Topology-aware Parallel Data Processing: Models, Algorithms and Systems at Scale

Spyros Blanas
The Ohio State University

Paraschos Koutris
Univ. of Wisconsin–Madison

Anastasios Sidiropoulos
Univ. of Illinois–Chicago

ABSTRACT

The analysis of massive datasets requires a large number of processors. Prior research has largely assumed that tracking the actual data distribution and the underlying network structure of a cluster, which we collectively refer to as the topology, comes with a high cost and has little practical benefit. As a result, theoretical models, algorithms and systems often assume a uniform topology; however this assumption rarely holds in practice.

This necessitates an end-to-end investigation of how one can model, design and deploy topology-aware algorithms for fundamental data processing tasks at large scale. To achieve this goal, we first develop a theoretical parallel model that can jointly capture the cost of computation and communication. Using this model, we explore algorithms with theoretical guarantees for three basic tasks: aggregation, join, and sorting. Finally, we consider the practical aspects of implementing topology-aware algorithms at scale, and show that they have the potential to be orders of magnitude faster than their topology-oblivious counterparts.

1. INTRODUCTION

Even though a parallel data processing task can utilize millions of compute cores in a datacenter, data manipulation algorithms so far have been analyzed in a topology-agnostic manner: common assumptions are that the data distribution is uniform and that the underlying network topology is a fully connected graph with bidirectional links. At its essence, the assumption is one of homogeneity: every compute core has roughly the same data processing capability and can communicate with every other node with roughly the same latency and bandwidth.

In reality, modern datacenters have a complex network structure and heterogeneous compute resources. On the network side, large clusters often have "fat tree" networks with oversubscribed links. In a fat-tree topology, the bandwidth within a rack is much higher than the bandwidth across racks. High-radix topologies such as CLOS are gaining popularity in commercial datacenters [1], and high-performance computers like BlueGene have featured hypercube topologies in the recent past. The compute side also exhibits divergent performance profiles between multi-cores, lean many-cores like Xeon Phi and massive parallelism in GPUs. The picture gets even more complicated in a cloud computing setting where the infrastructure is shared between concurrent tasks with weak performance isolation.

We posit that **making data processing topology-aware** is a significant and achievable research milestone to scale data analysis to large-scale computers. Optimizing for complex topologies is a challenging task that requires tackling open research questions that span theory, algorithms and practice. At the theoretical level, the opportunity is to devise models that jointly capture compute and communication costs, and discover tighter lower bounds for specific topologies. At the algorithmic level, the opportunity is to develop approximations and heuristics to achieve these lower bounds. At the practical level, the opportunity is to develop a system prototype to demonstrate that topology-aware data processing outperforms the state of the art in practice.

This research trajectory is motivated by encouraging preliminary results that we have obtained for parallel aggregation [22]: Even in a trivial network topology, one can prove that it is hard to approximate the plan that minimizes the network cost of a parallel aggregation within any constant factor. This is in stark contrast to the complexity of similar questions in classical network models, which admit efficient algorithms. Yet in practice, a greedy topology-aware heuristic performs aggregation up to 3.5× faster than the state of the art in parallel data processing. Our research vision is to deliver similar results for other common data processing operations and in non-trivial topologies. We identify three thrusts in realizing this vision.

Thrust 1: Models. From a modeling perspective, prior parallel computation models cannot sufficiently capture certain intricacies of complex parallel systems. We propose to develop a *theoretical model* that will capture the cost of computation in a topology-aware manner, which is something that current theoretical parallel models cannot do. We propose a new theoretical model that captures the cost of communication and computation in a complex, heterogeneous network topology.

The vertex-capacitated model can incorporate different facets of network behavior, including congestion, compute node heterogeneity, and bandwidth limitations. We then plan to investigate how we can model different types of commonly used network topologies, cost functions and node capabilities in our model. Then, based on the proposed model,

10th Annual Conference on Innovative Data Systems Research (CIDR '20) January 12-15, 2020, Amsterdam, Netherlands.

one can develop lower bounds that will shed light on the hardness of the algorithmic problems, as well as on the effect of network topology in the efficiency of communication for the data processing tasks at hand.

Thrust 2: Algorithms. From an algorithmic perspective, the focus is on three fundamental data processing tasks that form the backbone of any data-to-knowledge pipeline: *aggregation*, *joins* and *sorting*. Since the standard way of modeling a network is through a graph, developing algorithms that minimize the cost for the aforementioned data processing tasks is tightly related to optimization problems in graph theory, such as routing and graph partitioning. The new theoretical model leads to several exciting algorithmic questions, the main of which is: Given access to an approximation of a certain statistic on the data distribution, how do we obtain a data transfer and computation protocol with minimum cost for a specific data processing task and network topology?

Thrust 3: Systems. The systems thrust connects theory to practice and measures performance improvement in the field. The key research questions pertain to (1) discovering network topology information in a hosted environment, (2) overlapping computation with communication during topology-aware query processing, including offloading computation in the network. The goal is to demonstrate that topology-aware algorithms can be implemented efficiently in practice, in support of our thesis that one can capture some, hitherto untapped, power of scale in modern datacenters.

