☁Collectives: Towards Cloud-aware Collectives with Rank Reordering

Liang Luo, Jacob Nelson, Luis Ceze, Arvind Krishnamurthy

Abstract

Collectives are heavily used by distributed communication backends, but their performance is hurt by the non-uniform and dynamic network throughput of VMs, due to the topology of datacenter networks and multi-tenancy nature of the cloud environment.

In this paper, we present ☁Collectives (Cloud-aware collectives), a prototype tool that accelerates collectives by reordering the ranks of participating VMs such that the communication pattern dictated by the selected collectives operation best exploits the locality in the network. ☁Collectives is non-intrusive, requires no code changes nor rebuild of an existing application, and runs without support from cloud providers. Our experimental application of ☁Collectives on *allreduce* operations in public clouds results in a speedup of up to 3.7x in multiple microbenchmarks and 1.3x in real-world workloads of distributed training of deep neural networks and gradient boosted decision trees using state-of-the-art frameworks.

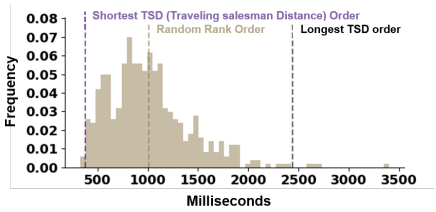

Figure 1: Performance distribution of *allreduce* task of 100MB data with ring algorithm varies widely with 500 random rank orders on Azure.

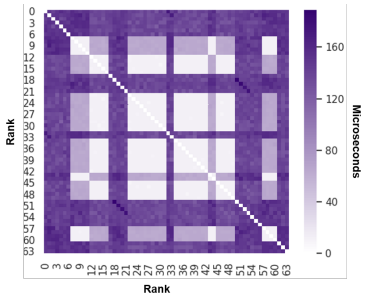

Figure 2: Pairwise RTT probe on 64 Azure Standard F64 v2 VMs shows non-uniform latency in point to point VM communication.

1 Introduction

Collective communication operations are an essential component of many data-parallel computation frameworks. Originally developed for high-performance computing frameworks such as MPI [60], they are now widely used for cloud-based distributed machine learning and big data analytics [59, 9, 1, 42, 33, 8, 54] workloads too. With increasingly more complex models [48, 44, 53] calling for larger data exchanges, and rapid deployment of faster accelerators [13, 25, 32] demanding more frequent exchanges, efficient execution of these workloads relies on good communication performance of collectives.

Unfortunately, achieving good collectives performance in a cloud environment is fundamentally more challenging than in an HPC world, because the user has no control over node placement, topology and has to share the infrastructure with other tenants. These constraints have a strong implication on the performance of collectives. As a result, the bottleneck of running these workloads on the cloud has shifted from computation to communication [61, 38, 37].

Consider a common practice of applying *allreduce* ring collectives, a popular algorithm, in the cloud context, where a randomly-ordered IP list (obtained through the provider) of VMs is used to form a virtual ring on which data is passed along, with i-th VM sending data to $i+1$-th VM. But do different ways of forming ring (through permutation of VMs in the list) exhibit the same performance? The answer is most likely no, as the ring that corresponds to shorter total hop cost will likely perform better (Figure 1). On the other hand, not all ways of forming rings achieve the same cost, because the *point to point communication cost (bandwidth, latency, or collectively referred to as locality in this work) is different across VMs* (Figure 2), due to the hierarchical structure of the datacenter network, and the dynamic nature of traffic from other tenants. Consequently, running collectives with a randomly-ordered list of VMs results in unpredictable and subpar performance.

Our work focuses on discovering a permutation of the IP list that exploits the network locality for efficient communication, in a completely transparent way, by minimizing the cost model of a given collectives parameterized with the actual hop cost. To do so, we need to (1) efficiently identify the underlying network bandwidth/latency constraints (or collectively, locality); (2) accurately build cost model for the collectives at hand; (3) effectively approximate the minimum of these complex cost functions.

This paper proposes ☁Collectives, a tool that uses accu-

rate network probes to discover locality within the underlying datacenter network, and uses it to solve a communication cost minimization problem with constraints, with the rank of each VM as the unknowns. We use reordered ranks as input to unmodified communication backends in microbenchmarks including OMB [21], Nvidia NCCL [6], Facebook Gloo [28] and real-world workloads of training deep neural networks with Pytorch/Caffe2 and gradient boosted decision trees using LightGBM [42, 33] and found a speedup of up to 3.7x in various *allreduce* operations and 1.3x in end-to-end performance across EC2 and Azure.

