SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

May 2022, Vol. 65 152101:1–152101:19
https://doi.org/10.1007/s11432-020-2996-2

Efficient distributed algorithms for holistic aggregation functions on random regular graphs

Lin JIA[1], Qiang-Sheng HUA[1*], Haoqiang FAN[2], Qiuping WANG[1] & Hai JIN[1]

[1]*National Engineering Research Center for Big Data Technology and System/Services Computing Technology and System Lab/Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074*, China;*
[2]*Institute for Interdisciplinary Information Science, Tsinghua University, Beijing* 100084*, China*

Received 13 March 2020/Revised 18 May 2020/Accepted 17 July 2020/Published online 27 May 2021

Abstract In this paper, we propose efficient distributed algorithms for three holistic aggregation functions on random regular graphs that are good candidates for network topology in next-generation data centers. The three holistic aggregation functions include SELECTION (select the k-th largest or smallest element), DISTINCT (query the count of distinct elements), MODE (query the most frequent element). We design three basic techniques — Pre-order Network Partition, Pairwise-independent Random Walk, and Random Permutation Delivery, and devise the algorithms based on the techniques. The round complexity of the distributed SELECTION is $\Theta(\log N)$ which meets the lower bound where N is the number of nodes and each node holds a numeric element. The round complexity of the distributed DISTINCT and MODE algorithms are $O(\log^3 N/\log\log N)$ and $O(\log^2 N \log\log N)$ respectively. All of our results break the lower bounds obtained on general graphs and our distributed algorithms are all based on the $\mathcal{CONGEST}$ model, which restricts each node to send only $O(\log N)$ bits on each edge in one round under synchronous communications.

Keywords distributed algorithms, holistic aggregation function, random regular graph, $\mathcal{CONGEST}$ model, communication complexity, round complexity

Citation Jia L, Hua Q-S, Fan H Q, et al. Efficient distributed algorithms for holistic aggregation functions on random regular graphs. Sci China Inf Sci, 2022, 65(5): 152101, https://doi.org/10.1007/s11432-020-2996-2

1 Introduction

Aggregation functions compute statistics out of data. An aggregation function f is categorized into non-holistic and holistic according to whether it complies the decomposability, $f(A, B) = f(f(A), f(B))$. Non-holistic aggregation functions include MIN, MAX, MEAN and SUM that are decomposable. Holistic aggregation functions, for example, SELECTION (select the k-th largest or smallest element), DISTINCT (query the count of distinct elements), MODE (query the most frequent element), which are not decomposable, are proved to be harder than non-holistic aggregation functions [1]. While MIN and MAX, which can be regarded as two special cases of SELECTION, are decomposable, the general SELECTION cannot be directly solved via combining the corresponding k-th largest (or smallest) elements from its subsets. Researchers have investigated non-holistic aggregation functions in areas like wireless sensor networks [2, 3], peer-to-peer networks [4], and smart grid [5]. On the other hand, holistic aggregation functions can help detect system anomalies. For example, the system can signal the administrator potential failures if it observes the CPU overhead of 10% percentile exceeds a threshold via the SELECTION function.

We focus on designing distributed algorithms for computing aggregation functions. Specifically, we compute aggregation functions in a network consisting of N nodes with each node possessing one $(\log N)$-bit numeric element and design our algorithms under the $\mathcal{CONGEST}$ model [6] where each node in the network can only send one $O(\log N)$-bit message to each neighbor in every synchronous round. Non-holistic aggregation functions can be efficiently solved without congestion because intermediate results

* Corresponding author (email: qshua@hust.edu.cn)

can be merged along the paths. However, holistic aggregation functions are harder to solve in that we need to collect all raw elements in one node in order to get exact answers. Naively gathering all elements in one node needs $O(N)$ rounds, where we construct a breadth-first-search (BFS) spanning tree out of the network, and each node sends its element to its parent along the spanning tree. Upon one node receiving an element from its children, it transfers the element to its parent until the root node receives all of the elements. Under the $\mathcal{CONGEST}$ model, since one node can at most send $O(\log N)$ bits ($O(1)$ elements) in each round, the number of rounds to finish the algorithm is bounded by the total number of elements sent. Thus, in distributed systems, gathering all raw elements at one node costs $O(N)$ synchronous rounds or violates the $\mathcal{CONGEST}$ model.

We explore designing efficient distributed algorithms for three holistic aggregation functions, SELECTION, DISTINCT and MODE under the $\mathcal{CONGEST}$ model. From state-of-the-art results for holistic aggregation functions on general graphs under the $\mathcal{CONGEST}$ model, for SELECTION, Ref. [1] proposed a distributed deterministic algorithm running in $O(D\log_D^2 N)$ rounds and a distributed randomized algorithm completing in $O(D\log_D N)$ rounds, and they proved the lower bound for distributed SELECTION is $\Omega(D\log_D N)$ on general graphs. For DISTINCT and MODE, Ref. [7] proved that the lower bound of exact computation on general graphs is $\Omega(N/\log N + D)$. For DISTINCT, a randomized (ϵ, δ)-approximation algorithm runs in $O(D + \frac{\log(1/\delta)}{\epsilon^2})$ rounds where $0 < \epsilon,\ \delta < 1$ and $\Pr\{|\hat{F_0} - F_0| \leqslant \epsilon F_0\} \geqslant 1 - \delta$ (F_0 is the number of distinct elements and $\hat{F_0}$ is the estimated value of F_0) [7]. For MODE, Ref. [8] proposed a randomized algorithm of $O(D + F_2/m_1^2 \cdot \log k)$ rounds where F_2 refers to the second moment of the elements in the input set, m_1 is the number of the occurrences of the most frequent element, and k is the number of distinct elements. To obtain a high accuracy for DISTINCT, ϵ could be set to N^{-c} where c is a constant, thus the upper bound of this algorithm is $O(N^{2c})$ rounds with δ being a constant. By applying the worst scenario to the MODE where $k = N$, the MODE finishes in $O(N\log N + D)$ rounds.

Yet the upper bounds for both DISTINCT and MODE algorithms are still quite high (polynomial round complexity). Are there more efficient (in polylog round complexity and even break the lower bounds on general graphs) and accurate (with small error and even no error) algorithms for these holistic aggregation functions? In this paper, by utilizing random regular graphs (RRGs), we show the answer is yes.

RRGs, which are good expanders and have high resilience [9], are good candidates for the next-generation network topology of ever-growing data centers [10, 11]. There are reasons why we choose RRGs rather than other topologies investigated in [12, 13]. First, RRGs are easier to scale compared to network topologies like the fat tree [12] and the hypercubes [13]. In particular, the degree of the node (the number of direct links one node handles) is universally bounded by a constant, which means we can expand the network by merely adding nodes and switches of the same make. Second, in [10], with the problem of wiring RRGs settled, the lack of regular structure brings to RRGs the capability of retaining characteristics in incremental expansions. Third, recent measurements in [14–16] show that expander-based networks such as Jellyfish (RRGs) [10] and Xpanders [17,18] outperform other data center topologies regarding high-performance metrics such as worst-case throughput and cost. We note that Xpanders could also benefit from our techniques in practice since they bear similar properties, e.g., the number of edges of each node is bounded by a constant, as RRGs.

In order to design algorithms for holistic aggregation functions, we make use of two properties of RRGs concluded below:

(1) The diameters of RRGs are $O(\log N)$, which means the messages can be delivered efficiently between any two nodes in the network.

(2) The degree of each node of RRGs is the same constant, showing symmetry which facilitates communications.

These properties seemly facilitate the design of efficient distributed algorithms, and we can obtain an $O(N/d)$ round-complexity distributed algorithm for any aggregation function where d is the degree of the RRG using the above properties. The procedure is the same as on general graphs. Each node sends its element to the root along a constructed BFS tree. Notice on RRGs each node connects d neighbors, which means the root can always receive d elements from its children each round. This confirms a $\max(O(N/d, O(D)))$ algorithm for gathering all elements to the root. Even though it is possible to merge duplicate elements to speed up functions like DISTINCT, yet designing polylog-round complexity distributed algorithms for holistic aggregation functions under $\mathcal{CONGEST}$ model on RRGs is still non-trivial.

Table 1 Results on general graphs and random regular graphs

Upper and lower bounds on general graphs		
Functions	Upper bounds	Lower bounds
SELECTION	$O(D\log_D N)$ [1]	$\Omega(D\log_D N)$ [1]
DISTINCT	$O(N^{2c})$ [7]*	$\Omega(N/\log N + D)$ [7]
MODE	$O(N\log N + D)$ [8]	$\Omega(N/\log N + D)$ [7]
All aggregation functions	$O(N)$ (trivial method)	– (trivial)
Results on RRGs (random regular graphs)		
Functions	Existing algorithms	Our results
SELECTION	$O(\log^2 N/\log\log N)$ [1]	$\Theta(\log N)$
DISTINCT	$O(N^{2c})$ [7]*	$O(\log^3 N/\log\log N)$
MODE	$O(N\log N)$ [8]	$O(\log^2 N\log\log N)$
All aggregation functions	$O(N/d)$ (trivial method)	–

∗ Obtained by guaranteeing high probability $1 - 1/N^c$ of accuracy where c is a constant $c > 0$.