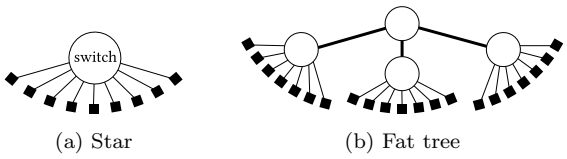
2. THRUST 1: TOPOLOGY-AWARE MODEL

Network Model. We model the network topology using a directed graph $G = (V, E)$. Each directed edge in E represents a network link, where the direction follows the flow of data on the link. We distinguish a subset of nodes in the network $V_C \subseteq V$ to be *compute* nodes: these can store data and perform computation on their local data. We assume that G is strongly connected, such that each compute node can communicate with all other compute nodes.

Computation. A parallel algorithm $\mathbb{A}$ in our model proceeds in sequential *rounds* (or *phases*). In the beginning, each compute node v holds part of the input I, denoted $X_0(v) \subseteq I$. The goal of the algorithm is to compute a query (or function) over the input I. We use $X_i(v)$ to denote the data stored at compute node $v \in V_C$ after the i-th round completes, where $i = 1, \ldots, r$. At every round, the compute nodes first perform some computation on their local data. Then, they communicate by sending data to other compute nodes in the network. We assume that for a data transfer from compute node u to compute node v, the algorithm must explicitly specify the routing path(s).

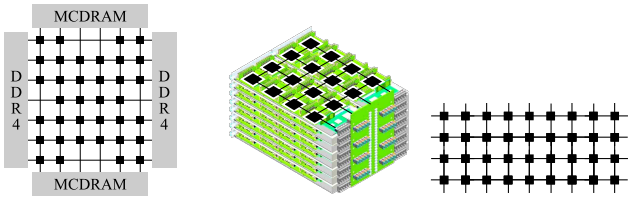
Cost Model. Since algorithms proceed in sequential rounds, we decompose the cost of the algorithm, denoted $\mathsf{cost}(\mathbb{A})$, as the sum of the costs for each round i, $\mathsf{cost}(\mathbb{A}) = \sum_{i=1}^{r} \mathsf{cost}_i(\mathbb{A})$. In order to model $\mathsf{cost}_i(\mathbb{A})$, we introduce a cost function $f_e : E \to \mathbb{R}$ for each link $e \in E$. Let $Y_i(e)$ denote the total data that is routed through link e during round i. Then, our *edge-capacitated model* measures the cost as

$$\mathsf{cost}_i(\mathbb{A}) = \max_{e \in E(G)} f_e(|Y_i(e)|).$$



(a) Star (b) Fat tree

Figure 1: Popular tree topologies.



(a) Intel KNL (b) Cray XC40 (c) 2D torus

Figure 2: Examples of planar (a), shallow-minor-free (b) and bounded genus (c) graphs.

For example, we associate with each link e a *capacity* $w_e > 0$, which captures the link bandwidth, and define $f_e(x) = x/w_e$. In this case, the cost of each round is the cost of transferring data through the most bottlenecked link in the network graph. The edge-capacitated model does not take into account computation, and hence captures the cost when computation is fast and communication is the bottleneck.

We note that the proposed model can capture the computation cost by performing a simple transformation of the underlying graph that adds virtual nodes for each compute node, and then encodes compute costs as edge weights. It is also possible to define a vertex-capacitated version of our model, where there is a cost function f_v associated with each compute node, and the cost is defined by taking the maximum cost over all nodes. Although the vertex-capacitated model would be more general, it is unclear if it would provide any stronger insights for topologies of practical interest.

Network Topologies. Several network routing problems are known to admit improved solutions when the underlying topology has a certain desirable structure [2, 8]. Even though our model supports general topologies, results that are tailored to the following network structures are of immense practical interest.

Trees. Perhaps the simplest topology is that of a *star network*, as shown in Figure 1a. At the microarchitectural level, CPU cores commonly exchange data through a shared last-level cache, which implicitly forms the center of a *star topology*. Designing optimal algorithms for basic tasks such as aggregation or sorting is nontrivial even in this apparently simple network. A special class of a tree topology is the popular *fat tree* topology in Figure 1b. In a fat tree topology, the capacity of each link increases closer to the root.

Planar graphs and their generalizations. Many popular network topologies are planar, such as meshes, grids, and trees of rings. Many-core processors such as the older Intel Knights Landing and the new 2nd generation Intel Scalable Processor platform interconnect cores in a *grid* [16] (see Figure 2a). Many interesting topologies are "nearly-planar", such as the 2-dimensional torus shown in Figure 2c. Many of these graphs can be modeled using the language of graph minor theory [28]. Furthermore, algorithmic results on minor-free graphs often extend to more general classes, such as *shallow-*

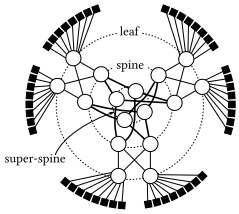


Figure 3: CLOS-5 topology, a high-radix network topology.

