Algorithms Based on Divide and Conquer for Topic-Based Publish/Subscribe Overlay Design

Chen Chen, *Member, IEEE*, Hans-Arno Jacobsen, *Senior Member, IEEE*, and Roman Vitenberg, *Member, IEEE*

***Abstract*—Overlay design for topic-based publish/subscribe (pub/sub) systems is of primary importance because the overlay forms the basis for the system and directly impacts its performance. This paper focuses on the MinAvg-TCO problem: Use the minimum number of edges to construct a *topic-connected overlay* (TCO) such that all nodes that are interested in the same topic are organized in a directly connected dissemination suboverlay. Existing algorithms for MinAvg-TCO suffer from three key drawbacks: 1) prohibitively high runtime cost; 2) reliance on global knowledge and centralized operation; and 3) nonincremental operation by reconstructing the TCO from scratch. From a practical point of view, these are all severe limitations. To address these concerns, we develop algorithms that dynamically join multiple TCOs. Inspired by the divide-and-conquer character of this idea, we derive a number of algorithms for the original MinAvg-TCO problem that accommodate a variety of practical pub/sub workloads. Both theoretical analysis and experimental evaluations demonstrate that our divide-and-conquer algorithms seek a balance between time efficiency and the number of edges required: Our algorithms cost a fraction (up to 1.67%) of the runtime cost of their greedy alternatives, which come at the expense of an empirically insignificant increase in the average node degree. Furthermore, in order to reduce the probability of poor partitioning at the *divide* phase, we develop a bulk-lightweight partitioning scheme on top of random partitioning. This more refined partitioning imposes a marginally higher runtime cost, but leads to improvements in the output TCOs, including average node degrees and topic diameters.**

***Index Terms*—Algorithm, overlay, publish/subscribe.**

I. Introduction

PUBLISH/SUBSCRIBE (pub/sub) has become a popular communication paradigm that provides a loosely coupled form of interaction among many publishing data sources and many subscribing data sinks [1]–[3]. This work focuses on *topic-based pub/sub*: Publishers associate each publication message with one or more specific topics, and subscribers register their interests in a subset of all topics.

Manuscript received November 25, 2012; revised March 05, 2014 and August 16, 2014; accepted October 09, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Kasera. Date of publication December 02, 2014; date of current version February 12, 2016.

C. Chen and H.-A. Jacobsen are with the Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: chenchen@eecg.toronto.edu; jacobsen@eecg.toronto.edu).

R. Vitenberg is with the Department of Informatics, University of Oslo, 0316 Oslo, Norway (e-mail: romanvi@ifi.uio.no).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2014.2369346

A distributed pub/sub system often organizes nodes (i.e., brokers or servers) as an application-level overlay in a federated or peer-to-peer manner. The overlay infrastructure forms the foundation for distributed pub/sub and directly impacts the system's performance and scalability, e.g., message latency and routing cost. Constructing a high-quality overlay for distributed pub/sub is a key challenge and fundamental problem that has received attention both in industry [1] and academia [4]–[7].

Chockler *et al.* [4] introduced the notion of *topic-connected overlay* (TCO). Informally speaking means that all nodes interested in the same topic are organized in a connected dissemination suboverlay. A TCO ensures that messages published on each topic can be transmitted to all nodes interested in this topic without using noninterested nodes as intermediate relays. Publication routing atop such overlays saves bandwidth and computational resources otherwise wasted on forwarding messages of no interest to the node. It also results in smaller routing tables. Also, from the perspective of security, TCOs are preferred in the context where users within a *trusted group* want to share messages among themselves without having the messages travel outside the group.

