Expeditus: Distributed Load Balancing with Global Congestion Information in Data Center Networks

Peng Wang, Hong Xu
Department of Computer Science, City University of Hong Kong

ABSTRACT

We propose Expeditus, a distributed congestion-aware load balancing mechanism for Clos data center networks. The fundamental challenge in making load balancing congestion-aware is, how to collect real-time (in the order of RTT) congestion information from all possible paths, in a scalable and efficient manner. A naive solution requires each edge switch to have congestion information for $O(k^4)$ paths for a k-pod fat-tree, and recent proposals like CONGA only work for the two-tier leaf-spine topology. Expeditus relies on scalable one-hop information collection, where a switch only monitors buffer occupancy from and to its $k/2$ upstream neighbors, respectively. It further uses a two-stage path selection mechanism to aggregate relevant congestion information across switches and make near-optimal path selection decisions during TCP handshaking. We outline the basic idea of these mechanisms in this extended abstract. Preliminary ns-3 simulations demonstrate that Expeditus outperforms ECMP in fat-tree networks, and outperforms CONGA significantly in leaf-spine topology.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

Datacenter network; Load balancing; Congestion control; Distributed

1. INTRODUCTION

Clos topologies, such as fat-tree [1], are extensively used in data centers with a large number of equal-cost paths. In such a network, load balancing is critical to ensure network performance. ECMP is the *de facto* load balancing solution in practice that chooses a path based on hashing. ECMP is simple to implement and does not require per-flow state at switches. It however suffers from two well-known drawbacks, largely due to its *congestion agnostic* nature. First, hash collisions between mice flows and elephant flows cause long tail latency for mice [4, 9]. Mice flows collide with elephants on the same path, and as a result are often queued behind elephants in the egress ports. Second, hash collisions among elephant flows cause low throughput and bandwidth utilization, as the elephants have to share the same link when other links may be completely free [2, 5, 7].

CoNEXT Student Workshop '14, December 02-05 2014, Sydney, Australia

http://dx.doi.org/10.1145/2680821.2680825.

We present the preliminary design and evaluation of Expeditus[1], a distributed congestion-aware load balancing mechanism targeted at fat-tree topology. Intuitively, with congestion information—queueing delay of network paths—the load balancing algorithm can direct flows to better paths leading to better latency and throughput. As load balancing needs to be done for each and every flow, a distributed protocol is preferred for large-scale production deployments to cope with bursty traffic [3].

The fundamental challenge is that, to make the best load balancing decision for a flow, we need to know real-time (in the order of RTT) congestion information from all possible paths between the flow's source and destination, as congestion can happen in any part of the network and changes frequently [3, 4]. This is a daunting task in fat-tree, where there are $k^2/4$ unique paths between two edge switches in different pods, and an edge switch potentially communicates with $k^2/2-1$ edge switches. Thus, a naive solution requires storing $O(k^4)$ path information at each edge switch. More importantly, it is impossible to collect real-time congestion information for all these paths, either by passive means using packets to carry the queue length of the path they take, or by actively sending probing packets for measurements.

Expeditus relies on two novel designs, one-hop congestion information collection and two-stage path selection, to overcome this challenge. Specifically, each edge (aggregation) switch only monitors buffer occupancy of its $k/2$ egress ports as the northbound state, and $k/2$ egress ports at aggregation (core) switches connecting to it as the southbound state. One-hop information collection ensures real-time congestion state is available without scalability concerns or measurement overhead. Expeditus then applies a two-stage path selection method during TCP handshaking to aggregate relevant congestion information across switches and make near-optimal per-flow load balancing decisions. Thus, path can be decided for the following data packets.

We notice an independent recent work, CONGA [3], which also proposes distributed congestion-aware load balancing. The key difference between CONGA and Expeditus is that CONGA is designed for the simple leaf-spine topology, and collects congestion information on an end-to-end basis. Each packet opportunistically feeds back the congestion state of a random path in the reverse direction between the source and destination. As explained, such an approach does not work for the complex fat-tree topology. As we will further show in Sec. 3, even in leaf-spine networks this design provides marginal performance benefits over ECMP.

[1]Latin for "expedited," pronounced /ˈek-spə-ˌdɪtəs/ .

2. DESIGN

We sketch the design of collecting congestion information and two-stage path selection in a k-pod fat-tree.

2.1 Collecting Congestion Information

Each edge/aggregation switch monitors both the northbound direction, from itself to the upstream switches, as well as the southbound direction from upstream switches to itself. Clearly the northbound information can be easily obtained by polling the buffer occupancy of its own egress ports when necessary [6]. The southbound information is remote and needs to be transmitted by packets traversing those links.

When a packet is about to exit a core switch, the egress port buffer occupancy is stamped into its IP header together with the core switch's IP. The downstream aggregation switch maintains a southbound congestion information table (SCIT) with $k/2$ entries. When it receives the packet, it fetches the core switch's IP to infer the ingress port the packets comes from, and updates the corresponding entry with the latest congestion information. It updates core switch's IP with its own address, computes the egress port based on the packet's destination, stamps buffer occupancy and forwards it out. The edge switch then updates its SCIT in the same way.

Finally, each entry of the SCIT will timeout if there has been no update. When an entry times out, it resets to zero meaning there is assumed to be no congestion.

