP servers such as Apache) communicate with backend databases and other services. For simplicity, we focus on independent scaling of virtual machines in this position paper. Other research addresses co-placement of VMs that communicate with each other [29], [13], [15], [22], [3]; our architecture can also incorporate these ideas.

III. Architecture

Our proposed architecture is shown in Figure 1. In this architecture, the access network of the data center consists of the access connection layer and the load-balancing layer. The former provides Internet connectivity to the data center and consists of border routers that are connected through the access links to the access routers (ARs) belonging to the ISPs from which the data center obtains its connectivity. The latter consists of LB switches that are connected to the border routers on one side and to the servers on the other. The LB switches are connected to the servers though an existing L2/L3 switch fabric, so that each switch can in principle (and as we will see in Section III-B, in practice) include any server into its load-balancing groups. Greenberg et al. [7] pointed out that address translation most switches use to reach physical servers across subnet boundaries sometimes conflicts with application requirements. Patel et al. [19] address this issue by using packet encapsulation in place of address translation, an option offered by some products (e.g., MidoNet [16]) and readily implementable in any OpenFlow-compliant switch. The border routers and the LB switches are fully interconnected (most likely through a thin layer of L2 switches) to enhance the platform reliability. The rest of this section discusses key aspects of this architecture in more detail.

A. Hierarchical Resource Management

In order to scale resource management to the mega datacenter dimensions, we propose a two-level hierarchical resource management architecture where physical servers are divided logically into groups, which we call *pods*[1]. A server pod manager only knows the servers and applications of its pod, and dynamically provisions resources to applications within its pod. For example, when detecting that the load of the some application is increasing, it can increase the capacity of some VMs running an instance of the application (reducing the capacity of other applications if needed), or create more active instances of this application on lightly loaded servers in the same pod. Existing resource allocation algorithms, e.g., as proposed in [23], [28], can be applied here. The size of a pod (the number of hosts and deployed applications) should be small enough so that these algorithms can cope with the pod's scale. In our architecture, given the reported scalability of existing resource managers [23], [28], we target each pod to be limited to about 5,000 servers and 10,000 VMs (whichever comes first). We say a VIP *covers* a pod when the pod contains a VM whose RIP maps to this VIP. Similarly, an application covers a pod when the pod contains an instance of the application.

Besides pod managers, there is a datacenter-scale resource manager (referred to later as global manager below, although one must realize that resource management can also occur at yet higher level – across multiple data centers) that monitors resource utilization of all the pods and balances the load among them. A pod can be overloaded for several reasons. It may be overloaded due to approaching its processing capacity limits, e.g., its servers may be overloaded with no spare capacity within the pod for the server pod manager to resolve the issue. A more subtle issue is that the server pod manager itself may become "overloaded" due to too many servers and applications in the pod, which increases the decision space for the pod manager and slows down its resource allocation algorithms beyond acceptable levels.

The global manager performs three key functions in our architecture. First, as just discussed, it implements the top level in our hierarchical resource management scheme to avoid overload of server pods. Second, it monitors and manages other datacenter-scale resources, such as LB switches and access links. Third, it contains the VIP/RIP manager component as described in Section III-C.

[1]These server pods should not be confused with pods and racks in a traditional datacenter network topology. In fact, as discussed later, we assume more advanced recently proposed topologies.

