Rethinking the Network Stack for Rack-scale Computers

Paolo Costa Hitesh Ballani Dushyanth Narayanan
Microsoft Research

Abstract

The rack is increasingly replacing individual servers as the basic building block of modern data centers. Future rack-scale computers will comprise a large number of tightly integrated systems-on-chip, interconnected by a switch-less internal fabric. This design enables thousands of cores per rack and provides high bandwidth for rack-scale applications. Most of the benefits promised by these new architectures, however, can only be achieved with adequate support from the software stack.

In this paper, we take a step in this direction by focusing on the network stack for rack-scale computers. Using routing and rate control as examples, we show how the peculiarities of rack architectures allow for new approaches that are attuned to the underlying hardware. We also discuss other exciting research challenges posed by rack-scale computers.

1 Introduction

While today's large-scale data centers such as those run by Amazon, Google, and Microsoft are built using commodity off-the-shelf servers, recently there has been an increasing trend towards server customization to reduce costs and improve performance [29, 31, 33]. One such trend is the advent of rack-scale computing.

1.1 Rack-scale computing

Rack-scale computers (RASCs) comprise 100s to 1,000s of micro-servers that are connected by a network fabric. Their emergence is due to two hardware innovations:

System-on-chip (SoC) integration combines cores, caches and network interfaces in a single die. SoCs are widespread on mobile platforms as they save power and space, and the same advantages also apply in the server domain. SoCs enable vendors to build *micro-servers*: extremely small server boards containing computation, memory, network interfaces, and sometimes flash storage. For instance, the Calxeda ECX-1000 SoC [26] hosts four ARM cores, a memory controller, a SATA interface, and a fabric switch onto a single die.

Fabric integration connects such micro-servers into a high-bandwidth low-latency network. Typically this is done by using a "distributed switch" architecture; micro-servers are connected via point-to-point links into a multi-hop direct-connect topology. The point-to-point links can simply be traces on a PCB back-plane and can run custom physical and link protocols that are not exposed to the micro-servers. They can thus offer high bandwidth (10–100 Gbps) and low per-hop latency (100-500 ns). A "fabric controller" on each node provides a NIC interface to the micro-server and also forwards packets for other micro-servers. Any topology that has a small number of links per node can be used; 2D and 3D torus are popular choices adapted from supercomputing architectures.

We use the term "rack-scale" because we do not expect this technology to scale to the entire data center. At the rack-scale, it is possible to achieve high bandwidth and low latency with low-dimensional topologies such as a 3D torus. Scaling beyond the rack, however, would *i)* introduce large over-subscription (due to the higher density and higher bandwidth per link compared to today) and *ii)* incur higher propagation delays (dictated by the speed of light and the additional switching latency), which would be unacceptable for many operations, e.g., remote memory access.

Early examples of RASCs have appeared on the market. For example, HP's Moonshot [30] is a 4.3 rack-units chassis with 45 8-core Intel Atom SoCs and 1.4 TB of RAM in a 3D torus topology. The AMD SeaMicro 15000-OP [34] stacks 512 cores and 4 TB of RAM within 10 rack-units using a 3D torus network fabric with a bisection bandwidth of 1.28 Tbps. Intel's proposed Rack-scale Architecture [28, 32] combines SoC and fabric integration with silicon photonics, which support link bandwidths of 100 Gbps and higher. RASC designs have also been proposed by the academic community such as the soNUMA [16] and Firebox [3] platforms.

Given the amount of innovation happening at the hardware level, it is important to understand the implications for the software stack (operating system, network and storage layers, and applications). In the rest of this paper, we explore the implications for the RASC network stack, and discuss the research challenges and opportunity lying ahead. We defer an exploration of the rest of the software stack to future work.

1.2 Rack-scale networking

A prominent feature of RASCs is the distributed switch architecture where each node functions as a small switch and forwards traffic from other nodes. This results in a multi-hop direct-connect topology, e.g., a 2D mesh or a 3D torus, characterized by high path diversity. Further, performance efficiency dictates "resource disaggregation", whereby each micro-server can access the resources at other micro-servers across the network [14]. Therefore, the same network fabric carries both IP and non-IP (e.g., memory and storage) traffic.