2 Background

We provide an overview of typical structures and performance implications for datacenter networks and a brief introduction to the various popular collectives operations.

2.1 Locality in Datacenter Network

Modern datacenters are hierarchical, with a group of machines connecting to a top of rack switch, which is in turn connected to upper-level devices [46, 29, 55, 35]. This particular topology induces locality [38], as the communication performance between two physical hosts is not the same. For example, VMs within the same rack have the best and stable performance, as the physical link is not shared. On the other hand, links between hosts residing in different racks are shared, and the communication performance depends on factors like hop count, link congestion, oversubscription ratio [19], and dynamic load. Traditionally, topology information is crucial for achieving optimal performance as many collectives implementations generate routines based on this information [23, 49, 50, 24, 37]. But in a cloud-environment, this information is not accessible.

Many attempts have been made to reconstruct the network topology, but mostly are infeasible in the cloud environment: [36, 17] rely on link layer protocols such as SNMP, [34, 16] assume homogeneous clusters with bare-metal or para-virtualized devices in a strict tree structure. More importantly, we will show a full understanding of actual physical topology is not necessary for a cloud environment, because with multiple tenants sharing the same hardware in the cloud, better static physical affinity does not always equal higher performance, as dynamic traffic load ultimately decides the cost of each link.

2.2 Collectives

Collectives works by decomposing a compound operation into a series of point to point operations between two nodes according to a predefined communication schedule. collectives most often appear in MPI contexts [56, 58, 52, 20, 15].

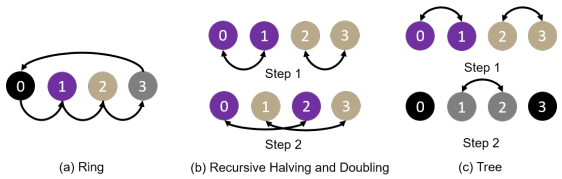

Figure 3: Rounds of communications (color-coded) in various popular algorithms used in collectives with 4 nodes.

Typically, all nodes in collectives participate in the communication, usually running symmetric tasks. collectives can be used in many tasks such as *(all)reduce, broadcast, (all)gather, and (reduce)scatter* [7], and it is thus impractical to individually optimize each task. Fortunately, many of the tasks can be decomposed into multiple stages of collectives primitives (e.g., *allreduce* can be decomposed into *reducescatter* followed by *allgather*, and these two stages are usually the reverse of each other), and therefore, we only need to focus on accelerating such primitives. Further, most of the primitives are implemented using a handful of well-known algorithms, and in this paper, we focus on accelerating the communications in these algorithms, rather than the actual tasks they are applied to. We now introduce these algorithms. We use N as the number of participating nodes, S as the amount of data to process per node.

Ring [51]. As shown in Figure 3(a), ring algorithms work by connecting nodes to form a virtual ring. Data is then passed along the ring sequentially. Ring algorithms require $O(N)$ steps to complete, sending $O(NS)$ amount of data.

Halving Doubling [58]. As shown in Figure 3(b), halving doubling works by recursively doubling the distance (in terms of rank ID) while halving the total amount of data sent in each round, requiring $O(log_2 N)$ steps to finish while also sending $O(NS)$ amount of data on the wire.

Tree. In a simple form, a single tree is built where data is transferred from leaves to the root and vice-versa [31]; in a more optimized setting, a pair of complementary binary trees are built to fully utilize the full bisection bandwidth [57], each sending and receiving $S/2$. For binary trees, $O(log_2 N)$ rounds of communication are required, also sending $O(NS)$ bytes on the wire.

BCube [3]. BCube is very similar to halving doubling from a structural perspective, in the sense that nodes are organized into a group of B peers. BCube operates in $O(log_B N)$ rounds, and each node in each round would peer with a unique node in another $B-1$ groups. Each node communicates $\frac{S}{B^i}$ amount of data in round i. BCube achieving a total bytes on wire of $O(\sum_{i=0}^{log_B N-1} \frac{S}{B^{i+1}})$.