2.2 Two-stage Path Selection

Expeditus then applies a two-stage path selection method in TCP handshaking to make congestion-aware decisions. For a new flow, SYN and SYN-ACK is routed by ECMP as we do not have congestion information of remote switches to choose a better path yet. Upon receiving a SYN, the destination edge switch chooses the least congested path to the aggregation layer, by adding its local southbound information to the northbound information at the source edge switch carried in the SYN. Thus, the aggregation switches with the least combined congestion at the first and final hops are decided. This completes the first stage of path selection.

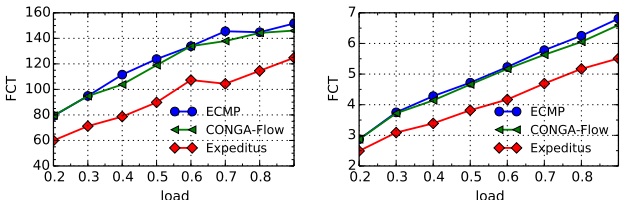
The second stage path selection is similar and involves the aggregation switch choosing its path to the core layer. When sent from destination host to source host, SYN-ACK will route through the destination and source aggregation switches determined in the first stage. When SYN-ACK reaches the source aggregation switch, it then chooses the best core switch by adding its local northbound information with the southbound information at the destination aggregation switch carried in the SYN-ACK. Two-stage path selection thus finds a near-optimal path based on instantaneous network state, without switches exchanging information and storing congestion information for all $O(k^4)$ paths.

3. EVALUATION

In this section, we evaluate Expeditus using ns-3 with realistic Clos topologies and empirical workload from a production web search cluster reported in [4]. We construct a 16-pod fat-tree [1] as the baseline fabric topology, which is widely used in data centers. To better compare our mechanism with CONGA, we also consider a two-tier leaf-spine topology, which uses one pod of the above fat-tree with edge and aggregation switches as the leaf and spine layers.

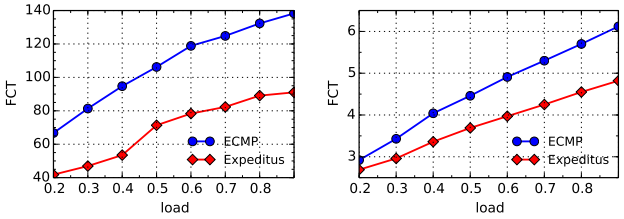
We begin with Expeditus's performance in leaf-spine network, compared with CONGA and ECMP. We note that flowlet switching itself also helps load balancing [8]. The performance gain of being congestion-aware is not isolated as CONGA includes flowlet switching. To make the comparison fair, we evaluate CONGA-Flow that works on a per-flow basis. From Figure 1, we find that Expeditus achieves favorable performance gains ranging from 25%–30% for all flows. More interestingly, we observe that benefits of CONGA-Flow are rather limited. In most cases CONGA-Flow is similar to ECMP. As we explained in Sec. 1, in CONGA-Flow, only the congestion state of one path is carried in a packet. Between two racks, in order to have complete congestion information of all n equal-cost paths, there needs to be at least n concurrent flows, where n equals the number of spine switches. This is difficult to satisfy, resulting in an incomplete view of the network state and little benefits compared to congestion-agnostic ECMP

We now turn to 16-pod fat-tree, comparing the performance of Expeditus with ECMP in a large 3-tier Clos network. Since CONGA has not been generalized to larger topologies like fat-tree topology, we do not implement CONGA in fat-tree topology. The results are shown in Figure 2. We observe that by providing better load balance with global congestion information, Expeditus provides salient flow completion times (FCT) gains with unmodified TCP. For mice flows less than 100 KB, Expeditus provides 35%–50% at the 99-th percentile over ECMP. Meanwhile, for elephants, Expeditus is also around 25% faster on average.



(a) (0,100KB]: 99-th percentile (b) (100KB, ∞]: Avg

Figure 1: FCT in leaf-spine network. Network oversubscription 2:1, port buffer 150 KB.



(a) (0,100KB]: 99-th percentile (b) (100KB, ∞]: Avg

Figure 2: FCT in a 16-pod fat-tree. Network oversubscription 4:1, port buffer 150 KB.

4. CONCLUSION

We have presented Expeditus, a network layer congestion-aware load balancing scheme. Expeditus collects necessary congestion information on a one-hop basis, and employs two-stage path selection to aggregate such information and route flows. Preliminary results show that Expeditus can improve network performance for both mice and elephant flows, and outperforms state-of-the-art. Our next step is to prototype Expeditus and evaluate it with advanced TCP variants, as well as to provide theoretical models to analyze its performance.

Acknowledgments

We thank Mohammad Alizadeh and Kai Chen for helpful feedback and suggestions.

5. REFERENCES

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

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proc. USENIX NSDI*, 2010.

[3] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proc. ACM SIGCOMM*, 2014.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proc. ACM SIGCOMM*, 2010.

[5] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz. Per-packet load-balanced, low-latency routing for Clos-based data center networks. In *Proc. ACM CoNEXT*, 2013.

[6] Cisco. Cisco nexus 3548 - active buffer monitoring. `http://www.cisco.com/c/en/us/products/collateral/switches/nexus-3548-switch/white_paper_c11-715895.pdf`, September 2012.

[7] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the impact of packet spraying in data center networks. In *Proc. IEEE INFOCOM*, 2013.

[8] S. Kandula, D. Katabi, S. Sinha, and A. Berger. Dynamic load balancing without packet reordering. *SIGCOMM Comput. Commun. Rev.*, 37(2):51–62, April 2007.

[9] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. DeTail: Reducing the flow completion time tail in datacenter networks. In *Proc. ACM SIGCOMM*, 2012.