A number of existing approaches build separate dissemination overlays on a per-topic (or, more generally, per-interest) basis [8]–[11], thereby attaining the TCO property. While these construction methods are efficient, the resulting overlays may exhibit prohibitively high node degrees. This does not meet the imperative requirement for a pub/sub overlay to have a low link fan-out per node. While overlay designs for different applications might be principally different, they all strive to minimize node degrees, e.g., DHTs [12] and small-world networks [13]. First, it costs a lot of resources to maintain adjacent links for a high-degree node (i.e., monitor the links and the neighbors [4], [5]). For a typical pub/sub system, each link would have to accommodate a number of protocols, service components, message queues, etc. Second, the node degree directly influences the size of the pub/sub routing tables, the complexity of matching, and the efficiency of message delivery. While the benefits for systems where nodes have a limited amount of resources are obvious (e.g., in the context of P2P and mobile), it should be noted that the original motivation for Chockler *et al.*'s design [4] stems from cloud environments. Namely, it was based on the observations that full-mesh approaches do not scale in an IBM data center application environment and that bounded-degree overlays are required. Third, a low diameter is another preferred property for pub/sub overlays [4]: The overlay diameter impacts many performance factors for pub/sub routing, e.g., message latency. With lower

diameter, message delays are likely to be smaller because fewer hops are needed for message delivery. For example, the initial GooPS design required that each source–sink pair would be no more than three overlay hops apart [1].

To address these issues, Chockler *et al.* posed the problem of constructing a TCO with the minimum average node degree, proved the NP-hardness, and developed a greedy approximation algorithm [4]. Other variations of this problem have also been considered [5]. While these approaches give rise to overlays with quite small node degrees, their runtime costs are significant [4], [5]. Furthermore, all these approaches require the global knowledge of all participating nodes including their interests. All these approaches are based on centralized operations that neither lend themselves to decentralized execution nor support dynamic changes to nodes, topics or interests without reconstruction of the TCO from scratch.

From a practical point of view, these constraints are not satisfactory as system failures inevitably lead to network partitions, which motivate the need for joining existing and correctly functioning suboverlays. Similarly, partitions may come from administrative or scheduled maintenance tasks that affect part of the system. It would be excessively expensive to tear down existing connections and to reestablish the overlay from scratch, which is unscalable with the size of the network.

In practice, the set of machines in a data center tends to exhibit nonnegligible variation over time [14], so that the ability to handle dynamic changes (such as node churn) is essential. While global system knowledge is not an absolute put-off for large organizations that often employ various forms of highly available and centralized directory services to reliably track configuration information, a scalable overlay construction algorithm, which can cope with partial system information, poses an appealing alternative. Similar concerns motivate the need for a decentralized solution.

To address these concerns, we propose a number of novel techniques to tackle the TCO construction problem. First, we formulate the TCO join problem and establish its complexity properties before proposing algorithms for this problem. Given two or more TCOs, we aim to construct a single TCO that includes all nodes of the sub-TCOs and respects all interest relations. We establish that it is NP-hard to add minimum number of edges across sub-TCOs. We develop approximation algorithms for this TCO join problem and empirically evaluate their performances. Based on our approach, we devise divide-and-conquer algorithms for the original TCO construction problem, which basically divide the entire network into smaller partitions, conquer each partition separately by building sub-TCOs locally, and then combine the separate sub-TCOs into a single, global TCO. We present a numerical method for selecting the number of partitions, which is a key parameter for the performance of our divide-and-conquer algorithms. We also compare a number different partitioning techniques in the context of divide-and-conquer, including random partitioning, node clustering, and a novel bulk-lightweight partitioning method. We discuss the pros and cons of these techniques, both theoretically and empirically.

Both theoretical analysis and experimental evaluations show that it takes divide-and-conquer algorithms significantly less time to construct a TCO from scratch compared to previous algorithms, at the expense of an insignificant increase in the node degree of the resulting TCO. In our experiments, based on typical pub/sub workloads, we show that the runtime cost of the proposed algorithms is at most 1.67% of that of the existing state-of-the-art approaches, while the average node degree is only 2.12 larger. In addition, our divide-and-conquer approach only requires partial information about nodes, topics, and interests at each phase. Moreover, our divide-and-conquer approach is better suited to accommodate dynamic node joins and leaves that tend to affect small portions of the overlay.