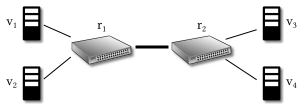


Figure 4: Parallel aggregation example.

minor-free graphs [26]. Figure 2b depicts the topology of the blade servers of a Cray XC40, which is obtained by connecting several 2-dimensional grids along a "spine"; the resulting graph is not minor-free, but shallow-minor-free.

High-radix network topologies. Warehouse-scale computers use *high-radix* networks. Many cloud vendors use a CLOS-5 network topology [1] (Figure 3), the Cray XC40 computers use the Dragonfly topology,while the IBM Blue Gene/Q features a multi-dimensional torus topology.

3. THRUST 2: ALGORITHMS

The theoretical model gives rise to several new algorithmic problems. This section focuses on three tasks that are ubiquitous in data processing: aggregation, joins, and sorting.

3.1 Parallel aggregation

Prior work by the authors proposed GRASP, a GReedy Aggregation Scheduling Protocol, which is a heuristic algorithm for the parallel aggregation problem [22]. Despite its apparent simplicity, GRASP performs favorably in experiments. The algorithm constructs a set of transmission pairs, chosen greedily such that data is aggregated along network paths that connect nodes with common aggregation keys to avoid unnecessary retransmissions of the same key. The intuition is best illustrated through an example.

Example 1. Topology-aware Parallel Aggregation *Consider a network with four compute nodes $\{v_1, v_2, v_3, v_4\}$ and two routing nodes $\{r_1, r_2\}$, as shown in Figure 4. The links that connect the compute to the routing nodes have a bandwidth of one tuple per time unit, while the link between the two routing nodes has twice the bandwidth. Suppose v_1 stores tuple t_1, v_2 stores tuple t_2, and v_3 stores tuple t_3. We want to compute the sum over all the tuples in the network, and want the aggregation result in node v_4.*

We consider two different algorithms and measure their cost. Algorithm $\mathbb{A}_1$ is topology-oblivious and needs only one round. $\mathbb{A}_1$ sends all tuples to v_4 concurrently for v_4 to perform the aggregation. Since the link (r_2, v_4) has to transfer three tuples, the cost is 3 time units and thus $\mathsf{cost}(\mathbb{A}_1) = 3$. Algorithm $\mathbb{A}_2$ operates in two rounds. In the first round, node v_3 sends tuple t_3 to v_4, and node v_1 sends t_1 to v_2. The compute node v_2 now computes the partial sum between t_2, t_3 and sends the result to v_4 in the second round. The cost of each round is 1 in this case for a total cost of $\mathsf{cost}(\mathbb{A}_2) = 2$, a $1.5\times$ speedup over the topology-oblivious algorithm.

Towards an $O(\log^2 n)$-approximation for aggregation on star networks. The GRASP algorithm described above is competitive in experiments, yet its performance lacks a provable theoretical guarantee. One approach to obtain an algorithm for the aggregation problem with a provable guarantee is as follows: We partition the compute nodes into *buckets* $Z_0, Z_1, \ldots, Z_{\log b}$, where Z_i contains all nodes with an incident link of capacity in the range $[2^i, 2^{i+1})$. For each i, we pick a *representative* node $z_i \in Z_i$, and create an instance Φ_i of the all-to-one aggregation problem where every node Z_i must send its data to z_i. In order to solve each of these smaller instances $\Phi_0, \ldots, \Phi_{\log b}$, it suffices to consider the case of all-to-one aggregation for star networks where each link has the same capacity, up to a factor of two. We refer to these networks as *uniform stars*. We obtain a $O(\log n)$-approximation algorithm for uniform stars as follows:

1. **Computing an aggregation round.** In each round i of the protocol, we define the set $A_i \subseteq Z_i$ of *active nodes* to be the ones who have not transmitted their input yet. In the first round all nodes are active. We construct an auxiliary complete graph Γ_i with vertex set A_i. For each edge $\{x, y\}$ in Γ_i, we associate a *cost* $c(x, y) \geq 0$ to be equal to the size of the union of the sets of x and y. We compute a min-cost matching M_i in Γ_i. For each edge $\{x, y\} \in M_i$, we transmit either $x \to y$, or $y \to x$, depending on whether $|x|$ or $|y|$ is the smallest.
2. **The multiple-round protocol.** The cost of one round is at most the cost of an optimal solution. Since the number of active nodes decreases by a factor of two in each round, there are at most $O(\log n)$ rounds, and hence the cost of the resulting schedule is a $O(\log n)$-approximation.

Combining this $O(\log n)$-approximation on uniform stars with the reduction outlined above, we obtain a $O(\log b \cdot \log n)$-approximation for general stars. This can be further improved to a $O(\log^2 n)$-approximation using a more careful bucketing scheme.

Towards provably good algorithms for general network topologies. Our plan for extending the above algorithm to the case of general network topologies is as follows:

From stars to fat trees. A fat tree of depth $d = 1$ is a star, for which we have already described an $O(\log^2 n)$-approximate algorithm. An aggregation problem on a fat tree with $d > 1$ can be decomposed into a sequence of d families $\mathcal{F}_1, \ldots, \mathcal{F}_d$ of aggregation instances. For each $i \in [d]$, the instances in $\mathcal{F}_i$ aggregate from all nodes at level $d-i+1$ to the nodes at level $d-i$; specifically, each node at level $d-i$ aggregates the data of its children. Concatenating the resulting aggregation protocols, one obtains a $O(d \log^2 n)$-approximate solution for the original aggregation instance.