3 Motivation

This section motivates the necessity of ☁Collectives by demonstrating the implication of rank order on the performance of cloud-based collectives. We start by highlighting

the asymmetric, non-uniform link cost in the cloud environment, by launching 64 Standard F64 VMs on the Azure cloud. We then run an in-house hybrid DPDK and ping-based [4] latency probe (§4.2) between each pair of VM node. The result is summarized in Figure 2 as a heatmap. We observe the pairwise round-trip latency can range from sub-10 to hundreds of microseconds.

We then proceed to examine the performance of *allreduce* operation using Facebook Gloo [2], running the Ring chunked algorithm with 512 Standard F16 VMs. To derive a performance distribution, we use 500 randomly generated rank orders to generate 500 samples, and each is the average runtime of a 10-iteration reduction of 100MB of data. The result is summarized in the yellow distribution in Figure 1. The performance of different rank orderings of VM varies drastically, ranging from 330ms to 3400ms, with a mean of 1012ms and a standard deviation of 418ms. Now we have established the profound influence of rank ordering of VM nodes on the performance of collectives algorithms, the goal of this paper is to derive an approximately optimal rank-ordering given a selected collectives algorithm such that it maximizes the performance.

4 Design and Implementation

We now describe ☁Collectives, a tool that takes in a list of VM nodes and a target algorithm, accurately and efficiently probes their pairwise distance, and uses that information to construct a rank order of VMs that attempts to minimize the total cost of communication.

4.1 Cost Models for Collective Algorithms

☁Collectives builds a cost model $\mathbb{C}_{\mathbb{O}}$ for each popular algorithm used in collectives $\mathbb{O}$, parameterized with the number of participating nodes N and size S. This section details the cost models for popular algorithms. We use $c_{i,j}(S)$ to refer to the cost for transferring S amount of data from node i to j. We further define $MAX_{i=0}^{j}(f(i)) = MAX(f(0), ..., f(j))$. We assume N a power of 2 to simplify explanation, and allow arbitrary rank r to alias to canonical rank $(r+N)$ mod N.

Ring. The cost model of the ring algorithm is the sum of the cost of each hop when traversing the ring:

$$\mathbb{C}_r(N, c, S) = \sum_{i=0}^{N-1} c_{i,i-1}(S)$$

Having Doubling. The cost of halving doubling is the sum of costs for each round of communication, which in turn is the max cost of all communications in that round.

$$\mathbb{C}_{hd}(N, c, S) = \sum_{i=0}^{log_2 N-1} MAX_{j=0}^{\frac{N}{2}-1} c_{j,j+2^i}(\frac{S}{2^{i+1}})$$

Tree. The cost of running tree algorithms depends on the number of trees and how trees are constructed. The total cost is the maximum cost of all trees, which is in turn determined by the maximum cost of each subtree. We provide a cost model for a popular variant of tree algorithm: double binary tree as used in [5].

$$\mathbb{C}_{dbt}(N, c, S) = T(0, N-1, S)$$

where $T(i, j, S)$ is expressed recursively:

$$T(i,j,S) = \begin{cases} 0 \textbf{ if } i \geq j \\ MAX(c_{\frac{i+j}{2}, \frac{3i+j}{2}-1}(\frac{S}{2}) + T(i, \frac{i+j}{2}-1), \\ c_{\frac{i+j}{2}, \frac{i+3j}{2}+1}(\frac{S}{2}) + T(\frac{i+j}{2}+1, j)) \textbf{ otherwise} \end{cases}$$

Similarly a mirrored tree is built by decrementing each node's rank in the tree without changing the tree structure.

BCube. The cost of running the BCube algorithm is similar to halving doubling, except in each round, each node communicates with $B-1$ peers, instead of 1.

$$\mathbb{C}_b(N, c, S, B) = \sum_{i=0}^{log_B N-1} MAX_{j=0}^{\frac{N}{B}-1} MAX_{k=1}^{B} c_{j,j+kB^i}(\frac{S}{B^{i+1}})$$