The crux to design low round complexity distributed algorithms for holistic aggregation functions is to increase parallelism without introducing heavy congestion owing to the increased communication. Specifically, we manage to divide the problem into small pieces and solve them individually and design techniques that coordinate communications to avoid congestion. To this end, we design three techniques as the building blocks, namely Pre-order Network Partition, Pairwise-independent Random Walk, and Random Permutation Delivery on RRGs, and devise our algorithms based on the techniques.

At the end of this section, we give a comparison between our results and previous results in Table 1 to give readers a more perceptual intuition about the efficiency of our algorithms. By applying state-of-the-art solutions to RRGs, SELECTION can be computed in $O(\log^2 N/\log\log N)$ rounds, and the randomized algorithms for DISTINCT and MODE can finish in $O(N^{2c})$ and $O(N\log N)$ rounds, respectively. Note that the results for DISTINCT and MODE are both exact while the former results are approximate. Our algorithm for SELECTION is quadratically faster than the previous one, and our algorithms for DISTINCT and MODE are exponentially faster. All the results on RRGs break the lower bound obtained on general graphs.

The rest of the paper proceeds as follows. Section 2 introduces the system model and problem definitions. Section 3 describes and analyzes three basic techniques. Based on the techniques, Section 4 proposes the distributed algorithms for SELECTION, DISTINCT, and MODE. Analysis regarding correctness, round complexity, and message complexity of the algorithms will be presented in Sections 5–7, respectively. Section 8 concludes this work and discusses future work.

2 System model and problem definitions

2.1 System model

The network is an undirected connected graph $G = (V, E)$, where V is the set of nodes, and E is the set of edges connecting nodes in the network. The diameter of G, which is the length of the longest shortest path of G, is denoted by D. Each node represents a processor, and the edges represent point-to-point connections under $\mathcal{CONGEST}$ model, where computations are performed in synchronized rounds. A node can only directly exchange information limited to $O(\log N)(N = |V|)$ bits with its neighbors. Nodes can perform local computations and decide what to exchange in the next round. For all algorithms, only one node is required to report the correct answer. Nevertheless, a broadcast suffices for all nodes reporting the answer. Each node's local computation time is neglected, and we mainly focus on the round complexity metric which indicates the maximum number of synchronous rounds needed to complete a distributed algorithm for any node. We also analyze the message complexity, i.e., the total number of sent messages of all nodes in our algorithms.

2.2 Problem definitions

Assuming each node $v \in V$ holds one numeric element of $O(\log N)$ bits, forming a set S. We give the definitions of our problems distributed SELECTION, distributed DISTINCT and distributed MODE on set S below.

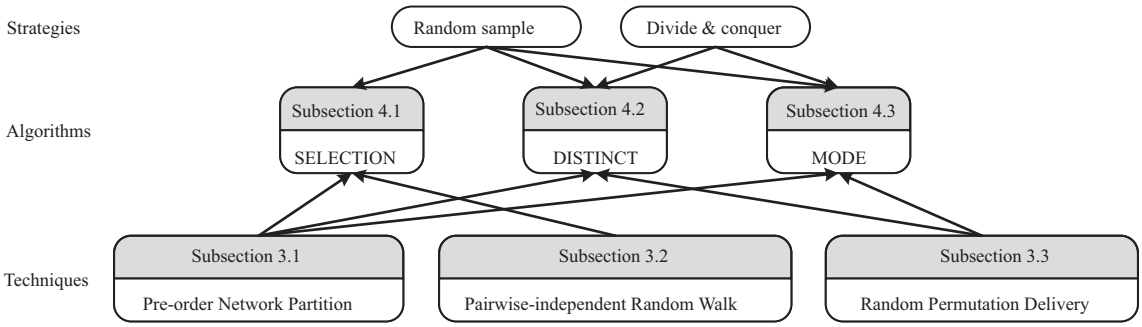

Figure 1 Relationships between our ideas, techniques and algorithms.

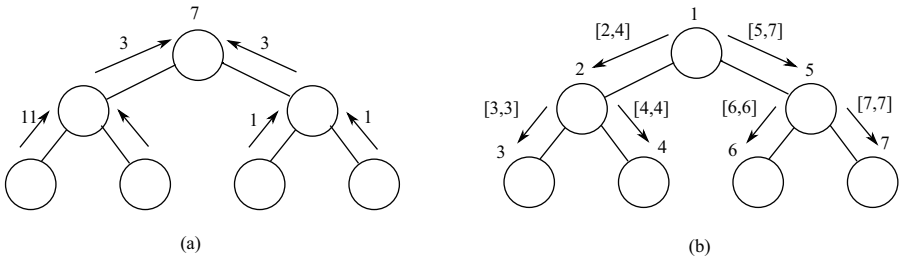

Figure 2 Computing the order by broadcast and convergecast. The sizes of subtrees are computed by convergecast. Then labels are assigned in a top-down manner.

Definition 1 (Distributed SELECTION)**.** For a distributed stored set S and an element k, the distributed SELECTION problem is to find the k-th largest (smallest) element in S under the $\mathcal{CONGEST}$ model.

Definition 2 (Distributed DISTINCT)**.** For a distributed stored set S, the distributed DISTINCT problem is to find the count of distinct elements in S under the $\mathcal{CONGEST}$ model.

Definition 3 (Distributed MODE)**.** For a distributed stored set $S = \{s_1, \ldots, s_N\}$, we denote m_i as the number of occurrences of s_i in S. The distributed MODE problem is to find the element s_i with the maximum m_i where $1 \leqslant i \leqslant N$ under the $\mathcal{CONGEST}$ model.

3 Basic techniques

In this section, we will give a detailed illustration and analysis of the techniques. Figure 1 outlines the relationship between the techniques and three proposed algorithms. In the SELECTION algorithm, we use Pre-order Network Partition and Pairwise-independent Random Walk. As for the algorithms for DISTINCT and MODE, they both benefit from Pre-order Network Partition and Random Permutation Delivery. Note that we assume the network is based on RRGs in the techniques. We place some of the proofs in Appendix.

3.1 Pre-order Network Partition

Pre-order Network Partition divides the network into groups of nodes where sub-problems can be solved individually. It enables high parallelism for three holistic aggregation functions. We denote k as the number of groups the network is partitioned into. Pre-order Network Partition conducts in two phases — label and group.

In the label phase, firstly, an $O(D)$ rounds distributed breadth first search (BFS) generates a spanning tree out of the network. Next, we assign labels to each node in a top-down manner. We assign a label range $(1, N)$ to the root, and then conduct a second BFS. Each node receives a label range from its parent. In each synchronous round, upon receiving the range, the node labels itself the left-most number of the range, and sends remaining labels to its children according to their sizes. After a second $O(D)$ rounds, all nodes get their labels. We represent this procedure in Figure 2.

In the group phase, we mark each node its group number according to its label and the total number of groups. We note here that the supported algorithm decides the number of groups. We simply designate

node i to group $\lfloor \frac{k \cdot i}{N} \rfloor$ to balance group sizes. Each node knows the labels of the nodes in the same group after the group phase.

In the following, we will prove that intra-group message delivery is efficient after partitioning. Lemma 1 shows that each edge is only shared by no more than two groups and we prove it in Appendix A.1.

Lemma 1. Pre-order Network Partition divides the network into groups with every edge of the network being shared by no more than two groups.

Lemma 1 indicates that each edge of the network is shared by no more than two groups, and thus efficient intra-group communications are guaranteed. As Pre-order Network Partition only involves BFS which finishes in $O(D)$ rounds. Each node transmits $O(1)$ messages in the whole process, indicating that the message complexity is $O(N)$.

Theorem 1. Each node of the network can be pre-order labeled and partitioned into groups in $O(\log N)$ rounds with the message complexity of $O(N)$.

Proof. Pre-order Network Partition involves two phases. The label phase consists of two BFS which finishes in $O(D)$ rounds. The label phase finishes when the second BFS in the label phase completes. Considering the diameter D of RRGs is $O(\log N)$, the round complexity of Pre-order Network Partition is $O(\log N)$. The BFS conducted in the label phase only requires each node to send $O(1)$ messages in total. This suggests the message complexity of Pre-order Network Partition is $O(N)$.

3.2 Pairwise-independent Random Walk

We propose Pairwise-independent Random Walk to mix elements in the network, and thus enable uniform sampling in the SELECTION algorithm. Pairwise-independent Random Walk highly inherits from the standard random walk. In the following, we denote the degree of RRGs as d.

Assume adjacency matrix $G_{(i,j)}$ presents the given graph $G = (V, E)$ where each entry encodes whether two nodes i and j are connected. Algorithm 1 outlines this procedure. Node i has a probability of $1/d$ to try to swap its element with one of its neighbors. Only when two nodes both choose each other as the target they exchange their elements. This indicates that for one node, the probability of swapping its element with one of its neighbors is $1/d^2$, which suffices the pairwise-independent property as we prove in Lemma A1.

Algorithm 1 Pairwise-independent Random Walk

```
repeat O(log N/(1 − λ2)) times do
for each node u in the network do
u randomly chooses a neighbor v;
if v also chooses u do
u and v swap their numbers;
end if
end for
end repeat
```

The transition matrix of our random walk is

$$M = (G/d)/d + (1 - 1/d)I,$$

where I is an identity matrix.