From fat trees to general networks. Räcke's seminal work on oblivious routing [27] provides a powerful framework for obtaining approximation algorithms for routing and partitioning problems on graphs. At the heart of Räcke's approach is a hierarchical graph decomposition scheme which partitions the graph into a set of clusters, and each cluster is decomposed recursively. When contracting each cluster to a single vertex, one obtains an *expander graph*. It is natural to ask whether a $\log^{O(1)} n$-approximation algorithm can be obtained for the case of expander graphs via an approach that uses multicommodify flow techniques.

3.2 Join Processing

The second fundamental task of interest is the join between two relations, as it can be the stepping stone for considering multi-way joins, and joins followed by aggregations.

Binary Join. To compute the equi-join $R \bowtie S$, typical topology-agnostic algorithms use either a *parallel hashing*

algorithm, or partition the largest relation and broadcast the smallest one. Parallel hashing maps the tuples by hashing using the values of the join attribute. The default goal of the hashing procedure is to uniformly distribute the load. However, this approach is suboptimal even for simple network topologies, such as a star topology, or when the initial distribution of the input data is skewed across the compute nodes. It can be shown that one can easily take topological information into account even for a simple star network and obtain good speedup by carefully constructing a partitioning function (see Section 5). However, finding the best partitioning function is non-trivial when the topology is more complex than a star, and especially so when the topology is not a tree and hence there are multiple routing paths to the destination. Both are intriguing aspects for further study.

Algorithms also need to consider the case where the join has *data skew*. In such a case, it is necessary to treat the values of the join attribute that appear very frequently differently, similar to techniques from topology-agnostic parallel algorithms [5]. In the extreme, the join becomes equivalent to computing a *cartesian product*. The optimal algorithm that computes the cartesian product in a uniform star topology [19] does not extend to other topologies, hence it is necessary to design new techniques to achieve optimal performance in non-uniform topologies. Prior literature has proposed algorithms (such as flow join [29]) that treat frequent join keys differently, but the theoretical properties of these practical solutions are not well understood.

Multi-way Joins. For queries with multiple joins, there are two possible approaches. A baseline solution computes a multi-way join as a sequence of binary joins and optimizes each binary join separately. One can hope to do better by exploiting the join structure. For example, in the case of *acyclic queries*, semi-joins can be used to eliminate redundant tuples across all relations involved. The second approach seeks to optimize the query as a whole, applying similar ideas to recent work in parallel join processing [4, 5]. A possible outcome would be algorithms that come with worst-case theoretical guarantees on the communication cost, in analogy to the single-machine setting [25]. Concretely, consider the 3-way join $R(A,B) \bowtie S(B,C) \bowtie T(C,A)$. Instead of computing the query as a sequence of two binary joins, one can compute the join in a single push, by reshuffling the data such that the join can be performed locally thereafter.

3.3 Sorting

A parallel sorting algorithm takes as input a relation partitioned across the compute nodes. It then reshuffles the tuples of the relation, and returns a sorted order of the compute nodes, $v_0, v_1, \ldots, v_n$, such that for any tuples t in v_i, t' in v_j, whenever $i < j$ we have $t \leq t'$.

In the context of network-agnostic parallel models, several theoretically optimal sorting algorithms have been proposed [9, 13]. However, these algorithms are complex and see limited practical use. A promising direction is to build upon a simple framework called *Parallel Sort by Regular Sampling*, PSRS [32], which is commonly used in massively parallel settings. PSRS uses sampling to find $n-1$ *splitters*, which are elements that split the table into n intervals of roughly equal size. Each interval is assigned to one compute node, and the input tuples are partitioned to the appropriate node in one round according to the interval they belong to. Finally, the data is locally sorted.

To adapt PSRS to our topology-aware model, one must design a sampling procedure that takes topology into account: for example, one can sample non-uniformly from compute nodes so as to exploit links with high bandwidth, or avoid routing data through links with low bandwidth. Furthermore, the ability to push sampling down into the routing layer is particularly promising (see Section 4). The challenge is to ensure that the intervals created by the splitters will vary in size depending on network constraints (e.g., link bandwidth) and compute constraints (e.g., processing speed, CPU core count and memory capacity).

4. THRUST 3: A SYSTEM PROTOTYPE

A number of practical considerations lie in the path towards making topology-aware algorithms practically usable.

Topology discovery

The algorithms presented thus far assume transparency of the network structure, network routing decisions and network performance. However, in practice, the structure and the performance of a network may not be known in advance or may change dynamically. The question that arises is how can an optimization procedure estimate salient network metrics to accelerate data processing.