4.2 Probing for Pairwise Distance

We need to determine values for $c_{i,j}(S)$ with end-to-end measurements. We first consider commonly used, simple linear model: $c_{i,j} = LAT_{i,j} + \frac{S}{BW_{i,j,S}}$ where $LAT_{i,j}$ being the one-direction latency from i to j, and $BW_{i,j,S}$ the bandwidth achieved with data size of S. There are immediate challenges of deriving $BW_{i,j}$ correctly: first, $BW_{i,j,S}$ varies depending on the size of packet size S being transferred: e.g., on a 10Gbps link it is unlikely to saturate full bandwidth while sending small packets. Figure 4 (left) shows the how bandwidth varies with the size of the buffer in a point to point *iperf* (TCP, using DCTCP [12]) test on two 30Gbps D64 nodes on Azure. It is cumbersome to create such profile for each pair of VMs; second, even if a profile like this is constructed, it may still fall short when multiple streams are competing for bandwidth: the streams do not share the bandwidth equally, but rather, one stream can consistently outperform the other in a long time trace, as shown in Figure 4 (right) with 3 D64 nodes on Azure, when both of them can achieve similar throughput when run individually. It seems intractable to derive an accurate BW given S and a set of competing streams. Third, many algorithms operate in chunked mode, allowing overlapping of sending (of processed elements) and receiving (of unprocessed elements), and the simple model does not capture this in the first place.

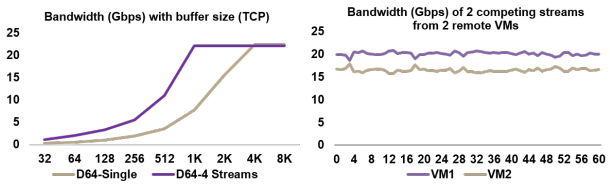

Figure 4: Left: TCP throughput depends on the buffer size and the number of concurrent streams. Right: while TCP is designed to be fair, an empirical 60s trace in the cloud shows the two streams connecting to the same VM from two other remote VMs do not share the bandwidth equally.

It is both beneficial and interesting to accurately model throughput behavior in a multi-stream environment [47, 12, 45, 39, 30], but that subject is worthy of its own topic. Instead, we compromise by dropping the bandwidth component in the model, leaving only the latency component. The rationale behind this stems from the well-known theoretical TCP bandwidth model of $BW = O(\frac{MSS}{RTT\sqrt{p}})$ [41] given constant drop rate p and window MSS. The fact that higher latency induces lower bandwidth in TCP streams lets us approximate costs by only probing for latency.

To accurately and efficiently probe for pairwise latency, we built an in-house DPDK based echo tool, leveraging network enhancement provided by the clouds [43, 14]. Probing of N nodes can finish in N rounds, with each round probing for $N/2$ pairs of VMs. At round r, node i sends $10k$ probes to $(i+r+1)$ mod N and responds to $10k$ probes from $(i-r-1+N)$ mod N sequentially. To derive an accurate reading, we take the RTT of 10th percentile. Each probe is a UDP packet with a 32-bit payload that encodes sequence number and round id for fault tolerance. When DPDK cannot be used, we use *fping*, a ICMP Echo-based latency probing tool. For each entry in c, we update $c_{i,j} \leftarrow MAX(c_{i,j}, c_{j,i})$ to make it symmetric.

4.3 Minimizing the Cost Model

We parameterize the cost model with values of probed c. To derive a rank ordering that minimizes $\mathbb{C}_{\mathbb{O}}$, we perform the following transformation: let set of variables $\mathbb{R}$ defined as $r_i, i \in [0, N-1]$ be a permutation of $[0, N-1]$ to be solved, and we replace each $c_{i,j}$ with c_{r_i,r_j}. We can then establish a bijection from the original rank ordering to the desired order $r_i \leftrightarrow i$ once r_is are solved. We flatten $c_{i,j} \leftrightarrow c'_{iN+j}$ to use theory of arrays to allow direct solving with conventional optimizing SMT solvers such as Z3 [22, 27].

Unfortunately, we find solvers inefficient, perhaps due to the non-convex, non-linear nature of the objective function and a large search space ($N!$). Instead, we take a two-stage process. The first step employs a range of stochastic search techniques such as simulated annealing [18], with a few standard heuristics (e.g., permuting a random sub-array, permuting random pairs) for obtaining neighboring states and a timeout. When the search returns with an initial result C_0, we generate an additional SMT constraint $\mathbb{C}_{\mathbb{O}} < C_0$ to better guide pruning for solvers. We let the solver continue to run for a few minutes, and we either find a better solution or will use C_0 as the final value. The end-product of this process is a rearranged list of VMs.