Then we analyze the mixing time based on this random walk that focuses on the moving of every single element and each pair of elements instead of analyzing a random walk of all elements which has a state space of $n!$. The rationale behind is that after the mixing time is deducted, the positions of all elements and all pairs of elements will be uniformly distributed in the network, i.e., the corresponding random walk converges to a stationary state of uniform distribution. Specifically, we deduce the number of rounds needed to reach the uniform distribution over all pairs $(i, j) \in V \times V$ which requires the following.

(1) The final position of each element is uniformly distributed over all nodes. (2) The final position of any two elements is uniformly distributed over all $N(N-1)$ node pairs.

The final position of each element must be uniformly distributed over all nodes because M is symmetric and stochastic which indicates that the stationary distribution is a uniform distribution.

We denote the transition matrix of a pair of nodes as M_2 which is thus an $N^2 \times N^2$ matrix, and each entry of M_2, $(M_2)_{(p,q),(r,s)}$ is the probability of two elements located at p and q going to r and s after one

step respectively. We compare M_2 with the transition matrix $\tilde{M}_2$ when the elements are truly pairwise independently moving. We can see $\tilde{M}_2$ is obtained by taking the Kronecker product of M:

$$(\tilde{M}_2)_{(p,q),(r,s)} = M_{(p,r)} \otimes M_{(q,s)}.$$

For a random walk with uniform distribution as its stationary state, the mixing time (the rounds needed to converge to the stationary state) depends on the eigenvalue gap (the difference between the largest and the second largest eigenvalue). If it is a constant, the distribution of the positions of the number converges towards uniform distribution exponentially faster [19]. We denote λ_2 as the second largest eigenvalue of M. The mixing time can be concluded in Theorem 2 with the proof in Appendix A.2.

Theorem 2. After $t = O(\frac{\log N}{(1-\lambda_2)/d})$ steps of the Pairwise-independent Random Walk, the statistical distance between the distribution of the position of elements D_t and the uniform distribution over all pairs $\{(i,j) \in V \times V | i \neq j\}$ is

$$\frac{1}{2} \sum_{i \neq j} \left| (D_t)_{(i,j)} - \frac{1}{N(N-1)} \right| = O(\epsilon),$$

where ϵ is a positive value depending on t $(0 < \epsilon < 1)$.

Theorem 3. After $t = O(\frac{d \log N}{(1-\lambda_2)})$ steps of the Pairwise-independent Random Walk, the message complexity is $O(\frac{dN \log N}{(1-\lambda_2)})$. The message complexity to finish Pairwise-independent Random Walk is $O(\frac{dN \log N}{(1-\lambda_2)})$.

Proof. In each round, every node randomly picks one neighbor and sends one message to it. The algorithm completes in $O(\frac{d \log N}{(1-\lambda_2)})$ rounds. So the total number of messages is $O(\frac{dN \log N}{(1-\lambda_2)})$.

It has been proved that the eigenvalue gap of a standard random walk on RRGs is $\Omega(1)$ [9], and the degree of RRGs is also a constant. Consequently, the eigenvalue gap of our random walk is a constant. We restate the round complexity and message complexity of the Pairwise-independent Random Walk on RRGs as below.

Corollary 1. After $t = O(\log N)$ steps of the Pairwise-independent Random Walk, the statistical distance between the distribution of the position of an element D_t and the uniform distribution over all pairs $\{(i,j) \in V \times V | i \neq j\}$ is

$$\frac{1}{2} \sum_{i \neq j} \left| (D_t)_{(i,j)} - \frac{1}{N(N-1)} \right| = O(N^{-\alpha}),$$

where α is a constant, and the message complexity is $O(N \log N)$.

We conclude the Pairwise-independent Random Walk with a discussion about the gossip protocol [20]. Our technique achieves the pairwise-independency via a procedure of element swapping, which requires two adjacent nodes to choose each other in the same synchronous round. However, this swap procedure cannot be fulfilled in the gossip protocol where such an agreement cannot be established. As a consequence, gossip cannot be directly applied to our scenario.

3.3 Random Permutation Delivery

Random Permutation Delivery sends elements to different groups so that problems can be divided and conquered individually. The problem is how to efficiently coordinate the delivery of data flow. To understand the problem of how to deliver the packets, we observe two principles in our algorithms.

(1) The original group is divided into new groups. Each element knows which new group it should be sent to.

(2) It does not matter which specific node in the new group receives the element. Moreover, we want to randomize the destination to facilitate sampling operations that help reduce computations.

Based on the observations, we formulate the task of the Random Permutation Delivery as three stages and give an illustration in Figure 3.

(1) The first stage is to generate a random permutation. Each packet independently chooses an intermediate position randomly among all nodes and goes to it.

(2) In the second stage, each packet goes to a randomly chosen node in its destination interval which is also a group in our pre-order partition.

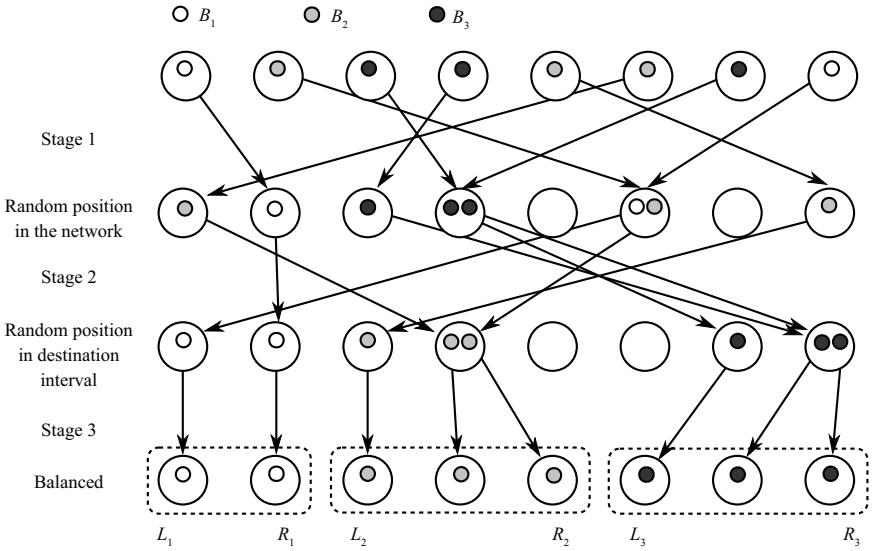

Figure 3 Three stages delivery. Each packet belongs to a group B_i and needs to be sent to a node in $[L_i, R_i]$. In the first stage, it is sent to a random node in the network. In the second stage, it is sent to a random node in the destination group. In the third stage, the packets in a group are re-allocated so that each destination receives exactly one packet.

(3) The third stage is a balancing stage to ensure each node finally receives exactly one packet.

To achieve the delivery, we borrow a randomized online algorithm which exploits the shortest paths in the network to achieve packet delivery in $O(C + D' + \log N)$ time from [21], where C is the congestion of edges defined as the maximum number of packets transferred along one edge given the paths are the set of all shortest paths, and D' is the maximum length of the shortest paths of all pairs of nodes.

Theorem 4.18 in [22] proved that the system congestion C_W, the maximum number of paths from the shortest paths system W (a set of N^2 shortest paths through the graph) that pass through the same node, is $O(N \log^2 N)$ in RRGs. We reformulate the result in [22] as Theorem 4.

Theorem 4. Assume $O(N)$ independent packets need to be delivered and the i-th packet goes from u_i to v_i. Let P be the set of the shortest paths that connect u_i and v_i. Then the congestion C of P is $O(C_W/N + \log N)$ with high probability if the random variables (u_i, v_i) are independent for each i and

$$\sum_{i=1} \Pr[u_i = x \wedge v_i = y] = O(N^{-1})$$

for any $x, y \in V$.

Theorem 4 implies that routing randomly delivered packets need $O(\log^2 N)$ rounds in RRGs, and we conclude Corollary 2.

Corollary 2. On RRGs, the round complexity of the Random Permutation Delivery is $O(\log^2 N)$ with high probability, and the message complexity is $O(N \log^2 N)$.

Proof. As proved, the number of rounds needed to finish the Random Permutation Delivery is $O(\log^2 N)$. The number of packets transmitted in one round is no more than the number of edges in the network, which means $O(dN)$ messages transmitted in total in one round. So the message complexity is $O(dN \log^2 N)$.

Notice that on RRGs, d is a constant, so the message complexity of the Random Permutation Delivery is $O(N \log^2 N)$.

4 Algorithms

In this section, we introduce the procedures of three algorithms. To give readers a clear picture of how the algorithms work, instead of from each nodes' perspective, we describe our algorithms from the network's point of view, which consists of steps that can be translated directly into a set of techniques; those techniques are either widely used, e.g., BFS, convergecast and broadcast, or proposed by us (see Section 3). When analyzing their round complexity, we explicitly detail how each step is translated into the techniques.

4.1 Distributed SELECTION

In the distributed SELECTION problem, we want to find the k-th largest (smallest) element in a set S stored in a distributed network. We base our selection algorithm on the quick-selection algorithm. Without loss of generality, we assume the elements in set S are distinct. The quick-selection algorithm proceeds as follows. A candidate set I is initialized as S. In each iteration an element m is randomly chosen and its rank $r_S(m) = |\{a \in S | a < m\}|$ is computed. If the rank is greater than k, discard the elements greater than m from I; otherwise, discard the elements less than m from I. The algorithm terminates when $|I| = 1$.