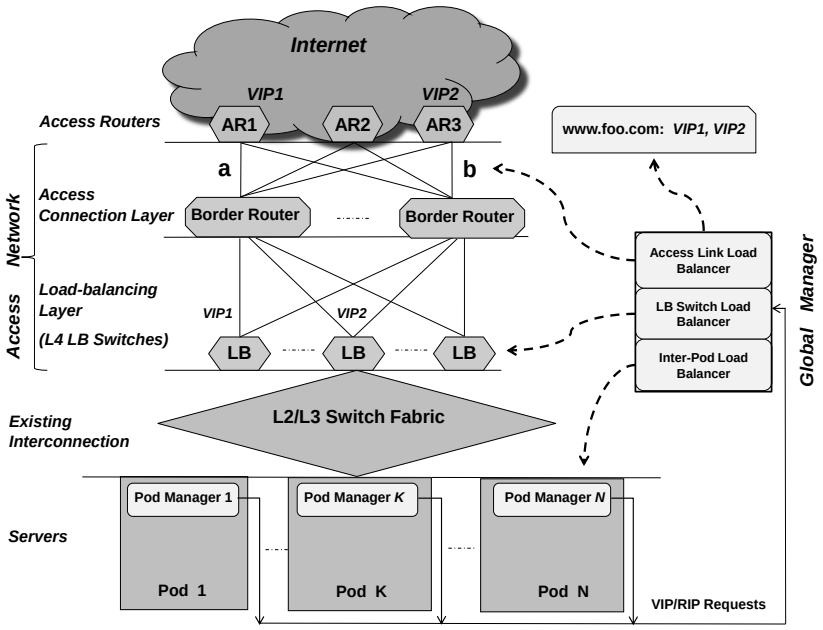

Fig. 1. Data Center Architecture. The dashed arrows represent both control and monitoring connections.

B. Positioning the LB Switch Fabric

In this section, we provide rationale for the placement of LB switching layers in our architecture. In traditional data centers, LB switches are placed close, or directly connected, to servers, which limits the number of servers they are able to reach, increases the number of LB switches required, and compartmentalizes data center resources. The traditional architectures accept this drawback because if an LB switch connects to a remote server, the traffic between the switch and the remote server will compete with other intra-datacenter traffic, and the amount of bandwidth available for traffic between the switch and the server will be unpredictable.

However, recent advances in data center topologies [2], [8], [17] guarantee bandwidth between any host-pair within the data center and provide flat address space to all the hosts. Thus, we place LB switches close to the border and connect them to servers through the L2/L3 switching fabric, which we assume implements one of these modern topologies. As a result, the LB switches can reach any host in the data center thus providing flexibility in forming load-balancing groups. In particular, this allows our architecture to support logical pods for the purpose of application instance management, which are independent of server location in datacenter network topology structures such as racks or physical pods.

One question is whether the LB switches would become communication bottleneck when placed at the access network. In this regard, we first note that the LB switches just need to process the traffic entering/leaving the data centers, which is only about 20% of total amount of traffic according to [8]: all intra-DC traffic flows below the load-balancing fabric. Furthermore, the LB switches are very powerful to offer high bandwidth capacity. For example, according to [12], Cisco LB switches are able to process 1.25 million packets per second, maintain 1 million concurrent TCP connections and provide 4 Gbps bandwidth when switching at layer 4. Suppose there are 300,000 applications. Even when each application is assigned only two VIPs, the number of required LB switches is at least $300,000 * 2/4,000 = 150$, which can provide about 600 Gbps aggregate external bandwidth. Thus this layer will not be a bottleneck.

C. VIP/RIP Management

Many resource management mechanisms discussed below involve reconfiguration of the LB switches to add or delete a VIP or RIP address. In fact, given that each switch can handle a limited number of VIPs and RIPs, such reconfiguration is also needed to balance the number of VIPs and RIPs configured on each switch.

To utilize LB switches (an expensive resource) to the fullest, our architecture makes all the LB switches available as globally shared resources for all applications. Consequently, various control elements such as individual server pod managers, as well as the global manager, can have independent and potentially competing needs for VIP/RIP configuration. In order to mediate and serialize all requests for VIP/RIP (re)configuration, we assign the responsibility to process any such requests to the global manager. Any component that needs to update the VIP/RIP configuration at any switch sends a request to the global manager. The global manager processes the requests sequentially according to their priority. For each request for a new VIP, the manager identifies an underloaded switch (i.e., one with few already-configured VIPs and a low data throughput being handled), allocates an unused IP address, and configures the switch with this IP address as a new VIP. For each request for an additional RIP, the manager considers the switches that host one of the VIPs of the corresponding application, selects the most appropriate switch with spare RIP capacity (according to some policy that

is orthogonal to the architecture we discuss here; e.g., the policy may take into account the current data throughput of the switches, the number of RIPs the switch handles already, and load on the access links carrying traffic to and from the corresponding VIP), and configures the extra RIP on this switch. VIP and RIP deletion is handled in a straightforward way.