Setup	Azure	EC2
Gloo Ring 100MB	0.58	0.78
OpenMPI Ring 100MB	0.81	0.94

Table 1: Spearman correlation coefficient between predicted performance from cost model and actual performance.

5 Evaluation

We evaluate ☁Collectives with a series of microbenchmarks from various communication backends and real-world applications that use collectives. We represent speedup by comparing the performance we get from the best rank order and the worst rank order. We avoid comparing with the original rank order because it is random, and has a wide performance distribution (Figure 1).

5.1 Experimental Setup

Our experiments are conducted on two public clouds, Azure and EC2. We enable network acceleration on both clouds and set TCP congestion control protocol to DCTCP. We include microbenchmarks that exercise ring, having doubling, double binary tree, and Bcube algorithms. All experiments run on Ubuntu 19. We focus our evaluation on one of the most important tasks in collectives, *allreduce* for its popularity and generality.

5.2 Prediction Accuracy of Cost Model

While the goal of the cost model is not to predict the actual performance, but rather, it should preserve the relative order of performance, i.e., $p_{pred}(\mathbb{R}_1) < p_{pred}(\mathbb{R}_2) \implies p_{real}(\mathbb{R}_1) < p_{real}(\mathbb{R}_2)$ should hold true for as many pairs of (R_i, R_j)s as possible. We demonstrate this for ring based collectives by generating 10 different rank orders, with the i-th order approximately corresponds to the $10i$-th percentile in the range of costs found by the solver. We obtain performance data for Facebook Gloo and OpenMPI running OSU Benchmark on 64 F16 nodes on Azure and 64 C5 nodes on EC2. We then compute Spearman [10] correlation coefficient between the predicted performance and the actual performance for each setup (Table 1).

5.3 Microbenchmark Performance

We evaluate ☁Collectives's efficacy with microbenchmarks of algorithms introduced in 2. We report a mean speedup of 20 iterations. We run all benchmarks with 512 F16 nodes

on Azure, except for NCCL, which runs on 64 P3.8xLarge GPU nodes on EC2. Specifically, we set $B = 4$ for BCube; for NCCL, we use a single binary tree reduction for small buffers and a ring for large buffers. In all benchmarks, we reduce a buffer of 100MB, except for Nvidia NCCL, where we reduce a small buffer of 4B to trigger the tree algorithm.

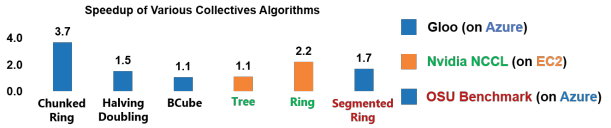

Figure 5: Speedups achieved by just using the rank ordering produced by our tool in various algorithms across multiple distributed communication backends at a large scale.

Figure 5 shows a summary of speedups achieved using ☁Collectives on these benchmarks, ranging from 1.1x-3.7x, with the ring-family algorithms benefiting the most. We speculate the reason for the effectiveness is that they have a much wider performance distribution, as each permutation of the order can potentially generate a different performance (cost of each hop is on the critical path); they also have simpler cost model, allowing the solvers to quickly navigate the objective landscape. On the other hand, halving doubling, BCube, and tree algorithms have complex objectives – sum of maximums, resulting in a narrower performance distribution because mutation of the ordering may not change the cost at all if the mutation does not cause the critical path to change.

5.4 End-to-end Performance Impact on Real-world Applications

Speedup Distributed Gradient Boosted Decision Tree Training. We evaluate ☁Collectives's impact on LightGBM [33], a gradient boosted decision tree training system. We use data parallelism to run *lambdarank* with metric *ndcg*. Communication-wise, this workload runs two tasks: *allreduce* and *reducescatter*, and they are called sequentially in each split of the iteration. At our scale of 512 nodes, LightGBM automatically chooses to use halving and doubling for both *reducescatter* and *allreduce*. We use a dataset that represents an actual workload in our commercial setting with 5K columns and a total size of 10GB for each node. We train 1K trees, each with 120 leaves. We exclude the time it takes to load data from disk to memory and report an average speedup of 1000 iterations. ☁Collectives generated rank orders speed up training by 1.3x.