In the procedure above, the candidate set shrinks at about half after each iteration, so the total number of iterations is $O(\log N)$. To reduce the iterations needed, in a distributed network, the crux is to sample more elements simultaneously. Ref. [1] sampled $O(D)$ elements in one iteration and proposed an algorithm of round complexity $O(D \log_D N + D)$. We show there exists a positive constant c such that in each iteration we can sample N^c elements and sort them on RRGs, and thus reduce the number of iterations to $O(1)$.

In our algorithm for distributed SELECTION, we maintain L and H as the current lower and upper bounds of the k-th element, respectively, and update the candidates set $I = \{a \in S | L \leqslant a \leqslant H\}$ in each round. Algorithm 2 frames the procedure of the distributed SELECTION algorithm with a description as follows.

(1) Initialize lower bound L, upper bound H and candidate set I (line 2).

(2) Randomly sample a subset S_{sample} from I (line 4).

(3) Sort S_{sample}, and then shrink the candidate set I by updating L and H and update k (lines 5 and 6).

(4) If I is small enough, sort I and find the k-th element m, return m as the answer. Otherwise, go to Step 2 (lines 7 and 8).

Algorithm 2 SELECTION

1: Input set S, a number k.
2: Set search range $[L, H]$ initially to the minimum and maximum of the input set S, and candidate set $I = S$.
3: **repeat**
4: Randomly sample N^c elements from I as S_{sample}.
5: Sort S_{sample}.
6: $L = (\frac{k_0}{|I|}|S_{\text{sample}}| - O(\sqrt{|S_{\text{sample}}|}\log N))$-th element of S_{sample};
$H = (\frac{k_0}{|I|}|S_{\text{sample}}| + O(\sqrt{|S_{\text{sample}}|}\log N))$-th element of S_{sample};
$k = k - |\{a \in S | a < L\}|$, $I = \{a \in S | L \leqslant a \leqslant H\}$.
7: **until** $|I| \leqslant N^c$.
8: Sort I, return the k-th element as the answer.

In each iteration, we reduce the range of k-th element until the size of I is small enough so we can sort I directly. The problem is how to sort S_{sample} efficiently. Accordingly, we propose a sort routine where we estimate the rank of each element in S_{sample}.

Since we sample N^c elements in each iteration, which also represents N^c nodes, it is possible to use remaining nodes to estimate the ranks. We use all nodes in the network in the sort routine. The sort routine leverages Pre-order Network Partition to divide the network into $O(N^c)$ groups with each group of size $O(N^{1-c})$, and then estimates the rank of each sampled element in the corresponding group. For a good estimation, we would like a uniform distribution of elements in each group. Here we resort to Pairwise-independent Random Walk.

We frame our sort process with Figure 4 and give an illustration with pseudocode in Algorithm 3.

(1) Partition the network into $|S_{\text{sample}}|$ groups (line 2).

(2) Spread sampled elements in its own group (line 3).

(3) Use Pairwise-independent Random Walk to mix all sampled elements (line 4).

(4) Estimate the rank of each sampled element in its own group (line 5).

4.2 Distributed DISTINCT

In the distributed DISTINCT problem, we want to find the number of distinct elements in a set S stored in a distributed network. Applying the quick-sort principle and our techniques to the distributed DISTINCT problem, we propose a deterministic algorithm that efficiently solves distributed DISTINCT.

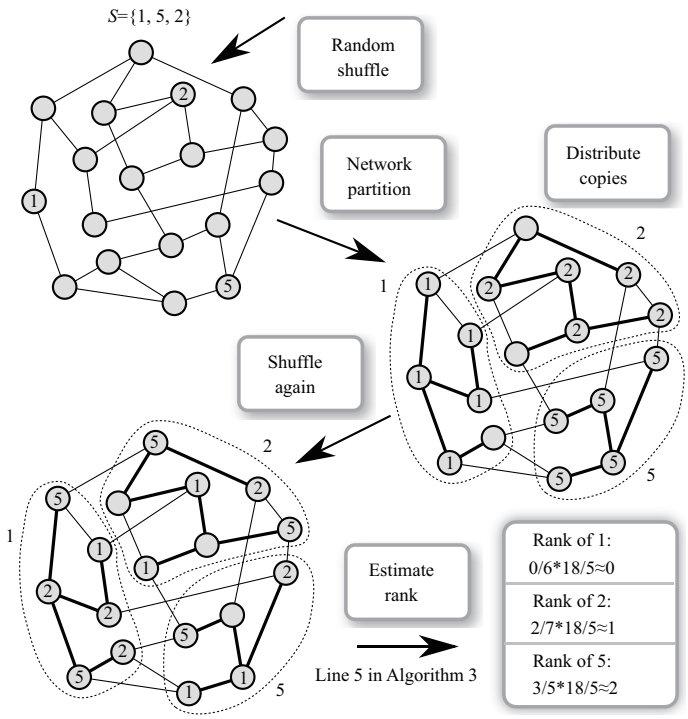

Figure 4 Sorting a sampled subset. The numbers are first shuffled in the network. Then the network is partitioned into 3 groups and each group is designated to a number. Each number distributes 5 copies of itself into the group. The copies are shuffled again, then all the copies are uniformly distributed. Finally, the rank of a number is estimated in the corresponding group.

Algorithm 3 SORT

1: Input sampled set S_{sample}.
2: Partition the network into $|S_{\text{sample}}|$ groups.
3: Spread sampled elements in its own group.
4: Shuffle elements using Pairwise-independent Random Walk.
5: Estimate $r_{S_{\text{sample}}}(s) = (r_g(s)/|g|) \cdot (N/l)$ in groups, where $r_g(s)$ is the rank of s in group g, l is the number of copies distributed to each group.

Specifically, we randomly sample a set of elements from S and divide S into subsets according to sampled elements. Then we apply the same procedure to the subsets recursively. During this process, we discard elements that are equal to sampled ones and count them in the result. Finally, we collect the sub-results from bottom to top.

The three phases of our algorithm are the following, with an example illustrated in Figure 5.

(1) Sample phase (line 3): Randomly pick l elements from S and denote them as $S_{\text{sample}} = \{s_0, s_1, \ldots, s_l, s_{l+1}\}$ where $s_0 = -\infty$ and $s_{l+1} = \infty$.

(2) Divide phase (lines 5 and 6): Broadcast S_{sample} to all nodes, and divide S into $l+1$ subsets with regard to the values of elements. Then each element goes to its sub-group via the Random Permutation Delivery. The elements in subset I_i satisfy $s_{i-1} \leqslant x \leqslant s_i, x \in I_i$.

(3) Combine phase (lines 7–9): Sum up the results gathered from sub-groups with the number of distinct elements in the sampled elements together, and return the sum to the group leader.

We give the pseudocode of our algorithm for distributed DISTINCT in Algorithm 4.

4.3 Distributed MODE

In the distributed MODE problem, we want to find the most frequent element and its frequency in a set S stored in a distributed network. Note that if we just modify the Combine phase of the DISTINCT algorithm to compute a MAX function instead of SUM, we can devise an algorithm to solve distributed MODE in the same number of rounds. We will call this algorithm Simple MODE in the following. However, we can develop a more efficient MODE algorithm by analyzing what really happens when it comes to MODE.

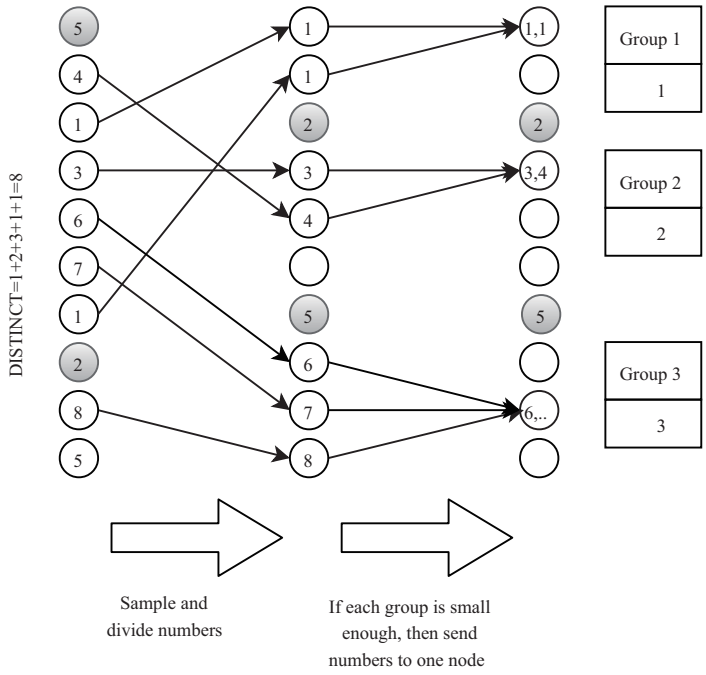

Figure 5 The procedure is shown from left to right. We first sample $S_{\text{sample}} = 2, 5$ out of S and broadcast them in the network (the numbers in the gray nodes). Each node computes the corresponding subset of its element. Discard the numbers in S_{sample} and deliver the remaining numbers into corresponding groups (in this example, we discard 2, 5). If the size of a group is greater than $O(\log N)$, run sample and divide phases recursively, otherwise we send the numbers of a group to one node and compute the DISTINCT locally. The bottom (right) returns the number of distinct numbers to the upper (left) level. The upper level adds this result by the number of distinct elements in S_{sample}. In this example, sub-groups in the middle level need not be divided because they are small enough.