Semi-transparent deployments. The first step is investigating topology discovery techniques for shared infrastructure in a non-adversarial environment. Examples include on-premises clusters and scientific computing facilities. In such deployments, infrastructure providers are often willing to share telemetery on network, compute and I/O utilization through tools such as Ganglia, Darshan and TOKIO. The main challenge in this environment is addressing the weak performance isolation between concurrent workloads. A promising avenue is to replace the fixed edge weights of the network model (see Section 2) with a probability function based on the available information on the predicted compute and network capacity.

Opaque deployments. The next step is considering deployments where the infrastructure provider is unwilling to share information such as network capabilities, routing decisions and VM placement information. This is common when deploying applications in the cloud. A system can implement an array of techniques to discover the topology, bandwidth and compute capabilities of the cluster in this common deployment scenario.

1. **Bandwidth sensing with pair-wise communication.** All-to-all communication is inherently non-scalable. In practice, algorithms need to limit the number of open connections per node. The most judicious use of resources is using one connection per node per phase. (This constraint has been used in prior work to minimize network contention [30].) In this setting, it suffices to measure the pair-wise bandwidth through a benchmarking procedure that is executed at system startup. The benchmark will construct a throughput matrix B that stores all pair-wise bandwidths and will be used during query optimization.
2. **Topology approximation techniques.** Another avenue of investigation considers how to approximate the network topology, specifically using sparsification and hierarchical tree approximation. Sparsification starts by assuming the network structure is a complete graph and then progressively removes edges such that clusters in the

graph can be easily discerned. The round-trip message latency can be a predictor of the number of "network hops" of each network path and can be used for sparsification. The hierarchical tree approximation probabilistically approximates any finite metric space with a tree metric space with a polylogarithmic distortion [3].

Overlapping communication and computation

The model assumes that the performance of an operation along a network path is the minimum of the network and compute capabilities of every node in the path. In other words, the model assumes that computation and communication can perfectly overlap. As the network performance in modern clusters approaches memory speeds [7], systems need to carefully consider how to achieve this property in practice. Open questions pertain to when should transfers be scheduled, and how can multiple cores share a NIC.

When to schedule data transfers. Connection management can take significant time for fast networks with RDMA capabilities, as this entails locating and pinning memory for DMA operations. The setup time is in the order of hundreds of milliseconds [23]—in other words, setting up a connection takes as much time as transmitting a few GB of data. In practice, these substantial startup and shutdown costs will determine the performance of short queries. A number of design alternatives can ameliorate this cost:

1. **Staged scheduling.** Scheduling communication in stages directly maps to the communication model that has been presented in Section 2.
2. **All-at-once scheduling.** The argument for scheduling all connections at once is that processes will naturally block until all data have been received. All-at-once scheduling allows transfers to start as early as possible.
3. **Fixed lookahead scheduling.** A promising strategy involves scheduling data transfers for up to k phases ahead. One can vary the lookahead factor at runtime to limit how many open connections need to be maintained, so that k is smaller for phases that require many open connections.

Effectively sharing the network adaptor (NIC). Fast networks support kernel bypass, which allows applications to directly communicate with the NIC through memory-mapped NIC device registers. However, this places the application in charge of determining how to most efficiently share the NIC across queries.

1. **Busy polling.** The challenge with busy polling is that the system needs to determine *a priori* how many CPU cores need to be designated as receivers and as senders to achieve peak data processing throughput. This determination is non-trivial and may vary during execution as the compute intensity of the receiving and sending pipelines changes. Naive solutions like oversubscribing threads do not work well with fast networks whose performance sharply decreases with high contention.
2. **Notification-based mechanisms.** A system can also rely on notifications for intra-thread communication. Although a thread wakeup is a costly event, the cost amortizes over data transfers that move substantial amount of data in one request. Event-based methods will also be relevant in compute-heavy data processing pipelines. The challenge is that notification-based mechanisms are well-suited for events at the millisecond frequency; events at the microsecond frequency are notoriously challenging.
3. **Dynamic.** Another strategy is to use a single thread to poll the network adapter and dispatch requests to sleeping threads. The single polling thread merely dispatches requests and does not touch incoming data. Threads are woken up using notification-based mechanisms when sufficient work has accumulated.

5. PRACTICAL EXAMPLES

This section focuses on two practical problems, network interference and topology-aware joins, where the theoretical model suggests that significant speedups can be obtained by leveraging the topology.

5.1 Network interference degrades a single link

Large datacenters are shared between multiple users, and tasks may interfere with other concurrently-running jobs when accessing shared resources such as the network. Tasks that interfere often experience a transient performance degradation. Consider a cluster with 4 nodes in a star topology, where a single downlink experiences a performance degradation due to interference. This performance degradation can be encapsulated in the network topology, as shown in Figure 5. We compute a natural join between two relations R, S. The total input size (in tuples) is $N = |R| + |S|$. Suppose that the initial distribution is such that each node holds N/p data, assigned at random.

MODEL. The model is an asymmetric star with p compute nodes. All links send with the same bandwidth w, and can receive with bandwidth w_i. Assume $\min_i w_i \leq w$. The standard topology-oblivious hash join algorithm will be bottlenecked by the slowest link and thus cost $C_{obl} \approx N/(p \min_i w_i)$.