The rack's network fabric is a departure from today's data centers, which mostly use tree-like topologies. Also, while direct-connect topologies have been used in high performance computing (HPC), the disaggregated nature of resources and the multi-tenant environment makes RASC network traffic more diverse and unpredictable than in HPC clusters. Such differences mean that both the traditional TCP/IP stack and solutions from the HPC domain, e.g., [6–8], are ill-suited to this environment.

As a concrete example, in this paper we focus on routing and rate control in RASCs. High path diversity in rack topologies and the wimpy nature of micro-servers pose a challenge for the TCP-family of rate control protocols as they use a single path and impose high processing overhead [16]. Even multi-path extensions like MPTCP [19] only consider few tens of paths. This is roughly three orders of magnitude smaller than the number of paths available here. Dually, approaches based on per-flow network queues with back pressure do not scale to the expected number of applications in a rack.

In the next section, we sketch a new routing and rate control design that leverages a key characteristic of these topologies– their direct-connect nature. The basic idea is to use a variant of Valiant Load Balancing for routing which ensures that each micro-server has a local view of the rack's global traffic matrix, and can independently determine the sending rate for its flows. This approach eschews any path probing. Using simulations, we show it can simultaneously achieve both low flow completion time (at the tail too) and tiny queues.

We finish the paper by highlighting other research challenges (and opportunities) resulting from the peculiarities of RASCs. For each example, we make a case for new designs that are attuned to and leverage the characteristics of the underlying hardware.

2 Routing and rate control in RASCs

In this section, we sketch our design and show how the peculiar characteristics of RASCs enable approaches that are unfeasible in traditional switch-based networks. We begin by focusing on two very basic questions for a RASC network stack. For a flow between two nodes, what path should each packet take (*routing*) and what rate should the packets be sent at (*rate control*)?

2.1 Design goals

We adopt three key design goals–

(i). Leverage the *path diversity* prevalent in rack network topologies. Using multiple paths can increase the throughput of a flow significantly, e.g., by up to a factor of six for a 3D torus.

(ii). *Good load balance* across network links. Data center workloads often have skewed communication patterns [5] which can result in hotspots and bring down overall network throughput.

(iii). *Low network queuing*. While a standard goal for network design, this is particularly important here because the micro-servers have very limited buffers and the network may carry traffic that is latency-sensitive.

2.2 Routing

Routing across direct-connect topologies has been extensively studied in the the scientific computing and HPC literature [8]. Minimal routing protocols like equal-cost multi-path (ECMP) [15] and randomized packet spraying [10] route packets only along shortest paths. This ensures low propagation delay but at the expense of load imbalance across network links.

In our design, we select Valiant Load Balancing (VLB) [23] as the underlying routing protocol. VLB is a non-minimal routing protocol in that packets are not always routed along the shortest path. However, VLB has excellent load balancing properties and is agnostic to the input traffic matrix [11, 13]. It works by randomly selecting, at the source, an intermediate hop to which the packet must be routed through before reaching the actual destination. This uniformly spreads traffic across the rack, regardless of the specific traffic matrix. This is very important in our context, because it gives a lower bound on the throughput achieved by the network.

A key consequence of using VLB in a direct-connect topology is *global visibility*. Since VLB evenly spreads the packets of a flow across the entire fabric, nodes can obtain a good approximation of the rack's flow matrix, just by observing the headers of packets they forward. This does not require any additional control traffic. We exploit this property in our rate control design.

While global visibility is a useful property, with

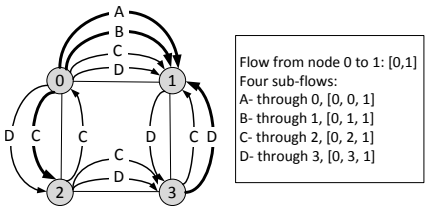

Figure 1: A flow from node 0 to 1 comprises four sub-flows *A*-*D*.