We organize this paper as follows. Section II introduces relevant notation and background information. In Section III, we formally introduce the TCO join problem. In Section IV, we discuss and analyze approaches to this problem. In Section V, we generalize our approach resulting in the design of a divide-and-conquer algorithm that substantially outperforms earlier algorithms. In Section VI, we elaborate on several characteristics and optimizations for the new divide-and-conquer algorithm. In Section VII, we extend the basic random partitioning method for divide-and-conquer in order to obtain a better balance between running time and the quality of output. In Section VIII, we present a thorough experimental evaluation of all algorithms developed in this paper, including comparisons of our work to earlier approaches, the ability of our algorithm to adaptively select the optimal number of partitions, and the performance of divide-and-conquer with different partitioning methods. In Section IX, we put our work in the context of related approaches. We defer a discussion of related work to Section IX to better position our approaches next to a broad scope of related approaches.

II. Background

In this section, we present definitions and background information essential for the understanding of the algorithms developed in this paper.

Let V be the set of nodes and T be the set of topics. We define the interest function $Int : V \times T \rightarrow \{true, false\}$ as: $Int(v, t) = true$ iff node $v \in V$ is interested in topic $t \in T$. Since the domain of Int is a Cartesian product, we also refer to this interest function as an interest matrix.

An overlay network $G = (V, E)$ is an undirected graph over the node set V with the edge set $E \subseteq V \times V$. Given an overlay network $G = (V, E)$, an interest function Int, and a topic $t \in T$, a subgraph $G(t) = (V(t), E(t))$ of G is *induced* by t if $V(t) = \{v \in V | Int(v, t)\}$ and $E(t) = \{(v, w) \in E | v \in V(t) \wedge w \in V(t)\}$. An overlay G is *topic-connected* if $G(t)$contains at most one *topic-connected component (TC-component)* for each topic $t \in T$. We denote a *topic-connected overlay* (TCO) as $TCO(V, T, Int, E)$, TCO for short. Fig. 1 gives an example.

Chockler *et al.* [4] introduced the parameterized family of *Scalable Overlay Construction* (SOC) design problems for pub/sub that captures the tradeoff between the overlay scalability and the cost of message dissemination. Their work focused on the MinAvg-TCO problem for minimizing the number of edges needed to create a TCO for a given instance. Problem 1 offers the formal definition of MinAvg-TCO.

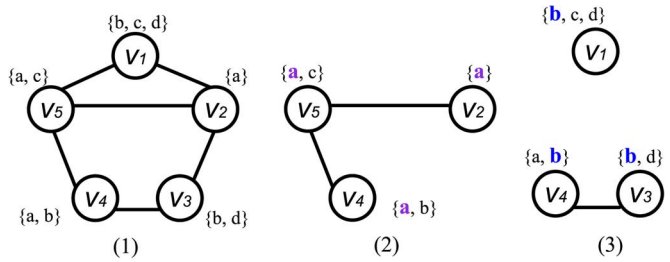

Fig. 1. (1) Overlay G. (2) Subgraph $G(a)$ is topic-connected. (3) Subgraph $G(b)$ is not topic-connected.

Problem 1: MinAvg-TCO (V, T, Int): Given an instance (V, T, Int), where V is a set of nodes, T is a set of topics, and Int is the interest function, construct a TCO with the smallest total number of edges (i.e., the minimum average node degree).

Chockler *et al.* [4] proved the NP-completeness of MinAvg-TCO and proposed the Greedy Merge (GM) algorithm that achieves a logarithmic approximation. GM is centralized and requires complete knowledge of V, T, and Int. The running time for GM is $O(|V|^2|T|)$. GM is only capable of building the TCO from scratch, whereas in practice, due to network partitioning, for instance, the combination of suboverlays is an important concern.

Algorithm 1 specifies the GM algorithm: It starts with an empty edge set and iteratively adds carefully selected edges one by one until attaining a TCO. At each iteration, GM greedily selects an edge whose addition to the overlay would maximally reduce the total number of TC-components for all the topics. In this paper, we use techniques from GM both to devise an algorithm for the TCO join problem in Section IV and to derive a more efficient approach for the MinAvg-TCO problem in Section V-A. GM also serves as an evaluation baseline in Section VIII.