ALGORITHM. A topology-aware algorithm can hash to node i with probability $w_i / \sum_i w_i$. Assuming that $w \geq \sum_i w_i/p$, the cost of this algorithm is $C_{opt} = N/\sum_i w_i$. The speedup is $\sum_i w_i/(p \min_i w_i)$, which grows in proportion to the slowdown of the slowest link.

SYSTEM. We implemented and deployed the topology-aware join algorithm in a star network with 4 compute nodes, and placed 10GB of data in each node at random. The nodes are interconnected with a 1Gbps link in this experiment. Downlink interference was introduced through a background process on a single node, which performed a multi-TB dataset download and was kept active for the duration of the experiments. Figure 6 shows the performance of the interference-oblivious and the topology-aware algorithms, as well as the predicted cost by the model (dashed line). The vertical axis shows the network transfer time for each join

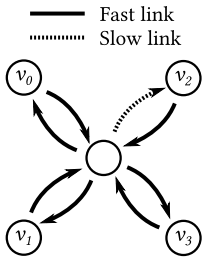


Figure 5: Performance degradation in downlink of v_2 due to interference.

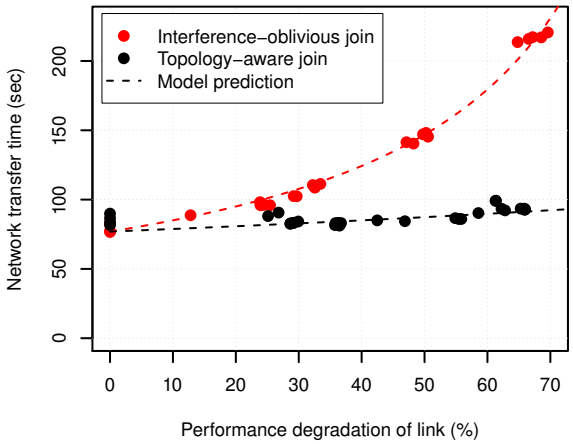


Figure 6: The topology-aware join is 2.3× faster than the interference-oblivious join when there is substantial performance degradation (70%) due to link interference.

algorithm, and the horizontal axis shows link interference as a percentage of the link capacity. Link interference was obtained by measuring how much data the background process downloaded during the network time of the algorithm, averaging per second, and dividing by the link capacity. We find that the time for the interference-oblivious join is increasing rapidly as all nodes are waiting for the slowest node to complete the data transmission and start local join processing. The topology-aware algorithm re-distributes data to faster nodes proportional to the available bandwidth, and is as much as 2.3× faster when there is substantial performance degradation (70%) due to link interference. The topology-aware algorithm is robust across different levels of interference, as its network time is within 20% of the time when there is no interference.

5.2 Joins in oversubscribed tree topologies

Tree network topologies are often oversubscribed at the root, hence data transmissions that need to cross the root may be slower than local transmissions. Fiugre 7 shows such an example: note how the transmission $v_0 \to v_1$ must use the slower links from/to the root, while $v_0 \to v_2$ only involves local, fast links. Assume we want to compute a natural join between two relations R, S, where $|R| < |S|$. Assume that each node holds equal input data, N/p, where $N = |R| + |S|$.

Model. Consider a tree with p compute nodes. All links have the same bandwidth w in both directions. The nodes are split into two groups of $p/2$ nodes, connected by a single slow link (path). A repartition-based hash join algorithm will incur a cost of $N/4w$, since half the data will have to cross the single link in the middle with probability $1/2$.

Algorithm. One can deploy a smarter, topology-aware strategy to join R and S that moves more data in the fast, local links of the tree but judiciously sends data over the slow link through the root: The S tuples are hashed only within their group and do not cross the root, while the R tuples are hashed across both groups. Now, the middle link will have a cost of $|R|/2$, while the other links will have a cost of about N/p. Hence, the cost will be $(1/w) \cdot \max\{|R|/2, N/p\}$. The speedup between the two algorithms is $O(\min\{N/|R|, p\})$. This speedup will grow as S becomes much larger than R, but will never exceed p.

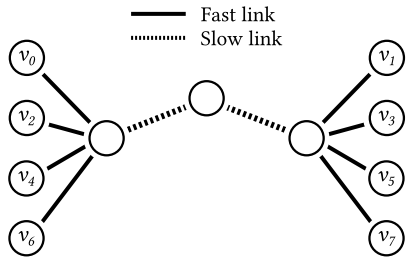


Figure 7: Tree topology with fast links within groups but slow links to the root.

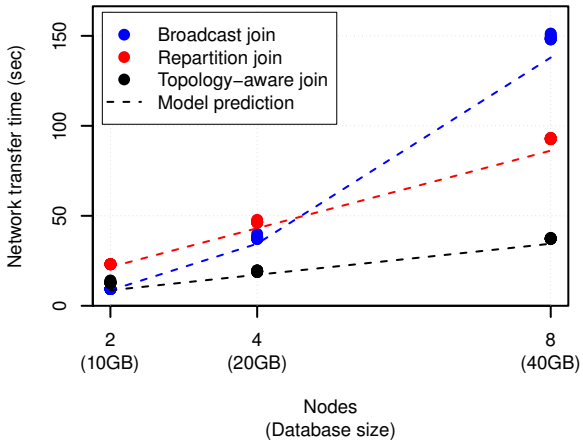


Figure 8: The topology-aware join is 2.5× faster than repartitioning at all scales because it transmits less data over the slow link.