Link	*A*	*B*	*C*	*D*	Flow (total)
$0 \to 1$	$\frac{w}{4}$	$\frac{w}{4}$	$\frac{w}{8}$	$\frac{w}{8}$	$\frac{3w}{4}$
$1 \to 0$	0	0	0	0	0
$0 \to 2$	0	0	$\frac{w}{4}$	$\frac{w}{8}$	$\frac{3w}{8}$
$2 \to 0$	0	0	$\frac{w}{8}$	0	$\frac{w}{8}$
$1 \to 3$	0	0	0	$\frac{w}{8}$	$\frac{w}{8}$
$3 \to 1$	0	0	$\frac{w}{8}$	$\frac{w}{4}$	$\frac{3w}{8}$
$2 \to 3$	0	0	$\frac{w}{8}$	$\frac{w}{8}$	$\frac{w}{4}$
$3 \to 2$	0	0	0	0	0

Figure 2: Links weights for sub-flows *A*-*D*. The flow's total weight w is split evenly among the four sub-flows.

vanilla VLB, a new flow may need to send many packets before all nodes become aware of it. Specifically, the expected number of packets to be sent before all n nodes in the rack have been chosen as an intermediate hop at least once is given by $\Omega(n \log n)$ [27]. With n=512, it would take around 3,490 packets for all nodes to be chosen as the intermediate hop. To address this, we propose a VLB variant called *VLB+*. For each flow, we generate a random permutation of the n nodes in the rack which specifies the intermediate hops for the next n packets to be sent. Thus, every node is guaranteed to be chosen as an intermediate hop for a flow after it has sent n packets. Since nodes that forward a flow's packets from its source to the intermediate hop and on to the destination also become aware of the flow, in practice, the number of packets needed to ensure global visibility is smaller.

2.3 Rate control

The basic idea behind our rate control protocol is that given knowledge of the network topology and all active flows, each node can independently determine the load on each network link and hence, the fair sending rate for its flows. Thus, we transform the distributed congestion control problem into one of local rate calculation. While the rack's topology is relatively static, the set of active flows can change rapidly. We begin with the assumption that nodes are aware of the rack's current traffic matrix.

We first define a few terms. A flow is characterized by its source and destination, $[s,d]$. From a rate computation perspective, each flow comprises n sub-flows, one for each intermediate hop its packets are routed through. Each sub-flow is thus characterized by a three-tuple $[s,i,d]$, $i \in [1,n]$ where n is the number of nodes in the rack. The use of VLB+ at the routing layer imposes the following constraint on the rate computation– all sub-flows for a flow should be assigned the same rate. This is because VLB+ chooses intermediate hops in a randomized round-robin fashion. Thus, any mismatch between the rates allocated to the sub-flows will result in queues building up at the flow's source.

To achieve per-flow fairness, each flow is assigned the same weight w which, in turn, is equally distributed among its sub-flows. Thus, the weight for each sub-flow is $\frac{w}{n}$. We use information about how packets are routed to determine the weight of each sub-flow at each link in the topology. We illustrate this through the example 2x2 mesh topology in Figure 1. Consider a flow $[0,1]$ which comprises four sub-flows *A*–*D*. Sub-flow *A* uses node 0 as the intermediate hop and its packets are routed directly from node 0 to 1. Thus, this sub-flow has a weight of $\frac{w}{4}$ on the link from 0 to 1 and no weight at any other link. The same is true for sub-flow *B* which uses node 1 as its intermediate hop.

Sub-flow *C* uses node 2 as the intermediate hop. Packets from the source are routed along the link from node 0 to 2. From node 2, there are two equally short paths to the destination, $2 \to 0 \to 1$ and $2 \to 3 \to 1$. To capture the fact that the sub-flow's packets are routed along these paths with equal probability, we split its weight across these paths. The resulting per-link weights for all sub-flows are shown in Figure 2.