III. MinAvg-TCO-Join Problem

In this section, we introduce the TCO join problem and discuss its similarities and differences with MinAvg-TCO.

The joining of existing TCOs over the same set of topics but disjoint sets of nodes can be achieved by the known approaches for MinAvg-TCO. That is, given sets of nodes and interest functions, compute the union of the sets of nodes and construct a common interest matrix as the union of individual interest functions (the union is well defined as the functions' domains are disjoint), and apply the algorithms for MinAvg-TCO to rebuild the TCO from scratch. However, in practice, it is often better to preserve existing edges and only to incrementally add edges as needed to achieve TCO combined from existing sub-TCOs. This is because establishing a new edge is a relatively costly operation, and incremental computation of additional edges is more efficient than the recomputation of the entire TCO. Also, to rebuild a new overlay from scratch, all existing connections in the existing overlays would have to be torn down, the new routing tables would have to be distributed, and the new connections would have to be established—all adding to the cost of reacting to a network partition or to incrementally evolving a TCO. We give the formal definition of the TCO join problem as follows.

Algorithm 1: Greedy Merge (GM) for MinAvg-TCO

GM (V, T, Int)
Input: V, T, Int
Output: $TCO(V, T, Int, E_{\mathsf{GM}})$
1: **for all** $v \in V$ **do**
2: for all $t \in T$ such that $Int(v, t)$ **do**
3: $Nodes[v][t] \leftarrow \{v\}$
4: **for all** $e = (v, w)$ **do**
5: $contrib \leftarrow |\{t \in T | Int(v, t) \wedge Int(w, t)\}|$
6: add e to $EdgeContrib[contrib]$
7: $E_{\mathsf{GM}} \leftarrow$ **buildEdges**$(V, T, Int, Nodes, EdgeContrib)$
8: **return** $TCO(V, T, Int, E_{\mathsf{GM}})$

buildEdges$(V, T, Int, Nodes, EdgeContrib)$

Input: $V, T, Int, Nodes, EdgeContrib$
/* $Nodes$: a two-dimensional array over $V \times T$ whose elements are subsets of V s.t. $\forall v \in V, t \in T$: 1) $Int(v, t) = true$, and 2) $\forall w \in Nodes[v][t]$: $Int(w, t) = true$, and w, v belong to the same TC-component for t.
$EdgeContrib$: an array of size $|T|$ where each element is an edge set. If edge $e \in EdgeContrib[i]$, then adding e to the current overlay would reduce the number of TC-components by i, where $1 \leq i \leq |T|$. */
Output: E_{GM}, an edge set that forms a TCO for (V, T, Int)
1: $maxContrib \leftarrow max\{i | linkContrib[i] \neq \emptyset\}$
2: $E_{\mathsf{GM}} \leftarrow \emptyset$
3: **while** $maxContrib > 0$ **do**
4: $e \leftarrow$ some edge (v, w) in $EdgeContrib[maxContrib]$
5: $E_{\mathsf{GM}} \leftarrow E_{\mathsf{GM}} \cup \{e\}$
6: delete e from $EdgeContrib[maxContrib]$
7: **for all** t such that $Int(v, t) \wedge Int(w, t)$ **do**
8: **for all** $v' \in Nodes[v][t]$, $w' \in Nodes[w][t]$, $(v', w') \neq (v, w)$, **do**
9: locate i such that $(v', w') \in EdgeContrib[i]$
10: delete (v', w') from $EdgeContrib[i]$
11: add (v', w') to $EdgeContrib[i - 1]$
12: $new\_tcc\_list \leftarrow Nodes[v][t] \cup Nodes[w][t]$
13: **for all** $u \in new\_tcc\_list$ **do**
14: $Nodes[u][t] \leftarrow new\_tcc\_list$
15: $maxContrib \leftarrow max\{i | EdgeContrib[i] \neq \emptyset\}$
16: **return** E_{GM}

Problem 2: MinAvg-TCO-Join(V, T, Int, p): Given p node-disjoint TCOs, i.e., $TCO_d(V_d, T, Int_d, E_d)$, $d = 1, \ldots, p$, construct $TCO(V, T, Int, E)$ with the minimum number of edges, where $V = \bigcup_{d=1}^{p} V_d$, $Int = \bigcup_{d=1}^{p} Int_d$ and $\bigcup_{d=1}^{p} E_d \subseteq E$.