Speedup Distributed Deep Neural Network Training. We show ☁Collectives's effectiveness on distributed training of DNNs with Caffe2/Pytorch, on 64 EC2 p3.8xLarge nodes with data parallelism and a batch size of 64/GPU. We train AlexNet on the ImageNet dataset. Since our ☁Collectives does not change computation and only improves communication efficiency of the *allreduce* operation at iteration boundary, we report speedup of training in terms of images/second, averaged across 50 iterations. We use the ring chunked algorithm, which achieves the best baseline performance, and the ☁Collectives-optimized rank ordering of VM nodes achieves a speedup of 1.2x.

6 Further Discussion

Cloud-aware MPI. We share the same goals as in [26, 11], as well as the use of end-to-end measurement to derive distance matrix. Our work, however, differs in the following way: (1) our work requires no change to existing framework or application; (2) [26] uses a two-stage, hierarchical scheme that sequentially invokes two collectives algorithms to leverage locality at rack-level, but adds additional latency. [26] evaluated only on simpler MPI primitives such as *gather/scatter/reduce* and achieved 13-27% speedup on 32 EC2 nodes, while we can achieve better performance at a much larger scale, running higher-level tasks such as *allreduce*; (3) additionally, we demonstrated actual end-to-end application-level speedups, while [26] ran only microbenchmarks and [11] did an analytical evaluation.

Integration with MPI. Most MPI distributions dynamically select underlying collectives algorithm based on the input size. This creates a challenge if all collectives algorithms share the same node rank and use the same MPI communicator. Code change is required to enable integration: on the MPI side, different collectives requires different communicator object to allow for different ordering. Alternatively, applications can fall back to use low-level MPI send/receive APIs, which is used to implement compound operations such as *allreduce* (a common practice [33]), and allows for individual rank ordering for each collectives algorithm.

Complements to Cost Models. While building accurate cost models is difficult, we can dynamically adjust rank ordering with help from tools such as TCP_INFO [40] that monitors link properties such as latency and bandwidth. Since we know the communication pattern, we can determine the critical path in a straightforward manner and find bottleneck transfer between node n_i and n_j in the system. From there, we can find a n_k to replace n_i such that the replacement results in a minimized cost objective.

Adapting to Dynamic Traffic. The above mechanism can be applied to adapt to dynamic network load in the cloud environment. The framework, however, must support the dynamic change of node ranks, which is possible and should come with a small cost as a full mesh of connections can be established beforehand among all nodes.

References

[1] Extend mxnet distributed training by mpi allreduce - mxnet - apache software foundation. https://cwiki.apache.org/confluence/display/MXNET/Extend+MXNet+Distributed+Training+by+MPI+AllReduce. (Accessed on 01/15/2020).

[2] Github - facebookincubator/gloo: Collective communications library with various primitives for multi-machine training. https://github.com/facebookincubator/gloo. (Accessed on 01/24/2020).

[3] gloo/algorithms.md at master facebookincubator/gloo. https://github.com/facebookincubator/gloo/blob/master/docs/algorithms.md. (Accessed on 01/26/2020).

[4] Home - dpdk. https://www.dpdk.org/. (Accessed on 01/24/2020).

[5] Massively scale your deep learning training with nccl 2.4 — nvidia developer blog. https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/. (Accessed on 01/25/2020).

[6] Nvidia collective communications library (nccl) — nvidia developer. https://developer.nvidia.com/nccl. (Accessed on 01/19/2020).

[7] pages.tacc.utexas.edu/~eijkhout/pcse/html/mpi-collective.html. http://pages.tacc.utexas.edu/~eijkhout/pcse/html/mpi-collective.html. (Accessed on 01/22/2020).

[8] Spark-mpi: Approaching the fifth paradigm - databricks. https://databricks.com/session/spark-mpi-approaching-the-fifth-paradigm. (Accessed on 01/15/2020).

[9] Writing distributed applications with pytorch pytorch tutorials 1.3.1 documentation. https://pytorch.org/tutorials/intermediate/dist_tuto.html. (Accessed on 01/15/2020).

[10] Spearman Rank Correlation Coefficient, pages 502–505. Springer New York, New York, NY, 2008.

[11] M. Alfatafta, Z. AlSader, and S. Al-Kiswany. Cool: A cloud-optimized structure for mpi collective operations. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pages 746–753, July 2018.