We can repeat the above process for all flows to determine per-flow weights at all network links. Given this setup, any node can independently compute the max-min fair rate for each flow through a simple iterative algorithm that we summarize here. At each iteration, we find the link with the highest total weight. This is the most bottlenecked link, and its capacity dictates the rates for all flows using the link. Specifically, we allocate the link's capacity among the flows across it in proportion to their weight at the link and mark these flows as allocated. To account for *host-limited* flows, we can add flow demands in the packet header. The rate assigned to each flow is then the minimum between its fair share and its demand. Finally, we remove the weights for all allocated flows from all network links and continue to the next iteration till all active flows have been allocated.

This simple design has a few advantages. First, it avoids the need to probe the network and induce congestion signals like packets drops and queuing to infer a flow's sending rate. High path diversity in RASCs makes the design of a congestion probing mechanism particularly challenging. Second, if nodes have an accurate view of the rack's traffic matrix, the rate allocation ensures both high network utilization and low queuing.

Similarly, we use a decentralized mechanism to garbage collect flow state, thus avoiding expensive global

coordination. For any given flow, by combining its per-link weights and the assigned rate, a node can locally estimate the next packet arrival time and set the timeout accordingly. If no packet arrives before the timeout expires, the flow is considered completed and removed from the list.

2.4 Preliminary results

We use simulations to evaluate our rack network design, hereon referred to as RASC-NET. While preliminary, the results indicate that our protocol is able to significantly decrease flow completion times while requiring only small packet queues in the network.

We use a packet-level network simulator to model a 512-node 3D torus. This is the same size and topology of the AMD SeaMicro 15000-OP. We assume a link bandwidth of 10 Gbps and we do not consider any computation overhead. For our experiments, we use a random permutation traffic matrix, in which we vary the traffic load, expressed as the fraction of sources in the network. All flows have a size of 10 MB and start at the same time.

We compare RASC-NET, comprising VLB+ and the rate control protocol described in Section 2.3, against three approaches. The first one, *TCP*, uses ECMP [15] as the routing protocol and TCP to control flow rates. This is representative of today's status quo. ECMP assigns different shortest paths to different flows between the same endpoints. The next two, *Ideal-M* and *Ideal-NM*, use random packet spraying (minimal routing) and VLB+ (non-minimal routing) respectively. With random packet spraying, a shortest path is randomly chosen for each packet in a flow. We denote them as "ideal" as they do not rate limit flows at the source but, instead, rely on unbounded per-flow queues at the nodes. This is impractical but provides upper bounds on the performance achievable by *any* rate control protocol for minimal and non-minimal traffic oblivious routing.

Figure 3 shows the 99^{th} percentile of the flow completion time (FCT) when varying the traffic load. The performance of *TCP* is limited by the fact the ECMP only uses a single path for each flow and, hence, cannot fully utilize the network capacity. In contrast, *Ideal-M* yields lower completion times as packets are allowed to use *all* shortest paths for a source-destination pair. However, especially for low traffic load, RASC-NET outperforms *Ideal-M* as the former can also use links not on the shortest path. As the number of flows increases, the gains from using non-minimal routing are partially offset by the increased channel load, and the gap between RASC-NET and *Ideal-M* narrows. Notably, RASC-NET performance is indistinguishable from *Ideal-NM*. This confirms that our rate control protocol achieves a good approximation of (ideal) unbounded flow queues.

Since RASC-NET does not probe the network to infer

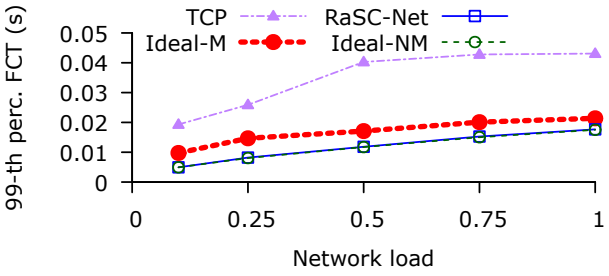

Figure 3: The 99^{th} percentile FCT against network load.