MinAvg-TCO is a special case of MinAvg-TCO-Join when each joining overlay contains only one node. Hence, all impossibility results about MinAvg-TCO (NP-completeness and impossibility of linear approximation unless P = NP) apply to MinAvg-TCO-Join as well. However, as we mentioned above, it is not practical to directly apply the existing MinAvg-TCO algorithms to MinAvg-TCO-Join.

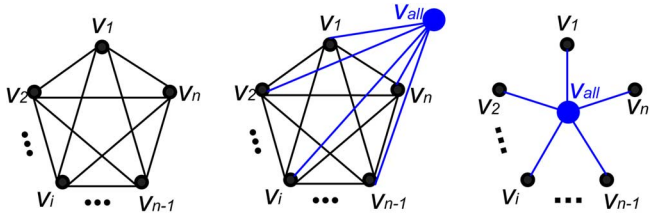

Fig. 2. (a) TCO_0 with $\Theta(n^2)$ edges. (b) TCO_1 built by incremental join has $\Theta(n^2)$ edges. (c) TCO_2 built from scratch has $\Theta(n)$ edges.

Given a MinAvg-TCO-Join instance, it is often more beneficial to tackle the problem incrementally rather than to construct the TCO from scratch. There is, however, a drawback: The total number of edges resulting from incremental addition of edges can be much larger than that of TCO construction from scratch, as illustrated in the following example.

Assume an n-node set $V = \{v_1, v_2, \ldots, v_n\}$ and an n^2-topic set $T = \{t_{ij} | i, j = 1, 2, \ldots, n\}$. We divide T into n groups $T_i = \{t_{ij} | j = 1, 2, \ldots, n\} = \{t_{i1}, t_{i2}, \ldots, t_{in}\}$, $i = 1, 2, \ldots, n$. Topics in which node v_i is interested, denoted by T_{v_i}, comprise two parts: 1) all topics in T_i, and 2) the ith topic t_{ji} from another topic group $T_j (\neq T_i)$. Therefore

$$T_{v_i} = \{t | t = t_{ij} \vee t = t_{ji}, j = 1, 2, \ldots, n\}, \qquad i = 1, 2, \ldots, n.$$

For an arbitrary i, the part of the interest matrix restricted to $V \times T_i$ looks as follows:

$$\begin{array}{c} \\ v_1 \\ v_2 \\ \vdots \\ v_i \\ \vdots \\ v_n \end{array} \begin{array}{c} \begin{array}{cccccc} t_{i1} & t_{i2} & \ldots & t_{ii} & \ldots & t_{in} \end{array} \\ \begin{pmatrix} 1 & 0 & \ldots & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & 1 & \ldots & 1 & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & \ldots & 1 \end{pmatrix} \end{array}.$$

The TCO_0 for this input instance must be an $n \times n$ clique because each v_i has to link to all other nodes in order to achieve topic-connectivity for topics in T_i [see Fig. 2(a)].

Assume a new node v_{all}, which is interested in all topics in the entire topic set T, requests to join. If we construct TCO_1 by incrementally adding edges to TCO_0, then the number of edges in TCO_1 is still $\Theta(n^2)$, as illustrated in Fig. 2(b). However, if we delete all existing edges and reconstruct the TCO from scratch, it is sufficient to simply connect v_{all} to all other $\{v_1, v_2, \ldots, v_n\}$, which results in a TCO_2 with $\Theta(n)$ edges [cf. Fig. 2(c)].