Algorithm 4 DISTINCT

```
Input set S', which is stored in a group of nodes.
if |S'| > O(log N) then
    Sample a set S_sample with size l = O(log N) from S'.
    Broadcast S_sample in the group.
    Divide elements in S into at most l + 1 subgroups according to their ranks in S_sample.
    Compute DISTINCT in each subgroup.
    Sum the results of all subgroups and the number of distinct elements and send it to the group leader.
else
    Send all elements to one node and compute r locally.
end if
Return r.
```

We observe that either of two things will happen when we sample a subset S_{sample} from S and divide S into sub-groups.

(1) MODE m appears many times in the network. Then we have a good chance that S_{sample} contains m. We can find it out by estimating the frequencies of the elements in S_{sample}.

(2) MODE m appears only a few times in the network. In this case, we use a hash function H to break the problem into sub-problems.

We can see that either situation is related to the frequency of MODE, so we can design our algorithm by bounding the frequency of MODE denoted by a value M. Formally, we maintain an upper bound of M. M is decreased recursively until we reach the first case.

Initially, we set M to N. In each iteration, we divide the set S into $l = O(\max\{N/(M \log N), 1\})$ groups $g_1, g_2, \ldots, g_l$ using Pre-order Network Partition. Choose a hash function $H : S \to \{1, 2, \ldots, l\}$. Each element x computes $H(x)$ and moves to the $H(x)$-th group by Random Permutation Delivery. After the delivery, each group is further partitioned into $l_2 = O((N/l)^{c_2})$ blocks where c_2 is a constant predefined. In each group, each block randomly picks one element and computes its frequency f_j in the block. Let f be the maximum frequency among the blocks. According to how large f is, the algorithm proceeds in two directions as introduced previously.

(1) If $f \geqslant \Theta(M/l_2)$, which indicates that MODE m may appear many times, we distribute the elements

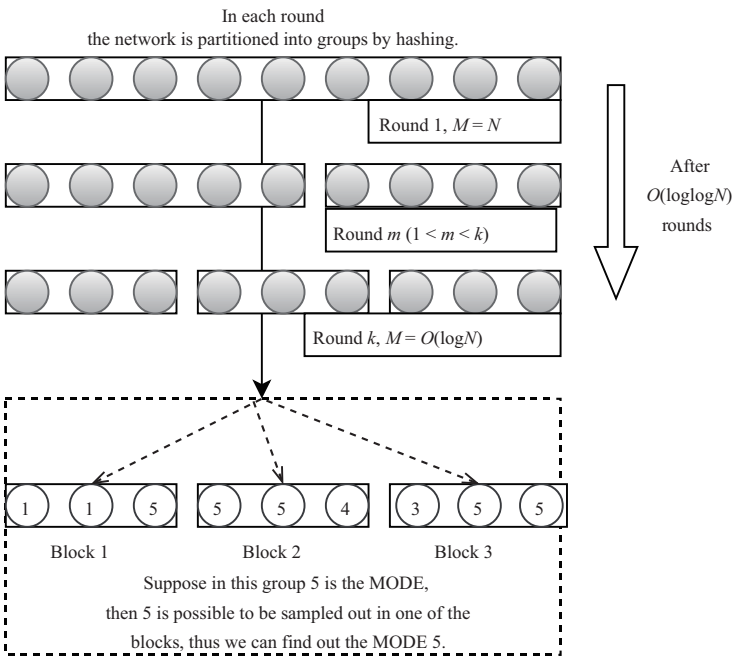

Figure 6 Computing MODE. As the upper bound of M shrinks, the number of groups to be partitioned increases. We terminate this procedure when at some stage the MODE is found out, or M is bounded small enough where we use Simple MODE (the same procedure as DISTINCT) in each group.

with frequencies larger than $\Theta(M/l_2)$ to all nodes in the same group and compute the exact frequencies of them. Report the element with the maximum frequency as MODE.

(2) If $f \leqslant \Theta(M/l_2)$, which indicates that MODE m only appears a few times, thus we need to adjust the upper bound of MODE. In this case, we set $M = O(\max\{fl_2, N/(l \cdot l_2)\})$.

When the upper bound M is small enough, we use simple MODE in each group to compute the exact MODE. At last, we combine our results from the groups. In the first case, the MODE has been computed at some level and the algorithm terminates. In the second case, each group runs the simple MODE algorithm at the bottom of the recursion. MODEs of groups are returned and combined to get the final answer. The pseudocode is outlined in Algorithm 5. We prove that our MODE algorithm is efficient both in round and message complexities in Section 7. We give an example of the second case in Figure 6.

Algorithm 5 MODE

Set $M = N$.
while $M \geqslant O(\log^{O(1)} N)$ **do**
  Set $l = O(\max\{\frac{N}{M \log N}, 1\})$.
  Choose a hash function H with range $[1, l]$. Divide the network into l groups using Pre-order Network Partition.
  Each element x moves to $H(x)$-th group using the Random Permutation Delivery.
  for each group **do**
    Partition the group into $l_2 = O((N/l)^{c_2})$ blocks where c_2 is a positive constant.
    Randomly sample one number and compute its frequency in the block.
  end for
  Let f be the maximum of all computed frequencies.
  if $f \geqslant \Theta(M/l_2)$ **then**
    Compute the exact frequency of those sampled numbers whose frequency exceeds $\Theta(M/l_2)$.
    Return the maximum of exact frequencies and the corresponding element as MODE.
  else
    Set $M = O(\max\{f \cdot l_2, \frac{N \log N}{l \cdot l_2}\})$.
  end if
end while
Divide the network into $l = O(\max\{\frac{N}{M \log N}, 1\})$ groups.
Run the simple MODE algorithm in each group and compute the maximum frequency of returned elements.

5 Analysis of distributed SELECTION

5.1 Correctness

We first give a conclusion that our algorithm for distributed SELECTION can compute an accurate result with high probability.

Lemma 2 shows that in each round we have an accurate estimation of the ranks of sampled elements with high probability.

Lemma 2. For each element s in the sampled subset $S_{\text{sample}} \subset I$, we define $r_{S_{\text{sample}}}(s) = |\{a \in S_{\text{sample}} | a < s\}|$. After each sampled element s distributes $l = O(N/|S_{\text{sample}})$ copies of itself into its group, we can estimate the rank of s in S_{sample} with high probability:

$$r_{S_{\text{sample}}}(s) \approx \frac{r_g(s)}{|g|} \cdot \frac{N}{l},$$

where $r_g(s)$ is the rank of s in its group and $|g|$ is the number of elements in the group of s.

Proof. Let g also denote the set of elements in the group that s belongs to. Let X_i denote whether a copy of $i \in S_{\text{sample}}$ that is smaller than s is shuffled to g. So the estimation of rank of s in S_{sample} is

$$\tilde{r}_{S_{\text{sample}}}(s) = \frac{\sum_i X_i}{|g|} \cdot \frac{N}{l}.$$

Let $X = \sum_i X_i$. After a Pairwise-independent Random Walk, the probability of every copy shuffled to g is $\mathrm{E}[X_i] = \frac{|g|}{N}$, so we have

$$\mathrm{E}[X] = r_{S_{\text{sample}}}(s) \cdot l \cdot |g|/N.$$

The variance of X is

$$\mathrm{Var}[X] = \sum_i \mathrm{Var}[X_i] + \sum_{i \neq j} (\mathrm{E}[X_i X_j] - \mathrm{E}[X_i]\mathrm{E}[X_j]).$$

After the Pairwise-independent Random Walk, we have

$$\mathrm{E}[X_i X_j] - \mathrm{E}[X_i]\mathrm{E}[X_j] = \epsilon + \frac{\binom{|g|}{2}}{\binom{N}{2}} - \left(\frac{|g|}{N}\right)^2 = O(N^{-1}),$$

where ϵ is a negligible quantity corresponding to the imperfectness of the random walk. So

$$\mathrm{Var}[X] = \sum_i \mathrm{Var}[X_i] + O(N) = O(N).$$

Then we prove with high probability that the deviation of $\tilde{r}_{S_{\text{sample}}}(s)$ and $r_{S_{\text{sample}}}(s)$ is bounded by $1/2$. Suppose $r_{S_{\text{sample}}}(s) - \tilde{r}_{S_{\text{sample}}}(s) \geqslant 1/2$, we have

$$X - \mathrm{E}[X] = \Omega\left(\frac{1}{2}(|g| \cdot l)/N\right) = \Omega(N^{1-2c}),$$

where N^c is the number of sampled elements per round.

By Markov inequality, we have

$$\Pr[|X - \mathrm{E}[X]| > \Omega(N^{1-2c})] \leqslant \frac{\mathrm{Var}[X]}{\Omega(N^{2-4c})} = O(N^{4c-1}).$$

When c is small enough, the probability is $N^{4c-1} \times N^c = o(1)$ by union bound. Every computed rank has an additive error smaller than $1/2$ and we can find the exact rank by rounding $r_{S_{\text{sample}}}(s) = \lfloor \tilde{r}_{S_{\text{sample}}}(s) + 0.5 \rfloor$ w.h.p. (with high probability).

Let $k_0 = k - |a \in S | a < L|$. The k-th element in S will be the k_0-th element in I. Lemma 3 proves that right lower bound and upper bound of the k_0-th element of I in each iteration are computed w.h.p.