IV. Control Knobs

In this section, we describe the mechanisms available to control load at various platform components – servers, server pods, LB switches, and access links. As we will see, our architecture makes a large number of control knobs available to both local server pod managers and the global resource manager to manage resources across the data center. The sheer number of these knobs increases the decision space and makes computing proper policies more challenging. In fact, most of the knobs affect each other, further complicating the issue. We are investigating algorithms involved in our ongoing work.

A. Selective VIP Exposure

Data centers are connected to the access routers (ARs) of ISPs through border routers with access links. Even traditional data centers are often connected to multiple ISPs for the reliability and economic concerns, and one can certainly expect this to be the case with mega data centers – not just for the above reasons but also to procure appropriate bandwidth. Here, traffic engineering is necessary to: (i) avoid overloading of any access link and (ii) control the traffic among the different access ISPs according to the business requirements (e.g., different link usage costs).

A naive way to implement such traffic engineering is VIP transfer between access links: we could withdraw the routes for some VIPs from the access routers associated with the overloaded access links and re-advertise them at other access routers with lightly-loaded access links. To avoid service disruption during the transition period, we could advertise padded autonomous system paths through the old routers before withdrawing these routes, and only withdraw them once no new connections come through the old routers. However, the load balancing based on such dynamic VIP advertising is slow and increases the number of route updates.

Instead, we propose a knob we refer to as *selective VIP exposing*, which works as follows. When advertising the VIPs to the access routers at ISPs, each VIP is only advertised selectively to certain access routers (typically only one). Then, the global manager can control which access router to use for each application by dynamically configuring the platform's authoritative DNS system to selectively reply to DNS queries from external clients with appropriate VIPs. For example, referring to Figure 1, suppose a website foo.com is assigned VIP1 and VIP2, which are advertised, respectively, at access router AR1 through access link a and at AR3 through link b. If link a is overloaded, the platform's DNS can resolve client queries for foo.com to VIP2 more frequently, causing more traffic to go through link b. Meanwhile, the platform can periodically withdraw blocks of unused VIPs from the old access routers and re-advertise them through lightly loaded access links.

With selective VIP exposing, overloaded links are relieved as soon as DNS starts exposing new VIPs, and routing updates are infrequent as they are decoupled from the load-balancing decisions and occur at most once per period for each unused VIP. The more VIPs are allocated to each application, the more flexibility the system would have for load balancing over the access links. However, too many VIPs per application increase the number of LB switches due to the limited number of VIPs a switch can handle, which translates into higher cost. In our current work, we assign three VIPs per application on average (popular applications are assigned more than unpopular applications) by default. The tradeoff between the flexibility for load balancing and the number of LB switches will be evaluated quantitatively in our ongoing work.

B. VIP Transfer Between LB Switches

Changes in demand for various applications can lead to a situation where an LB switch hosting VIPs of newly popular applications approaches its throughput limit (4Gbps). The global manager must rectify this situation by balancing the load among the LB switches. Selective VIP exposing mentioned above can also be applied here – the global manager can instruct DNS to expose only the VIPs of the applications configured at lightly-loaded LB switches and transfer the VIPs from overloaded LB switches when detecting that clients no longer are using them. In addition, our architecture allows an internal reassignment of VIPs without external router re-advertisements. Given that every LB switch connects to every border router, a VIP can simply be moved from the overloaded to an underloaded LB switch. Many LB switches (including the Cisco units we are considering here) allow programmatic reconfiguration to enact such transfers. The border router will be notified of this change so that future packets will arrive at the new LB switch, but no access routers are involved in the transfer. We refer to this mechanism as *dynamic VIP transfer*.