This example shows that the number of edges resulting from incremental addition can be $\Theta(n)$ times larger than the number of edges when building an TCO from scratch. However, our experimental findings in Section VIII show that for typical pub/sub workloads, this situation virtually never occurs in practice.

Next, we design several algorithms for MinAvg-TCO-Join, which incrementally add cross-TCO edges.

IV. Algorithms for MinAvg-TCO-Join Problem

We first devise the Naive Merge (NM) algorithm for MinAvg-TCO-Join, as specified in Algorithm 2. NM is based on a greedy heuristic giving rise to similar properties as for GM. We include the design and analysis of NM below to illustrate its weaknesses that motivate the need for a more sophisticated algorithm, presented thereafter.

The intuition behind NM is that we need to determine a set of cross-TCO edges that, in conjunction with the existing edges internal to the TCOs, produce a combined TCO. Below, we refer to inner edges as E_{inNM} and outer edges as E_{outNM}. We can obtain E_{inNM} by the union of all edges in sub-TCOs. NM starts with an empty E_{outNM} and iteratively adds edges one by one until the TCO is attained. At each iteration, NM selects a cross-TCO edge whose addition to the current overlay would maximally reduce the total number of TC-components.

Algorithm 2 follows the design of the GM algorithm given in Section II. The correctness, approximation ratio, and running time properties below can be established by adapting the respective proofs for the GM algorithm. For brevity, we only focus on the key differences and place the complete analysis of all proofs in the Appendix and in the technical report [15].

Lemma 1: Algorithm 2 outputs a TCO for the input (V, T, Int).

Line 5 in Algorithm 2 computes the set E_{outNM} of all cross-TCO edges. Let E'_{out} be an optimal edge set of cross-TCO edges, and T_d be the set of topics covered by overlay nodes in V_d, i.e., $T_d = |\{t \in T | \exists v \in V_d \text{ s.t. } Int_d(v, t)\}|$. Note, T_d is also the number of TC-components for TCO_d, and thus $\sum_{d=1}^{p} T_d$ is the total number of TC-components for the input of all TCOs. With proof techniques similar to [4, Lemma 6.5], we establish Lemma 2.

Lemma 2: Algorithm 2 has a log approximation ratio for the total number of edges in the TCO: $|E_{\text{outNM}}|/|E'_{\text{out}}| = O(\log(\sum_{d=1}^{p} T_d))$.

Lemma 3: The running time of Algorithm 2 is $\mathbb{T}_{\text{NM}} = O(|T| \cdot \sum_{i=1}^{p} \sum_{j>i}^{p} |V_i||V_j|)$.

Algorithm 2 also retains the undesirable properties of GM. The high running time cost as given by Lemma 3 is not insignificant. NM cannot be easily decentralized, and the overlay computation requires the complete knowledge of (V, T, Int). These shortcomings motivate the development of a new algorithm.

Our new algorithm for MinAvg-TCO-Join is based on the notion of a *star set* defined and illustrated next.

Definition 1: (*Star set*) Given an instance (V, T, Int), a *star set* is a subset $S \subseteq V$ such that

$$\bigcup_{s \in S} \{t \in T | Int(s, t)\} = \bigcup_{v \in V} \{t \in T | Int(v, t)\}.$$

Node s is a *star node* (or a *star*) if $s \in S$.

As illustrated in Fig. 3, a star set is a subset of overlay nodes that represents the interests of all the nodes in the overlay. The complete node set V is always a star set, but there probably exist many other star sets with much fewer stars. Star nodes can function as bridges for the purpose of determining cross-TCO connections. It is easy to see that it is possible to attain full topic-connectivity only by using cross-TCO links among star sets of different sub-TCOs. Suppose we have a number of TCOs such that each TCO includes nodes interested in a topic $t \in T$. Then, each of the star sets includes a node interested in t.