System. We implemented and deployed the topology-aware strategy on a tree-based network with 2, 4 and 8 compute nodes, split into two groups. The groups are connected via a slow 1Gbps link. Nodes within the group use fast 40Gbps links. For this experiment, every node stores 1 GB of R tuples and 4 GB of S tuples. Figure 8 shows the network time for the topology-aware algorithm that was described above with the standard repartition join and broadcast join algorithms in prior literature. The predicted cost from the model for each algorithm is shown as a dashed line. We observe that broadcast join becomes prohibitively expensive at scale as it needs to transmit the R tuples $p-1$ times to every other node. The repartition join time grows linearly with the data volume, but is suboptimal because it transmits tuples from the larger S table across the slow link. The topology-aware join only hashes S within each group to avoid the slow link and replicates the smaller R table to the both groups. Because the topology-aware join transmits less data over the slow link, the topology-aware join is 2.5× faster than repartitioning at all scales.

6. RELATED WORK

Relationship to literature on parallel models. The fundamental difference with other parallel models (e.g., BSP [34], MPC [4], LogP [11]) is that prior models view the network as a star topology, where each link and each node have exactly the same cost functions. In this sense, our model

can be viewed as a generalization, where both network and compute heterogeneity is taken into account.

Relationship to literature on topology-aware communication in HPC. The high-performance computing community has long considered the role of topology in communication, and multiple works have considered how common operations, such as MPI collectives, can be optimized for a given network topology [6, 14, 33]. In broad strokes, the main distinctions between the data processing algorithms we focus in this proposal and the general communication patterns that the HPC community considers are as follows:

1. Most operations during data processing are performed between two pairs of nodes and are scheduled in an ad hoc manner, in contrast with the collective nature of HPC operations;

2. The communication patterns in data processing operations are data-dependent and cannot be reliably known a priori, in contrast with the fixed nature of communication groups in HPC applications;

3. The data processing algorithm can tighten or relax aspects of communication like message priorities and message ordering guarantees at runtime on a per-query basis, in contrast with the more consistent runtime behavior of HPC applications;

4. In data analysis the communication pattern critically depends on the initial data distribution, in contrast with the simpler problem of mapping processes to compute units in the HPC literature;

5. Data processing algorithms can tolerate errors and offer resource elasticity, whereas fault tolerance and elasticity are much harder to achieve in general, given the diversity of HPC applications.

Relationship to literature on parallel DBMSs. Data management research has extensively studied ways to optimize aggregation, sorting and joins in parallel systems.

Related to aggregation, Larson [20] proposed to use partial preaggregation to reduce the input size. Shatdal and Naughton [31] compared the repartitioning and two-phase parallel aggregation algorithms and proposed adaptive aggregation algorithms. Neither has considered the opportunity to reduce the transmitted data volume by computing partial aggregates along network paths. Madden et al. [24] proposed a tiny aggregation service which does in network aggregation in sensor networks; Culhane et al. [10] propose LOOM, which builds an aggregation tree with fixed fan-in for all-to-one aggregations. These systems lack any theoretical guarantees.

Related to join processing, there is a long line of research focusing on parallel joins under skew, starting from the work of DeWitt et al. [12] and Wolf et al. [35], to more recent work on how to mitigate skew by Rödiger et al. [29], and Li et al. [21]. A different line of research has focused on designing join algorithms with theoretical guarantees [18, 19, 17, 15], both for binary and multi-way joins. To best of our knowledge, the above works do not take the topology into account when designing the optimal join algorithm.

7. OPEN QUESTIONS

Modeling node types. Our basic model distinguishes only two types of nodes: routing and compute nodes. One can extend the model with node types of different capabilities. For instance, we can define node types that can route, and also perform restricted forms of computational tasks (for example, aggregation, filtering, or comparisons).

Suitable cost functions. So far we have considered a cost function that estimates the communication cost based on the available network bandwidth. However, in many network settings it will be necessary to model other aspects of the topology, such as communication latency, compute and memory limits, path congestion, or network interference from other processes running in the background.

The general aggregation problem at large scale. Another interesting direction is extending the aggregation problem to the case where the partitioning scheme of the output is not specified by the database system, but by the aggregation algorithm. An important milestone towards answering this question is whether the optimization procedure can be performed with limited communication among nodes.

Optimal topologies for common processing tasks. An intriguing question is what is the optimal topology for common data processing tasks. This can answer what is the most effective way an infrastructure provider can invest a limited budget to accelerate data processing tasks.

8. ACKNOWLEDGMENTS

This research was supported in part by National Science Foundation grants SHF-1816577, CRII-1850348, III-1910014, TRIPODS CCF-1934915, CCF-1815145, CAREER-1453472 and a gift from Oracle Corp.