Note that, similar to selective VIP exposure, a VIP cannot be blindly transferred from one LB switch to another: while the VIP is in use by ongoing TCP sessions, packets of the same TCP session must arrive to the same RIP, and only the original switch knows this RIP. The global manager can increase the likelihood of a pause by using selective VIP exposing first to stop directing new clients to it. Although some clients will continue using this VIP in violation of time-to-live of old DNS responses [18], [4], the overall subsided usage will increase the likelihood of a pause to enact the transfer.

C. Server Transfer Between Pods

As mentioned above, pods are formed logically by the configuration of IP address of the servers and their hosted VMs, which enables a flexible knob for resource reallocation among the pods.

When the global manager detects an overloaded pod due to the pod's processing capacity, the global manager can allocate more resources to the pod by transferring servers from lightly loaded pods. To this end, it requests the server pod managers in

the donor pods to vacate some servers (remove any application instances running there). The vacated servers can then be handed to the server pod manager of the recipient pod.

While applying this strategy, the global resource manager must avoid "elephant" pods. Some applications in a given pod may become so popular that the global resource manager may add a large number of servers to the pod. As discussed earlier, this can hamper the operation of the server pod manager. When the pod manager becomes the bottleneck, the global manager can transfer servers – in this case along with its deployed application instances – out of the pod thus reducing the decision space for the pod manager to compute its local resource allocation policy.

D. Dynamic Application Deployment

Besides adding more resources to busy pods, another way to relieve a pod that is overloaded due to processing capacity constraints is to migrate or replicate applications to underloaded pods. This knob is enabled by recent advances in efficient virtual machine migration [25], [14] as well as the ability to configure load-balancing switches programmatically with new RIPs. Similarly, if an underutilized application covers many pods, the global resource manager can remove unnecessary instances of this application from the busier pods.

Again, the global resource manager must avoid elephant pods here, since an underutilized pod can receive many applications, which can overload and slow down the server pod manager beyond acceptable levels. Yet another consideration is that the number of application deployments and removals must be minimized as these operations are resource-intensive and can create turbulences in the platform operation.

E. VM Capacity Adjustment

A lighter-weight alternative to cloning or migrating a VM is to simply readjust VM capacity among the VMs co-located on the same physical server. Common VM monitors, e.g., VMWare ESX, allow VMs to be allocated hard slices of physical resources such as CPU cores and capacity share, memory, and bandwidth share. Further, these slices can be adjusted programmatically and, for many guest operating systems, "on the fly"—without needing a reboot [5]. By adjusting these slices, a server pod manager can reallocate server resources among the applications hosted within its pod.

F. RIP Weight Adjustment

Load-balancing switches also allow programmatic change to the weights they use in their load-balancing algorithms when they distribute the traffic coming to a VIP among the corresponding RIPs. If a VIP covers multiple pods, the global manager can adjust the load among the pods due to this VIP by updating the weights of the corresponding RIPs at the LB switch where the VIP is configured. For a VIP that has multiple RIPs within one pod, this pod's local manager can adjust these RIPs' weights to balance the load of servers within the pod. In particular, such intra-pod RIP weight adjustment is especially effective when employed in tandem with VM capacity adjustment. Note that to use this knob, the pod manager needs to be aware of which VIPs its RIPs are mapped to, and enact weight adjustment through requests to the global manager, which ensures that the total weight of the RIPs in the pod remains the same and therefore the load on other pods is not affected.

One advantage of this knob is that the resultant change can occur quickly, leading to highly agile resource management. Indeed, configuring the load balancing switches takes only several seconds [20], [28]. At the same time, this knob can redistribute the load among the pods only when they are covered by a common VIP, and within a pod only when it contains multiple RIPs mapped to the same VIP. We expect at least the latter to be a common case.

V. Discussion

In this section, we discuss some issues of the proposed architecture and sketch the solutions at high-level.