Lemma 3. In the SELECTION algorithm, with every element sampled with a predefined probability of $p \approx |S_{\text{sample}}|/|I|$, the rank of k_0-th element of I in S_{sample} is bounded by the interval

$$\left[k_0\frac{|S_{\text{sample}}|}{|I|} - O\left(\sqrt{|S_{\text{sample}}|}\log N\right),\quad k_0\frac{|S_{\text{sample}}|}{|I|} + O\left(\sqrt{|S_{\text{sample}}|}\log N\right)\right],$$

w.h.p.

Proof. Let t be the k_0-th element in I. For each $i \in S, i < t$, we associate a random variable X_i indicating whether $i \in S_{\text{sample}}$. So the rank of t in S_{sample} is $\sum_i X_i$. By Hoeffding's inequality, we have

$$\begin{aligned}\Pr\left[|X - \mathrm{E}[X]| \geqslant \Omega\left(\sqrt{|S_{\text{sample}}|}\log N\right)\right] &\leqslant 2\exp\left(-2\frac{(\sqrt{|S_{\text{sample}}|}\log N)^2}{|S_{\text{sample}}|}\right)\\ &< 2\cdot 2^{-2\log^2 N} \leqslant O(N^{-\Omega(1)}).\end{aligned}$$

This implies that the rank of the k_0-th element in S_{sample} is bounded by the interval $[k_0\frac{|S_{\text{sample}}|}{|S|} - O(\sqrt{|S_{\text{sample}}|}\log N), k_0\frac{|S_{\text{sample}}|}{|S|} + O(\sqrt{|S_{\text{sample}}|}\log N)]$ w.h.p.

5.2 Round complexity and message complexity

In the algorithm for distributed SELECTION, the size of candidate set in each iteration is narrowed by $O(\sqrt{|S_{\text{sample}}|}/\log N) = O(N^{c/2}/\log N)$, which means after a constant number of rounds, the range will be small enough (less than $O(N^c)$) as we conclude in Lemma 4.

Lemma 4. The number of iterations needed to compute the k-th element of S is bounded by $O(1)$.

Theorem 5. Our algorithm for distributed SELECTION can compute accurate k-th element of set S in $O(\log N)$ rounds with high probability with message complexity $O(N\log N)$.

Proof. In each iteration, the main work lies in the sort routine and the rank estimation. The sort routine leverages Pre-order Network Partition and Pairwise-independent Random Walk technique, which costs $O(\log N)$ rounds. Estimating the rank in each subgroup also introduces $O(\log N)$ rounds because a convergecast suffices to get all elements that are less (greater) than the estimated one.

Lemma 4 indicates the number of iterations needed to finish the algorithm is $O(1)$. The round complexity of SELECTION is $O(\log N)$.

Pre-order Network Partition introduces $O(N)$ messages in each iteration, and Pairwise-independent Random Walk transmits $O(N\log N)$ messages in total in each iteration. The rank estimation involves a convergecast which is the same as Pre-order Network Partition. As there are $O(1)$ iterations in total, the message complexity of the algorithm for SELECTION is $O(N\log N)$.

6 Analysis of distributed DISTINCT

6.1 Correctness

We prove that our algorithm can compute the correct result by induction.

Theorem 6. Our algorithm for distributed DISTINCT computes correct DISTINCT.

Proof. Suppose we can compute the DISTINCT of all groups with a size less than k accurately. When the size of a group is less than $O(\log N)$, the result is correct because DISTINCT is computed locally in the one leader node.

If the size of the group is less than k, the result must be correct. Otherwise, the group will be divided into $l+1$ groups with a size less than k. Correct results from the sub-groups are sent to the leader and combined with the sampled set, which suffices the correctness of the final result.

6.2 Round complexity and message complexity

We now show how quick our algorithm for distributed DISTINCT can achieve and analyze the message complexity by phases.

Lemma 5. The round complexity of each iteration of our algorithm for DISTINCT is $O(\log^2 N)$.

Proof. Firstly, $O(\log N)$ elements are sampled and broadcast inside a group. This procedure can be implemented using pipelining, which takes $O(D+\log N) = O(\log N)$ rounds. Then Random Permutation Delivery is used to divide the problem and form sub-groups, which is proved to cost $O(\log^2 N)$ rounds. Finally, results are sent to the leader, which takes at most $O(D) = O(\log N)$ rounds.

In summary, each iteration of the algorithm needs $O(\log^2 N)$ rounds to complete.

Lemma 6. The recursion depth of our algorithm for distributed DISTINCT is $O(\log N/\log\log N)$ w.h.p.

Proof. We first prove that the size of a sub-group shrinks at least by $O(\sqrt{\log N})$ on average. Consider a group g with N numbers and g is divided to $g_i, 0 < i < l$ subgroups. Without loss of generality, we assume $N > l = O(\log N)$. Set s to the size of a sub-group and $t = \Omega(N/\sqrt{\log N})$. The probability that s is greater than t is

$$\begin{aligned}\Pr[s \geqslant t] &= \sum_{s \geqslant t}(N-s-1)\frac{\binom{N-s-2}{l-2}}{\binom{N}{l}} \\ &= \frac{(N-t-1)\binom{N-t-2}{l-2}}{\binom{N}{l}} + \cdots + \frac{(l+1)\binom{l-2}{l-2}}{\binom{N}{l}} \\ &\leqslant N^2\left(\frac{l}{N-l}\right)^2\left(\frac{N-t}{N}\right)^{l-2}.\end{aligned}$$

Using the inequality $(1-a)^b \leqslant \mathrm{e}^{-a(b-1)}$, we get that

$$\Pr[s \geqslant t] \leqslant N^2\left(\frac{l}{N-l}\right)^2 \mathrm{e}^{-\frac{t}{N}(l-3)} \leqslant 2^{-(l-3)/\sqrt{\log N}} \leqslant O(2^{-\Omega(\sqrt{\log N})}).$$

Now define a good iteration as an iteration with every subgroup shrinks by more than $\Omega(\sqrt{\log N})$. Clearly, there are at most $O(\log N/\log\log N)$ "good" iterations. Suppose there are totally $m = \Theta(\log N/\log\log N)$ iterations to finish the algorithm. We bound the probability of at least $m/2$ iterations is bad by

$$P = \sum_{i=m/2}^{m}\binom{m}{i}\Pr[s \geqslant t]^i(1-\Pr[s \geqslant t])^{m-i} \leqslant 2^m\Pr[s \geqslant t]^{m/2} = O(2^{-\Omega(\log^{1.5} N/\log\log N)}).$$

Notice that this probability is $O(N^{-\alpha})$ for any constant α. By union bound, with high probability this kind of events never occur in the execution. Therefore, the recursion depth is $O(\log N/\log\log N)$.

Theorem 7. The round complexity of our algorithm for distributed DISTINCT is $O(\log^3 N/\log\log N)$ w.h.p., and the message complexity is $O(N\log^3 N/\log\log N)$.

Proof. The recursive depth of the algorithm is $O(\log/\log\log N)$ with each iteration costs $O(\log^2 N)$ rounds as proved. The round complexity of the algorithm for DISTINCT is $O(\log^3 N/\log\log N)$.

In each iteration, Pre-order Network Partition and Random Permutation Delivery send $O(N)$ and $O(N\log^2 N)$ messages, respectively. The broadcast and converge procedure send $O(N\log N)$ in total for $O(\log N)$ elements are broadcast to the corresponding group. The message complexity for distributed DISTINCT is $O(N\log^3 N/\log\log N)$ in total.

7 Analysis of distributed MODE

7.1 Correctness

For the correctness of our MODE algorithm, we have to address two failure cases below.

(1) One group is over-populated by numbers with the same $H(x)$.

(2) M is set below the real frequency of MODE.

Theorem 8. Set M to the current upper bound on the maximum frequencies and divide the network into $l = O(\frac{N}{M/\log N})$ groups. Choose an independent hash function $H : S \to \{1, 2, \ldots, l-1, l\}$, and then the number of elements mapped to each group is $O(N/l)$ with high probability.

Proof. For a group t, we define a random variable X_t on each element x, such that

$$X_t = \begin{cases} \text{frequency}(x), & H(x) = t, \\ 0, & H(x) \neq t, \end{cases}$$

where x is an element in S. Then the number of elements mapped to group t can be computed as $\sum_x X_t$ and we have

$$\mathrm{E}\left[\sum_x X_t\right] = \frac{N}{l}.$$

Here we import a mathematical tool named Chernoff bound to help us for a further proof.

Theorem 9. If n variables $X_1, X_2, \ldots, X_n$ are independent and have range $\{0, 1\}$, their sum $X = \sum_i X_i$ cannot be too far from the expectation. Concretely,

$$\Pr[X > (1+\delta)\mathrm{E}[X]] < \left(\frac{\mathrm{e}^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mathrm{E}[X]},$$
$$\Pr[X < (1-\delta)\mathrm{E}[X]] < \left(\frac{\mathrm{e}^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mathrm{E}[X]}.$$

By setting $\delta = \frac{\log N}{\mathrm{E}[X]}$ in Theorem 9, we have the following.