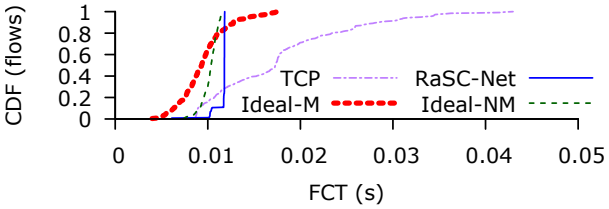

Figure 4: CDF of FCT for network load equal to 0.5.

the correct rates, it achieves tiny packet queues. In our experiments, across all values of the load, the maximum queue occupancy was 138 packets (203 KB of buffering). Low queue occupancy also reduces the amount of packet reordering occurring at the destination caused by the multi-path routing. For example, across all our experiments, the median size of the reordering buffer was 96 packets with a maximum value of 158 packets.

Finally, Figure 4 shows the CDF of the FCT for all four approaches when the network load is equal to 0.5. While both *Ideal-M* and *TCP* exhibit a long tail, in RASC-NET the FCT distribution is very narrow, which is particularly useful for data center workloads [9].

2.5 Discussion

While these results are encouraging, more work is needed to achieve a complete solution. Here, we highlight the key concerns and the solutions we are pursuing.

Short flows. Our design assumes that nodes are aware of the rack's current traffic matrix. For our 512-node topology, in the worst case it can take up to 511 packets for all nodes to be aware of a new flow. In data centers, however, most flows are only a few packets long. For example, in a typical data-mining workload [13], 80% of flows are less then 10KB. Yet, 95% of all bytes are in the 3.6% flows larger than 35MB. Thus, we could adopt an approach similar to HULL [1] and reserve some spare capacity to account for the short flows; a small spare capacity should absorb such flows while our rate-limiting mechanism ensures that medium-to-long flows do not congest the network.

Computation overhead. In our design, rack nodes recompute the rates whenever a new flow is detected or the timeout associated to an existing flow expires. In our case, however, the max-min computation is very cheap,

needing only a few iterations to converge. This is due to the use of VLB+ routing, which entails that each flow uses almost all links in the network. Consequently, the most bottlenecked link is actually the bottleneck for almost all active flows (e.g., 89.1% for the experiment in Figure 4). This can be optimized even further by stopping the computation after one iteration, thus trading-off a little utilization for reduced computation overhead.

Beyond per-flow fairness. Our approach can be extended to support sharing policies beyond per-flow fairness. For example, sources can include in the packet the *weight* and the *priority* of the flow. Nodes can then invoke the allocation algorithm over multiple rounds, one for each priority level. At each round, flows belonging to the corresponding priority level are allocated a rate in a weighted fashion. We expect that higher level fairness policies such as deadline-based [24] or tenant-based [18], can be mapped onto these two primitives, similar to pFabric [2].

Traffic-aware routing. VLB achieves good load balance at the expense of increasing the average path length. While this is a good trade-off in low-load regimes, it can be sub-optimal at high load or when the workload exhibits high locality. To compensate for this while still retaining the nice properties of VLB, we are considering using traffic-aware variants of VLB [21] which limit the amount of non-minimal traffic routing.

3 Research Directions

In this section, we briefly discuss some of the research questions that RASCs raise. Our intent is not to provide an exhaustive list but rather to give a taste of the exciting challenges and opportunities that lie ahead.