[12] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In Proceedings of the ACM SIGCOMM 2010 Conference, SIGCOMM 10, page 6374, New York, NY, USA, 2010. Association for Computing Machinery.

[13] Amazon. Amazon ec2 f1 instances. https://aws.amazon.com/ec2/instance-types/f1/. (Accessed on 03/04/2019).

[14] Amazon. Enable and configure enhanced networking for ec2 instances. https://aws.amazon.com/premiumsupport/knowledge-center/enable-configure-enhanced-networking/. (Accessed on 01/06/2019).

[15] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir. Ccl: A portable and tunable collective communication library for scalable parallel computers. IEEE Transactions on Parallel and Distributed Systems, 6(2):154–164, 1995.

[16] D. Battre, N. Frejnik, S. Goel, O. Kao, and D. Warneke. Evaluation of network topology inference in opaque compute clouds through end-to-end measurements. In 2011 IEEE 4th International Conference on Cloud Computing, pages 17–24, July 2011.

[17] Y. Bejerano, Y. Breitbart, M. Garofalakis, and Rajeev Rastogi. Physical topology discovery for large multisubnet networks. In IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428), volume 1, pages 342–352 vol.1, March 2003.

[18] D. Bertsimas and J. Tsitsiklis. Simulated annealing. Statistical Science, 8, 02 1993.

[19] K. Bilal, S. U. Khan, J. Kolodziej, L. Zhang, K. Hayat, S. A. Madani, N. Min-Allah, L. Wang, and D. Chen. A comparative study of data center network architectures. In ECMS, 2012.

[20] E. K. Blum, X. Wang, and P. Leung. Architectures and message-passing algorithms for cluster computing: Design and performance. Parallel Computing, 26(2-3):313–332, 2000.

[21] D. Bureddy, H. Wang, A. Venkatesh, S. Potluri, and D. K. Panda. Omb-gpu: A micro-benchmark suite for evaluating mpi libraries on gpu clusters. In J. L. Träff, S. Benkner, and J. J. Dongarra, editors, Recent

Advances in the Message Passing Interface, pages 110–120, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[22] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In International conference on Tools and Algorithms for the Construction and Analysis of Systems, pages 337–340. Springer, 2008.

[23] A. Faraj, P. Patarasuk, and X. Yuan. Bandwidth efficient all-to-all broadcast on switched clusters. In 2005 IEEE International Conference on Cluster Computing, pages 1–10, Sep. 2005.

[24] A. Faraj and X. Yuan. Message scheduling for all-to-all personalized communication on ethernet switched clusters. In 19th IEEE International Parallel and Distributed Processing Symposium, pages 10 pp.–, April 2005.

[25] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In Proceedings of the 45th Annual International Symposium on Computer Architecture, pages 1–14. IEEE Press, 2018.

[26] Y. Gong, B. He, and J. Zhong. Network performance aware mpi collective communication operations in the cloud. IEEE Transactions on Parallel and Distributed Systems, 26(11):3079–3089, Nov 2015.

[27] Google. Or-tools — google developers. `https://developers.google.com/optimization`. (Accessed on 01/26/2020).

[28] P. Goyal, P. Dollr, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2017.

[29] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, , P. Lahiri, D. Maltz, and and. Vl2: A scalable and flexible data center network. Association for Computing Machinery, Inc., August 2009. Recognized as one of "the most important research results published in CS in recent years".

[30] S. Ha, I. Rhee, and L. Xu. Cubic: a new tcp-friendly high-speed tcp variant. ACM SIGOPS operating systems review, 42(5):64–74, 2008.

[31] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2592–2600, 2016.

[32] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA '17, pages 1–12, New York, NY, USA, 2017. ACM.

[33] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In NIPS, 2017.

[34] J. Lawrence and X. Yuan. An mpi tool for automatically discovering the switch level topologies of ethernet clusters. In 2008 IEEE International Symposium on Parallel and Distributed Processing, pages 1–8, April 2008.

[35] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya. IncBricks: Toward in-network computation with an in-network cache. SIGOPS Oper. Syst. Rev., 51(2):795–809, Apr. 2017.

[36] B. Lowekamp, D. OHallaron, and T. Gross. Topology discovery for large ethernet networks. SIGCOMM Comput. Commun. Rev., 31(4):237248, Aug. 2001.