Corollary 3. If n variables $X_1, X_2, \ldots, X_n$ are independent and have range $\{0, 1\}$, we have

$$\Pr[X > \mathrm{E}[X] + \log N] < O(1 - N^{-\alpha}),$$
$$\Pr[X < \mathrm{E}[X] - \log N] < O(1 - N^{-\alpha}),$$

where α is a constant.

Since $\frac{N}{l} = M \log N = \Omega(\log N)$, and by Chernoff bound,

$$\Pr[X \geqslant \mathrm{E}[X] + \log N] \leqslant N^{-\alpha},$$

we have

$$\sum_x X_t = M \cdot O\left(\mathrm{E}\left[\sum_x X_x/M\right] + \log N\right) = M \cdot O\left(\frac{N}{lM} + \log N\right) = O(N/l)$$

with high probability $(1 - N^{\alpha})$.

From the above analysis, we do not have to worry about over-populating a group. Notice that to simplify the presentation we assume that the hash function is fully independent. In order to avoid more distribution stages and fit in $\mathcal{CONGEST}$ model, we can invoke Newman's Theorem [23] to reduce the length of the random bits to $O(\log N)$.

Theorem 10. M is the upper bound on the maximum frequency with high probability.

Proof. We have proved that each group contains $O(N/l)$ elements with high probability. Each group is partitioned into l_2 blocks, so each block has $O(N/(l \cdot l_2))$ elements. Let Y_i indicate whether an element y is sampled in the i-th block. Without loss of generality, we assume that the frequency of y is $\Omega(\frac{N \log N}{l \cdot l_2})$. So

$$Y_i = \begin{cases} 1, & y \text{ is sampled in the } i\text{-th block}, \\ 0, & y \text{ is not sampled in the } i\text{-th block}. \end{cases}$$

If the maximum frequency $M' > \Omega(\frac{N \log N}{l \cdot l_2})$, we have

$$\mathrm{E}[\mathrm{P}[Y_i = 1]] = M'/O(N/l) = \Omega(\log N/l_2).$$

Let $Y = \sum_{i=1}^{l_2} Y_i$ be the sum of the random variable Y_i, and then the expectation of Y is

$$\mathrm{E}[Y] = \mathrm{E}\left[\sum_{i=1}^{l_2} Y_i\right] = \sum_{i=1}^{l_2} \mathrm{E}[Y_i] \geqslant \Omega(\log N).$$

By Chernoff bound, we get

$$\mathrm{P}[Y \leqslant 1] \leqslant \mathrm{P}[Y \leqslant (1-\delta)\mathrm{E}[Y]] \leqslant \mathrm{e}^{-\delta^2 \mathrm{E}[Y]/2} \leqslant N^{-\Omega(1)},$$

where $\delta = 1/2$. So the real maximum frequency M' is sampled in one of the blocks and the computed frequency is at least $\Omega(M'/l_2)$ with high probability. If we set the hidden constants in the big O in line 15 in Algorithm 5 large enough, M is the upper bound on the maximum frequency w.h.p.

7.2 Round complexity and message complexity

Lemma 7. The number of iterations needed to shrink M to $O(\log^{O(1)} N)$ in MODE is $O(\log\log N)$ w.h.p.

Proof. At the end of each iteration of decreasing M, M is set to $O(\max\{fl_2, \frac{N\log N}{l\cdot l_2}\})$. We examine the decrease of M in two directions.

(1) If $f\cdot l_2$ is greater than $O(\frac{N\log N}{l\cdot l_2})$, we set $M = O(f\cdot l_2)$. This means that we reach the tight bound of the frequency of MODE.

(2) Otherwise, in each iteration we shrink M by

$$\frac{N\log N}{l\cdot l_2} = O(M^{1-c_2}\cdot \log^2 N),$$

where c_2 is a constant. Thus we get that, after $O(\log\log N)$ iterations, M shrinks to $O(\log^{O(1)} N)$.

Lemma 8. The recursion depth of Simple MODE is $O(\log\log N)$ w.h.p.

Proof. To compute MODE, when M is bounded by $O(\log^{O(1)} N)$, many instances of simple MODE are conducted. Each instance handles $O(\log^{O(1)} N))$ elements independently. We know that an iteration which can shrink the size of the group by half is a good iteration. Let l be the number of sampled elements and s be the size of the resulting group. We have

$$\begin{aligned}
\Pr[s < N/2] = 1 - \Pr[s \geqslant N/2] &= 1 - \sum_{s\geqslant t}(N-s-1)\frac{\binom{N-s-2}{l-2}}{\binom{N}{l}} \\
&\geqslant 1 - \left(\frac{N-N/2}{N}\right)^{l-2}\left(\frac{l}{N-l}\right)^2 \sum_{i=N-t-1}^{l+1} i \\
&\geqslant 1 - N^2\left(\frac{l}{N-l}\right)^2\left(\frac{1}{2}\right)^{l-2} \\
&= 1 - N^{-\Omega(1)}.
\end{aligned}$$

Notice that we can adjust the hidden constant in $l = O(\log N)$ so that the quantity above is $1 - O(N^{-\alpha})$. Then we can use a union bound to claim that every iteration of partition reduces the candidate group's size by at least a factor of 2. So w.h.p., the maximum recursion depth is

$$O(\log(\log^{O(1)} N)) = O(\log\log N).$$

Theorem 11. The round complexity of the algorithm for MODE is $O(\log^2 N\log\log N)$ w.h.p., and the message complexity is $O(N\log N\log\log N)$.

Proof. The number of iterations needed to bound M to $O(\log^{O(1)} N)$ is bounded by $O(\log\log N)$. In each iteration, Pre-order Network Partition takes $O(\log N)$ rounds, and Random Permutation Delivery needs $O(\log^2 N)$ rounds. The frequency computation requires converging results to leaders only and needs $O(\log N)$ rounds. We proved that the recursion depth of the Simple MODE is $O(\log\log N)$ with each iteration taking $O(\log N)$ rounds. So the round complexity is $(\log^2 N\log\log N)$.

There are $O(N \log N)$ messages transferred in total for all nodes in the network in each main iteration. The Simple MODE finishes in $O(\log \log N)$ iterations with $O(N \log N)$ messages in total for all groups transferred in each iteration. In summary, the message complexity of our algorithm for MODE is $O(N \log N \log \log N)$.

8 Conclusion

In this paper, we propose three techniques including Pre-order Network Partition, Pairwise-independent Random Walk, and Random Permutation Delivery. Based on the techniques, we further design distributed algorithms for three representative holistic aggregation functions SELECTION, DISTINCT, and MODE on Random Regular Graphs (RRGs). We prove our algorithms are efficient in both round complexity and message complexity. We believe the efficient algorithm will find their applications in RRGs or expanders based next-generation data centers, and our proposed techniques and their variants will have applications in designing other distributed algorithms for different network topologies.

We conclude the paper with a short discussion on its deployment. A viable deployment is on an SDN-enabled data center, where the routing algorithm, i.e., the Pre-order Network Partition and the Random Permutation Delivery, can be enforced by the control plane. Specifically, the control plane injects forwarding rules to the flow tables in the OpenFlow-enabled switches. Other functionalities including the Pair-wise Random Walk can be realized in each node that runs our protocols. To deploy on the WSNs (wireless sensor networks), a possibility is to adapt to the protocol suggested in [2], which investigated the SELECTION problem under synchronous communication. We shall leave the deployment on asynchronous communications as future work.

Acknowledgements This work was supported in part by National Natural Science Foundation of China (Grant Nos. 61972447, 61832006) and the Fundamental Research Funds for the Central Universities (Grant No. 2019kfyXKJC021). We thank the anonymous reviewers for the helpful comments to improve the presentation of this paper.

References

1 Kuhn F, Locher T, Wattenhofer R. Tight bounds for distributed selection. In: Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures, 2007
2 Li X Y, Wang Y J, Wang Y. Complexity of data collection, aggregation, and selection for wireless sensor networks. IEEE Trans Comput, 2011, 60: 386–399
3 Li H, Wu C, Hua Q S, et al. Latency-minimizing data aggregation in wireless sensor networks under physical interference model. Ad Hoc Netw, 2014, 12: 52–68
4 Bawa M, Garcia-Molina H, Gionis A, et al. Estimating Aggregates on a Peer-to-peer Network. Technical Report, 2003
5 Guan Z T, Zhang Y, Zhu L H, et al. EFFECT: an efficient flexible privacy-preserving data aggregation scheme with authentication in smart grid. Sci China Inf Sci, 2019, 62: 032103
6 Peleg D. Distributed Computing: A Locality Sensitive Approach. Philadelphia: Society for Industrial and Applied Mathematics, 2000
7 Locher T. Foundations of aggregation and synchronization in distributed systems. Dissertation for Ph.D. Degree. Zurich: ETH Zurich, 2009
8 Kuhn F, Locher T, Schmid S. Distributed computation of the mode. In: Proceedings of the 27th ACM Symposium on Principles of Distributed Computing, 2008
9 Wormald N C. Models of random regular graphs. In: London Mathematical Society Lecture Note Series. Cambridge: Cambridge University Press, 1999
10 Singla A, Hong C, Popa L, et al. Jellyfish: networking data centers randomly. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, 2012
11 Singla A, Godfrey P B, Kolla A. High throughput data center topology design. In: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, 2014
12 Leiserson C E. Fat-trees: universal networks for hardware-efficient supercomputing. IEEE Trans Comput, 1985, C-34: 892–901
13 Harary F, Hayes J P, Wu H J. A survey of the theory of hypercube graphs. Comput Math Appl, 1988, 15: 277–289
14 Jyothi S A, Singla A, Godfrey B, et al. Measuring and understanding throughput of network topologies. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2016. 1–12
15 Singla A. Fat-free topologies. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, 2016. 64–70
16 Kassing S, Valadarsky A, Shahaf G, et al. Beyond fat-trees without antennae, mirrors, and disco-balls. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017. 281–294
17 Valadarsky A, Dinitz M, Schapira M. Xpander: unveiling the secrets of high-performance datacenters. In: Proceedings of the 14th ACM Workshop on Hot Topics in Networks, Philadelphia, 2015
18 Valadarsky A, Shahaf G, Dinitz M, et al. Xpander: towards optimal-performance datacenters. In: Proceedings of the 12th International on Conference on Emerging Networking Experiments and Technologies, Irvine, 2016
19 Broder A Z, Shamir E. On the second eigenvalue of random regular graphs (preliminary version). In: Proceedings of the 28th Annual Symposium on Foundations of Computer Science, Los Angeles, 1987
20 Boyd S P, Ghosh A, Prabhakar B, et al. Randomized gossip algorithms. IEEE Trans Inform Theor, 2006, 52: 2508–2530
21 Leighton F T, Maggs B M, Rao S. Packet routing and job-shop scheduling in O(congestion + dilation) steps. Combinatorica, 1994, 2: 167–186