Converged fabric. Existing systems use a variety of interconnect technologies and protocols, e.g., QPI and HTX for inter-socket interconnection in NUMA systems, Ethernet or InfiniBand for inter-server communication, and storage area networks (SANs). By contrast, RASCs have a single fabric that is expected to carry different types of traffic. It is therefore worth investigating whether we can replace all these protocols with a single unified one and, in such case, what are the correct abstractions (e.g., packet-based vs. circuit-based and reliable vs. best-effort). Another important question is to understand how to share the network resources between different users and applications as well as between different classes of traffic. Recent proposals show that, to maximize utilization while ensuring fairness, each traffic type has different requirements; e.g., for memory traffic [17], storage traffic [20], and inter-server traffic [4, 18]. An open question is how such requirements and policies can be composed atop a converged fabric. Finally, the *mechanisms* used for enforcing these policies will likely vary according to the type of traffic. For instance, while a centralized controller might be a feasible option to handle storage traffic at this scale [22], the extremely low latency requirements of memory traffic will probably require decentralized mechanisms.

Inter-RASC network. Thus far, we have only focused on intra-RASC communication. It is important, however, to investigate how to interconnect multiple RASCs to form a large cluster. This includes both the physical wiring layout and the network protocols used to bridge between two RASCs. One simple option would be to just use traditional switches and tunnel RASC packets by encapsulating them inside Ethernet frames. While this would allow for a smooth transition from today's deployments, it has some limitations. First, given the high bisection bandwidth available within a rack, the only way to avoid creating high over-subscription would be to use high-radix switches with large back-plane capacity, in the order of (tens of) Terabits. This, however, would dramatically increase costs and it may even be unfeasible if 100+ Gbps links are to be deployed within a rack. Further, the need to bridge between the rack and the Ethernet domain would add further overhead and increase end-to-end latency. A more promising (albeit challenging) solution might be instead to directly connect multiple RASCs without using any switch, similar to [25].

Resource co-scheduling. As observed in the Introduction, we believe that the impact of RASC architectures extends beyond the network, to the entire stack. In particular, the small scale and the tight integration among resources provided by SoC designs create exciting opportunities for cross-resource approaches. As a concrete example, we are exploring the benefits of having a global scheduler that controls all resources (CPUs, memory, and network). For example, in case of network congestion, the scheduler can decide whether it is more convenient to redirect network flows, migrate the computation, migrate the data or a combination of all three. This differs from today's model in which the network controller operates largely independently from the job scheduler and the storage substrate. This also would require a new definition of fairness. While recent work on dominant resource fairness (DRF) [12] provides a promising starting point, it is not obvious how to extend it to capture the RASC disaggregated resource model [14].

4 Conclusions

We advocate the need for researchers and practitioners to reconsider existing software mechanisms and abstractions in light of the advent of rack-scale computing. In this paper, we started this process by considering the implications for network protocols. Our hope, however, is that this paper helps to create awareness about the unique opportunities posed by RASCs and can spur fruitful discussions in the community at large.

References

[1] ALIZADEH, M., KABBANI, A., EDSALL, T., PRABHAKAR, B., VAHDAT, A., AND YASUDA, M. Less Is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center. In *NSDI* (2012).

[2] ALIZADEH, M., YANG, S., SHARIF, M., KATTI, S., MCKEOWN, N., PRABHAKAR, B., AND SHENKER, S. pFabric: Minimal Near-optimal Datacenter Transport. In *SIGCOMM* (2013).

[3] ASANOVIC, K. FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers. In *FAST* (2014). Keynote.

[4] BALLANI, H., COSTA, P., KARAGIANNIS, T., AND ROWSTRON, A. Towards Predictable Datacenter Networks. In *SIGCOMM* (2011).

[5] BENSON, T., AKELLA, A., AND MALTZ, D. A. Network Traffic Characteristics of Data Centers in the Wild. In *IMC* (2010).

[6] CRAY INC. Modifying Your Application to Avoid Aries Network Congestion, 2013.

[7] CRAY INC. Network Resiliency for Cray XC30 Systems, 2013.

[8] DALLY, W., AND TOWLES, B. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

[9] DEAN, J., AND BARROSO, L. A. The Tail at Scale. *Communications of ACM 56*, 2 (2013).

[10] DIXIT, A. A., PRAKASH, P., HU, Y. C., AND KOMPELLA, R. R. On the Impact of Packet Spraying in Data Center Networks. In *INFOCOM* (2013).