A. Scalability of LB Switch Load Balancing

As described so far, our architecture assumes that the LB switches are managed directly by the global manager. In particular, the global manager must consider all the switches whenever it allocates new or reallocates existing VIPs in order to (i) balance the traffic among the LB switches and ensure that the number of VIPs/RIPs allocated to every switch stays within the switches' limits. The rationale behind this design is that our architecture keeps the number of LB switches small: given our target of 300K applications with 3 VIPs and 20 RIPs per application, we need only $\max(((300K \times 3)/4000), ((300K \times 20)/16000)) = 375$ LB switches.

However, given the large number of applications, the complexity of the VIP allocation can still be high. Indeed, let A be the number of applications hosted in the data center, k the average number of VIPs assigned to each application, L the number of LB switches and R the number of access routers. Then the total number of possible ways to place the applications among the LB switches is $A * L^k$ (note that a switch may host multiple VIPs for the same application, and the VIPs of the same application on the same switches are not interchangeable), which translates to the order of 10^{13} states for 300K applications, 400 switches and three VIPs per application. Depending on the allocation algorithm, managing the load-balancing layer may potentially face a scalability problem.

Should this become an issue, we could again resort to a hierarchical approach to address it. We would divide LB switches into logical pods, each managed by its own LB switch pod manager. The global manager would allocate addresses to LB switch pods (to be further allocated within the pod by the pod manager) and also redistribute the switches among the switch pods to balance their size and hence the work of the switch pod managers.

B. Policy conflicts

Our architecture exposes a number of control knobs and policy objectives, and different objectives may interfere with

each other. For example, to balance the load among the access links, the DNS system might preferentially expose the VIPs advertised over lightly-loaded access links. However, if these VIPs are mapped to physical servers at pods with high utilization, we would try to *reduce* the amount of traffic going through these VIPs to avoid the overload of these pods. Thus the policies for balancing the load among the access links may conflict with the policies for balancing the load among the pods.

To resolve this conflict, we can add another layer of LB switches (which we call the demand-distribution layer) between the access connection layer and the load-balancing layer. In the resulting *two-LB-layer architecture*, the external VIPs of each application are configured at the LB switches at the demand distribution layer. Meanwhile, each external VIP is mapped to several middle-layer VIPs (*m-VIPs*) configured on LB switches of the load-balancing layer. To conserve m-VIPs (recall that each switch can handle only a limited number of VIPs), all external VIPs of a given application can map to the same set of m-VIPs. Finally, each LB switch in the load-balancing layer with an m-VIP maps this address to a group of RIPs configured on the servers. Note that m-VIPs and RIPs are both private.

The two-LB-layer architecture decouples load balancing of the access links from that of the server pods: selective VIP exposing now affects only the access links and the LB switches at the demand distribution layer, while load balancing of server pods affects only the LB switches at the load-balancing layer. Unfortunately, this benefit comes at the expense of extra load balancing switches at the demand distribution layer. Due to the the high price of the LB switches, we are investigating efficient algorithms and mechanisms based on the architecture of Figure 1 in our ongoing work.

VI. Related Work

Several new architectures for data centers have been proposed [2], [17], [8]. These approaches concentrate on intra-DC networks. By guaranteeing bandwidth between any host-pair within the data center and providing flat address space to all the hosts, these approaches provide a foundation for our work. The approach in [9] suggests to use common servers as load balancing elements. Our architecture assumes hardware switches as they are typically more reliable and require less maintenance.

Authors in [10], [26] propose distributed approaches to address the scalability problem in data centers. In particular, [10] mentions a hierarchical scaling method, which envisions a layer on top of cluster-level resource managers, each managing up to 32 hosts and 3000 VMs. Our pods are decoupled from physical connectivity, facilitating logical pod formation; this allows much larger pods and dynamic transfer of servers among the pods. Further, unlike the above works, we consider the scalability of load-balancing fabric itself as well as load balancing among access links of the data centers.

In addition to maximizing utilization, energy is another objective in resource management that has received significant attention (e.g., [27], [11], [21]). Balancing energy and performance considerations adds further complexity to resource management algorithms; however our general architectural framework fully applies to this resource management aspect.