22 Vocking B. Shortest Paths Routing on Arbitrary Networks. Dissertation for Master's Degree. Paderborn: Universität Paderborn, 1994
23 Newman I. Private vs. common random bits in communication complexity. Inf Process Lett, 1991, 39: 67–71

Appendix A Proof of techniques

Appendix A.1 Proof of Lemma 1

Proof. Consider a network with N nodes where the nodes are pre-order labeled by $s_1, s_2, \ldots, s_N$. The labels of nodes in the same group are contiguous. Without loss of generality, select three groups $[s_a, s_b], [s_c, s_d], [s_e, s_f]$ with $s_a < s_b < s_c < s_d < s_e < s_f$, and suppose that there is one edge (s_u, s_v) that is used by all three groups.

If $s_a < s_u < s_v < s_b$, because (s_u, s_v) is also shared by $[s_c, s_d]$, we know that node s_c must be a descendants of s_a, and node s_d must be a predecessor of s_b and a successor of s_a. Accordingly, s_e and s_f hold the same relative position to s_a and s_b with s_c and s_d, which means $s_e < s_d$. This contradicts with our previous assumption.

If $s_c < s_u < s_v$, (s_u, s_v) must not be used by $[s_a, s_b]$ because $s_a < s_b < s_c < s_u < s_v$. This also contradicts the assumption.

Appendix A.2 Proof of Theorem 2

Lemma A1. For any vector x, we have

$$\|M_2 x - \tilde{M}_2 x\|_2 \leqslant \|x\|_2 \cdot O(1/d^2),$$

where the norm $\|\cdot\|$ is the ℓ_2 norm.

Proof. Consider the difference between $(M_2)_{(p,q),(r,s)}$ and $(\tilde{M}_2)_{(p,q),(r,s)}$. We aim at showing that the sum of the absolute value of any row in $M_2 - \tilde{M}_2$ is bounded by $O(d^{-2})$. This is done by enumerating the cases in which two matrices are different.

(1) p wants to swap with q and q wants to swap with p. The probability that both swaps succeed is $1/d^2$ in our random walk M_2, and $1/d^4$ in $\tilde{M}_2$. The probability that only one swap succeeds is 0 in M_2, and $2/d^3(1-1/d)$ in $\tilde{M}_2$. The total amounts of differences are bounded by $1/d^2 - (1/d^4 + 2/d^3(1-1/d)) = O(1/d^2)$.

(2) p wants to swap with q but q wants to swap with another node s. The probability of p successfully swapping with q is 0 in M_2, and $1/d^2$ in $\tilde{M}_2$. The probability of q sucessfully swapping with s is both $1/d^2$ in M_2 and $\tilde{M}_2$ for all s. The total amounts of differences are bounded by $O(1/d^2)$.

(3) p and q both want to swap with the same node r. The probability that both swaps are successful is 0 in M_2 and $1/d^4$ in $\tilde{M}_2$. The probability of single swap succeeding is both $1/d^2$. As p and q share at most d neighbors, total amounts of this kind of differences in one row are bounded by $O(d) \cdot O(1/d^4) = O(1/d^3)$.

Above all, the sum of the absolute value of any row in $M_2 - \tilde{M}_2$ is bounded by $O(1/d^2)$, that is, $\|M_2 - \tilde{M}_2\|_1 = O(d^{-2})$. So we have

$$\begin{aligned}\|M_2 x - \tilde{M}_2 x\|_2 &\leqslant \|x\|_2 \|M_2 - \tilde{M}_2\|_2 \\ &\leqslant \|x\|_2 \|M_2 - \tilde{M}_2\|_1 \\ &\leqslant \|x\|_2 \cdot O(1/d^2).\end{aligned}$$

With Lemma A1, we deduce the eigenvalue gap of M_2.

Lemma A2. The eigenvalue gap of M_2 is $\Omega((1-\lambda_2)/d)$.

Proof. For a standard random walk, the rounds needed to a converge to a stationary state are bounded by $O(\frac{\log N}{1-\lambda_2})$[1)]. Notice in our random walk M, an element changes its position with probability $1/d$. Thus the eigenvalue gap of M shrinks to $\Omega(1-\lambda_2)/d$. As the second largest eigenvalue of $\tilde{M}_2$ is the same as M, the eigenvalue gap of $\tilde{M}_2$ is also $\Omega(1-\lambda_2)/d$.

Let u_1, u_2 and μ_1, μ_2 be the first and second largest eigenvalue of M_2 and $\tilde{M}_2$ respectively. By using Weyl's inequality and Lemma A1, we have $|u_1 - \mu_1| \leqslant \|M_2 - \tilde{M}_2\|_2$ and $|u_2 - \mu_2| \leqslant \|M_2 - \tilde{M}_2\|_2$. Then we get

$$\begin{aligned}u_1 - u_2 &\geqslant -2\|M_2 - \tilde{M}_2\|_2 + \mu_1 - \mu_2 \\ &= \Omega((1-\lambda_2)/d) - O(1/d^2) \\ &= \Omega((1-\lambda_2)/d).\end{aligned}$$

Then we can prove Theorem 2.

Proof. Matrix M_2 is symmetric and its eigenvalues satisfy $\lambda'_1 \geqslant \lambda'_2 \geqslant \cdots \geqslant \lambda'_{N^2}$. It has orthonormal eigenvectors $v_1, \ldots, v_{N^2}$ such that any distribution D can be expressed as $D = \Sigma_{i=1}^{N^2} c_i v_i = \pi + \Sigma_{i=2}^{N^2} c_i v_i$, where $c_i = D^{\mathrm{T}} v_i$, π is a stationary distribution and $DM_2^t = \pi + \Sigma_{i=2}^{N^2} c_i \lambda'^t_i v_i$. Thus,

$$\begin{aligned}\|DM_2^t - \pi\|_1 &= \left\| \sum_{i=2}^{N^2} c_i \lambda'^t_i v_i \right\|_1 \\ &\leqslant \sqrt{N} \left\| \sum_{i=2}^{N^2} c_i \lambda'^t_i v_i \right\|_2 \\ &\leqslant \sqrt{N} \lambda'^t_2 \sqrt{\sum_{i=2}^{N^2} c_i^2} \\ &\leqslant \sqrt{N} \lambda'^t_2.\end{aligned}$$

1) Levin D A, Peres Y, Wilmer E L. Markov Chains and Mixing Times. Providence: American Mathematical Society, 2006.

We denote $\mathbf{1}_{(i,j)}$ as the indicator vector for any pair of elements (i,j), such that $\mathbf{1}_{(i,j),(i,j)} = 1$ and $\mathbf{1}_{(i,j),(p,q)} = 0$ where $(p,q) \neq (i,j)$. We define $\delta_{(i,j)}(t) = \frac{1}{2}\sum_{i\neq j} |\mathbf{1}_{(i,j)} M_2^t - \pi|$, and then we have

$$\delta_{(i,j)}(t) = \frac{1}{2}\|\mathbf{1}_{(i,j)} M_2^t - \pi\|_1 \leqslant \frac{\sqrt{N}}{2}{\lambda'}_2^t.$$

For any $x > 0$, $\ln x \leqslant x - 1$. So

$$\delta_{(i,j)}(t) \leqslant \frac{\sqrt{N}}{2}\mathrm{e}^{-t(1-{\lambda'}_2)}.$$

That is, $\epsilon = O(\frac{\sqrt{N}}{2}\mathrm{e}^{-t(1-{\lambda'}_2)})$. Note that $1-{\lambda'}_2$ is the eigenvalue gap of M_2 and we have proved it is $\Omega((1-\lambda_2)/d)$ in Lemma A2. By setting $t = \Theta(d\log N/(1-\lambda_2))$, for any fixed α, we can make $\epsilon = O(N^{-\alpha})$ and thus ϵ is negligible. So after $O(d\log N/(1-\lambda_2))$ steps, the positions of any pair of elements are uniformly distributed over all $N(N-1)$ nodes.