[37] L. Luo, J. Nelson, L. Ceze, A. Phanishayee, and A. Krishnamurthy. Parameter hub: A rack-scale parameter server for distributed deep neural network training. In Proceedings of the ACM Symposium on Cloud Computing, SoCC 18, page 4154, New York, NY, USA, 2018. Association for Computing Machinery.

[38] L. Luo, P. West, J. Nelson, A. Krishnamurthy, and L. Ceze. Plink: Discovering and exploiting locality for accelerated distributed training on the public cloud. In MLSys 2020, 2020.

[39] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Y. Sanadidi, and M. Roccetti. Tcp libra: Exploring

rtt-fairness for tcp. In I. F. Akyildiz, R. Sivakumar, E. Ekici, J. C. d. Oliveira, and J. McNair, editors, NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet, pages 1005–1013, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[40] M. Mathis, J. Heffner, and R. Reddy. Web100: extended tcp instrumentation for research, education and diagnosis. ACM SIGCOMM Computer Communication Review, 33(3):69–79, 2003.

[41] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. ACM SIGCOMM Computer Communication Review, 27(3):67–82, 1997.

[42] Q. Meng, G. Ke, T. Wang, W. Chen, Q. Ye, Z.-M. Ma, and T.-Y. Liu. A communication-efficient parallel algorithm for decision tree. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 1279–1287. Curran Associates, Inc., 2016.

[43] Microsoft. Create an azure virtual machine with accelerated networking — microsoft docs. https://docs.microsoft.com/en-us/azure/virtual-network/create-vm-accelerated-networking-cli. (Accessed on 01/06/2019).

[44] Microsoft. Turing-nlg: A 17-billion-parameter language model by microsoft - microsoft research. https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/. (Accessed on 05/25/2020).

[45] S. Molnr, B. Sonkoly, and T. Trinh. A comprehensive tcp fairness analysis in high speed networks. Computer Communications, 32:1460–1484, 08 2009.

[46] 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 SIGCOMM, 2009.

[47] K. Ogura, Z. Su, and J. Katto. Implementation experiments to evaluate a new tcp congestion control supporting loss-fairness. In 2010 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR 2010), pages 1–6, June 2010.

[48] OpenAI. Ai and compute. https://openai.com/blog/ai-and-compute/. (Accessed on 05/25/2020).

[49] P. Patarasuk, A. Faraj, and Xin Yuan. Pipelined broadcast on ethernet switched clusters. In Proceedings 20th IEEE International Parallel Distributed Processing Symposium, pages 10 pp.–, April 2006.

[50] P. Patarasuk and X. Yuan. Bandwidth efficient all-reduce operation on tree topologies. In 2007 IEEE International Parallel and Distributed Processing Symposium, pages 1–8, March 2007.

[51] P. Patarasuk and X. Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. Journal of Parallel and Distributed Computing, 69(2):117–124, 2009.

[52] R. Rabenseifner. Optimization of collective reduction operations. pages 1–9, 06 2004.

[53] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. Zero: Memory optimizations toward training trillion parameter models, 2019.

[54] J. Rehr, F. Vila, J. Gardner, L. Svec, and M. Prange. Scientific computing in the cloud. Computing in Science and Engineering, 12:34 – 43, 07 2010.

[55] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. Computer Communication Review, 45:123–137, 2015.

[56] P. D. Sack. Scalable Collective Message-passing Algorithms. PhD thesis, Champaign, IL, USA, 2011. AAI3503864.

[57] P. Sanders, J. Speck, and J. L. Traff. Two-tree algorithms for full bandwidth broadcast, reduction and scan. Parallel Comput., 35(12):581594, Dec. 2009.

[58] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in mpich. Int. J. High Perform. Comput. Appl., 19(1):4966, Feb. 2005.

[59] A. Vishnu, C. Siegel, and J. Daily. Distributed tensorflow with mpi, 2016.

[60] D. W. Walker and J. J. Dongarra. Mpi: a standard message passing interface. Supercomputer, 12:56–68, 1996.

[61] B. Zhang, Y. Ruan, and J. Qiu. Harp: Collective communication on hadoop. In 2015 IEEE International Conference on Cloud Engineering, pages 228–233, 2015.