[11] DOBRESCU, M., EGI, N., ARGYRAKI, K., CHUN, B.-G., FALL, K., IANNACCONE, G., KNIES, A., MANESH, M., AND RATNASAMY, S. RouteBricks: Exploiting Parallelism To Scale Software Routers. In *SOSP* (2009).

[12] GHODSI, A., ZAHARIA, M., HINDMAN, B., KONWINSKI, A., SHENKER, S., AND STOICA, I. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *NSDI* (2011).

[13] GREENBERG, A., HAMILTON, J. R., JAIN, N., KANDULA, S., KIM, C., LAHIRI, P., MALTZ, D. A., PATEL, P., AND SENGUPTA, S. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM* (2009).

[14] HAN, S., EGI, N., PANDA, A., RATNASAMY, S., SHI, G., AND SHENKER, S. Network Support for Resource Disaggregation in Next-generation Datacenters. In *HotNets* (2013).

[15] HOPPS, C. Analysis of an Equal-Cost Multi-Path Algorithm, 2000. IETF RFC 2992.

[16] NOVAKOVIC, S., DAGLIS, A., BUGNION, E., FALSAFI, B., AND GROT, B. Scale-out NUMA. In *ASPLOS* (2014).

[17] NYCHIS, G. P., FALLIN, C., MOSCIBRODA, T., MUTLU, O., AND SESHAN, S. On-chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects. In *SIGCOMM* (2012).

[18] POPA, L., KRISHNAMURTHY, A., RATNASAMY, S., AND STOICA, I. FairCloud: Sharing the Network in Cloud Computing. In *HotNets* (2011).

[19] RAICIU, C., BARRE, S., PLUNTKE, C., GREENHALGH, A., WISCHIK, D., AND HANDLEY, M. Improving Datacenter Performance and Robustness with Multipath TCP. In *SIGCOMM* (2011).

[20] SHUE, D., FREEDMAN, M. J., AND SHAIKH, A. Performance Isolation and Fairness for Multi-tenant Cloud Storage. In *OSDI* (2012).

[21] SINGH, A., DALLY, W. J., TOWLES, B., AND GUPTA, A. K. Locality-preserving Randomized Oblivious Routing on Torus Networks. In *SPAA* (2002).

[22] THERESKA, E., BALLANI, H., O'SHEA, G., KARAGIANNIS, T., ROWSTRON, A., TALPEY, T., BLACK, R., AND ZHU, T. IOFlow: A Software-defined Storage Architecture. In *SOSP* (2013).

[23] VALIANT, L. G., AND BREBNER, G. J. Universal Schemes for Parallel Communication. In *STOC* (1981).

[24] VAMANAN, B., HASAN, J., AND VIJAYKUMAR, T. N. Deadline-Aware Datacenter TCP (D^2TCP). In *SIGCOMM* (2012).

[25] WU, H., LU, G., LI, D., GUO, C., AND ZHANG, Y. MDCube: A High Performance Network Structure for Modular Data Center Interconnection. In *CoNEXT* (2009).

[26] Calxeda EnergyCore ECX-1000 ARM Server. `http://bit.ly/1nCgdHO`.

[27] Coupon collector's problem. `http://en.wikipedia.org/wiki/Coupon_collector's_problem`.

[28] Design Guide for Photonic Architecture. `http://bit.ly/NYpT1h`.

[29] Google Ramps Up Chip Design. `http://ubm.io/1iQooNe`.

[30] HP Moonshot System. `http://bit.ly/1mZD4yJ`.

[31] Intel, Facebook Collaborate on Future Data Center Rack Technologies. `http://intel.ly/MRpOMO`.

[32] Intel Rack Scale Architecture Overview. `http://ubm.io/1iejjx5`.

[33] Microsoft Contributes Cloud Server Specification to Open Compute Project. `http://bit.ly/1cyrIh3`.

[34] SeaMicro SM15000 Fabric Compute Systems. `http://bit.ly/1hQepIh`.