VII. Conclusion

This paper outlines a scalable architecture that supports datacenter-wide resource management for elastic Internet applications in a mega data center. Our architecture includes a scalable load-balancing fabric and provides effective knobs to balance load among the applications, servers, access links, as well as the load-balancing components themselves – the pod resource managers and switches in the load-balancing fabric. In the future, we plan to investigate algorithms for the resource management in this environment.

References

[1] http://www.datacenterknowledge.com/archives/2008/05/07/microsoft-300000-servers-in-container-farm.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM*, 2008.

[3] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera. A stable network-aware VM placement for cloud systems. In *IEEE/ACM CCGrid*, 2012.

[4] T. Callahan, M. Allman, and M. Rabinovich. On modern DNS behavior and properties. *CCR*, 43(3):7–15, 2013.

[5] David Davis. Using vSphere hot-add to dynamically add CPU and RAM. http://www.petri.co.il/vsphere-hot-add-memory-and-cpu.htm, SEP 2011.

[6] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM*, 2011.

[7] A. Greenberg, D. A. Hamilton, J.and Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *CCR*, 39(1):68–73, 2008.

[8] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *ACM SIGCOMM*, 2009.

[9] A. Greenberg, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. Towards a next generation data center architecture: scalability and commoditization. In *ACM PRESTO*, 2008.

[10] A. Gulati, G. Shanmuganathan, A. Holler, and I. Ahmad. Cloud-scale resource management: Challenges and techniques. In *USENIX HotCloud*, 2011.

[11] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: saving energy in data center networks. In *USENIX NSDI*, 2010.

[12] Cisco Systems Inc. Cisco Catalyst 6500 series content switching module.

[13] M. Korupolu, A. Singh, and B. Bamba. Coupled placement in modern data centers. In *IEEE IPDPS*, pages 1–12, 2009.

[14] H. A. Lagar-Cavilla, J. A. Whitney, R. Bryant, P. Patchin, M. Brudno, E. de Lara, S. M. Rumble, M. Satyanarayanan, and A. Scannell. SnowFlock: Virtual machine cloning as a first-class cloud primitive. *ACM Trans. Comput. Syst.*, February 2011.

[15] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *IEEE INFOCOM*, pages 1–9, 2010.

[16] MidoNet: Rise above your physical network. http://www.midokura.com/midonet/.

[17] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM*, 2009.

[18] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan. On the Responsiveness of DNS-based Network Control. In *ACM SIGCOMM IMC*, 2004.

[19] P. Patel, Deepak Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, and N. Karri. Ananta: cloud scale load balancing. In *ACM SIGCOMM*, pages 207–218, 2013.

[20] H. Qian, E. Miller, W. Zhang, M. Rabinovich, and C. E. Wills. Agility in virtualized utility computing. In *VTDC*, 2007.

[21] A. Qureshi, R. Weber, H. Balakrishnan, J. V. Guttag, and B. V. Maggs. Cutting the electric bill for Internet-scale systems. In *SIGCOMM*, pages 123–134, 2009.

[22] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee. Application-aware virtual machine migration in data centers. In *IEEE INFOCOM*, pages 66–70, 2011.

[23] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *WWW*, 2007.

[24] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. DONAR: decentralized server selection for cloud services. In *ACM SIGCOMM*, 2010.

[25] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *USENIX NSDI*, 2007.

[26] Y. O. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady. Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *IEEE CLOUD*, 2010.

[27] A.J. Younge, G. von Laszewski, Lizhe Wang, S. Lopez-Alarcon, and W. Carithers. Efficient resource management for cloud computing environments. In *Green Computing Conf.*, pages 357–364, 2010.

[28] W. Zhang, H. Qian, C.E. Wills, and M. Rabinovich. Agile resource management in a virtualized data center. In *ACM WOSP/SIPEW*, 2010.

[29] X. Zhu, C. Santos, D. Beyer, J. Ward, and S. Singhal. Automated application component placement in data centers using mathematical programming. *Int. J. of Network Management*, 18(6):467–483, 2008.