9. REFERENCES

[1] A. Andreyev. Introducing data center fabric, the next-generation facebook data center network. https://code.facebook.com/posts/360346274145943/.

[2] N. Bansal, Z. Friggstad, et al. A logarithmic approximation for unsplittable flow on line graphs. *ACM Trans. Algorithms*, 10(1), 2014.

[3] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS '96*, pages 184–193, 1996.

[4] P. Beame, P. Koutris, and D. Suciu. Communication Steps for Parallel Query Processing. In *PODS*, 2013.

[5] P. Beame, P. Koutris, and D. Suciu. Skew in Parallel Query Processing. In *PODS*, 2014.

[6] A. Bhatele, N. Jain, W. D. Gropp, and L. V. Kale. Avoiding hot-spots on two-level direct networks. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 76:1–76:11, New York, NY, USA, 2011. ACM.

[7] C. Binnig, A. Crotty, A. Galakatos, et al. The end of slow networks: It's time for a redesign. *Proc. VLDB Endow.*, 9(7):528–539, Mar. 2016.

[8] C. Chekuri, S. Khanna, and F. B. Shepherd. Edge-disjoint paths in planar graphs with constant congestion. *SIAM J. Comput.*, 39(1):281–301, 2009.

[9] Cole, Richard. Parallel Merge Sort. *SIAM Journal on Computing*, 17(4), 1988.

[10] W. Culhane, K. Kogan, C. Jayalath, and P. Eugster. Optimal communication structures for big data aggregation. In *INFOCOM*, 2015.

[11] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *PPOPP*, 1993.

[12] D. J. DeWitt, J. F. Naughton, D. A. Schneider, and S. Seshadri. Practical skew handling in parallel joins. In *VLDB*, pages 27–40, 1992.

[13] M. T. Goodrich. Communication-Efficient Parallel Sorting. *SIAM Journal on Computing*, 29(2), 1999.

[14] T. Hoefler and M. Snir. Generic topology mapping strategies for large-scale parallel architectures. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 75–84, New York, NY, USA, 2011. ACM.

[15] X. Hu, Y. Tao, and K. Yi. Output-optimal Parallel Algorithms for Similarity Joins. In *PODS*, 2017.

[16] Intel's new mesh architecture: The 'superhighway' of the data center. https://itpeernetwork.intel.com/intel-mesh-architecture-data-center/.

[17] B. Ketsman and D. Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *PODS*, pages 417–428, 2017.

[18] P. Koutris, P. Beame, and D. Suciu. Worst-Case Optimal Algorithms for Parallel Query Processing. In *ICDT*, 2016.

[19] P. Koutris and D. Suciu. A guide to formal analysis of join processing in massively parallel systems. *SIGMOD Record*, 45(4):18–27, 2016.

[20] P. Larson. Data reduction by partial preaggregation. In *ICDE*, pages 706–715, 2002.

[21] R. Li, M. Riedewald, and X. Deng. Submodularity of distributed join computation. In *SIGMOD 2018*, pages 1237–1252, 2018.

[22] F. Liu, A. Salmasi, S. Blanas, and A. Sidiropoulos. Chasing similarity: Distribution-aware aggregation scheduling. *PVLDB*, 12(3):292–306, 2018.

[23] F. Liu, L. Yin, and S. Blanas. Design and evaluation of an RDMA-aware data shuffling operator for parallel database systems. In *EuroSys*, 2017.

[24] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. In *OSDI 2002*, 2002.

[25] H. Ngo, C. Ré, and A. Rudra. Skew Strikes Back: New Developments in the Theory of Join Algorithms. *SIGMOD Record*, 42(4), 2014.

[26] S. A. Plotkin, S. Rao, and W. D. Smith. Shallow excluded minors and improved graph decompositions. In *SODA*, volume 90, pages 462–470, 1994.

[27] H. Racke. Minimizing congestion in general networks. In *IEEE FOCS*, pages 43–52, 2002.

[28] N. Robertson and P. D. Seymour. Graph minors XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

[29] W. Rödiger, S. Idicula, et al. Flow-join: Adaptive skew handling for distributed joins over high-speed networks. In *ICDE*, 2016.

[30] W. Rödiger, T. Mühlbauer, et al. Locality-sensitive operators for parallel main-memory database clusters. In *ICDE 2014*, pages 592–603, 2014.

[31] A. Shatdal and J. F. Naughton. Adaptive parallel aggregation algorithms. SIGMOD '95, pages 104–114, New York, NY, USA, 1995. ACM.

[32] H. Shi and J. Schaeffer. Parallel Sorting by Regular Sampling. *Journal of Parallel and Distributed Computing*, 14(4), 1992.

[33] J. L. Träff. Implementing the MPI process topology mechanism. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing, Baltimore, Maryland, USA, November 16-22, 2002, CD-ROM*, pages 40:1–40:14, 2002.

[34] L. G. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, August 1990.

[35] J. L. Wolf, P. S. Yu, J. Turek, and D. M. Dias. A parallel hash join algorithm for managing data skew. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1355–1371, Dec. 1993.