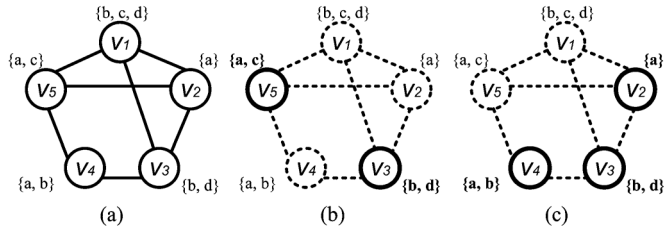

Fig. 3. (a) TCO. (b) $\{v_3, v_5\}$ is a star set thst covers all topics $\{a, b, c, d\}$. (c) $\{v_2, v_3, v_4\}$ is not a star set; it only covers $\{a, b, d\}$.

Algorithm 2: Naive Merge for MinAvg-TCO-Join

NaiveMerge(L_{TCO})
Input: A list L_{TCO} of p node-disjoint topic-connected overlays: $TCO_d(V_d, T, Int_d, E_d)$, $d = 1, \ldots, p$
Output: A topic-connected overlay $TCO(V, T, Int, E_{\text{NM}})$ where $V = \bigcup_{d=1}^{p} V_d$, $Int = \bigcup_{d=1}^{p} Int_d$, $\bigcup_{d=1}^{p} E_d \subseteq E_{\text{NM}}$

1: $V \leftarrow \bigcup_{d=1}^{p} V_d$; $Int \leftarrow \bigcup_{d=1}^{p} Int_d$; $nodesInts \leftarrow \emptyset$
2: **for** $d = 1$ to p **do**
3: add (V_d, Int_d) to $nodesInts$
4: $E_{\text{inNM}} \leftarrow \bigcup_{d=1}^{p} E_d$
5: $E_{\text{outNM}} \leftarrow$ **greedyConnect**$(V, T, Int, nodesInts)$
6: $E_{\text{NM}} \leftarrow E_{\text{inNM}} \cup E_{\text{outNM}}$
7: **return** $TCO(V, T, Int, E_{\text{NM}})$

greedyConnect$(V, T, Int, nodesInts)$
Input: $V, T, Int, nodesInts$
$//nodesInts$: A list of p (V_d, Int_d) pairs
Output: E_{out}, a set of cross-TCO edges that forms a TCO along with inner overlay edges

1: **for** $d = 1$ to p **do**
2: **for all** $t \in T$ **do**
3: **for all** $v \in V_d$ **do**
4: $Nodes[v][t] \leftarrow \emptyset$
5: $intNodes \leftarrow \{v \in V_d | Int_d(v, t)\}$
6: **for all** $v \in intNodes$ **do**
7: $Nodes[v][t] \leftarrow intNodes$
8: **for all** $e = (v, w)$ such that $v \in V_i, w \in V_j, i \neq j$ **do**
9: $weight \leftarrow |\{t \in T : Int_i(v, t) \wedge Int_j(w, t)\}|$
10: **if** $weight > 0$ **then**
11: add e to $EdgeContrib[weight]$
12: **return buildEdges**$(V, T, Int, Nodes, EdgeContrib)$

By connecting nodes from different star sets, we can achieve topic-connectivity for t.

Since the minimum star sets tend to be substantially smaller than the entire node set, considering only star nodes as candidates for cross-TCO links gives rise to a number of advantages. One, the running time of the TCO construction algorithms considered in this paper is roughly proportional to the square of the number of nodes, so the algorithm would run much faster if we only consider star nodes. Two, we can compute star sets of different TCOs in parallel in a fully decentralized fashion. Three, calculation of cross-TCO edges no longer requires the complete knowledge of (V, T, Int). A partial view of star nodes and their interests is sufficient.

Algorithm 3: Star Merge algorithm for MinAvg-TCO-Join

StarMerge(L_{TCO})
Input: A list L_{TCO} of p node-disjoint TCOs: $TCO_d(V_d, T, Int_d, E_d)$, $d = 1, \ldots, p$
Output: $TCO(V, T, Int, E_{\text{SM}})$, where $V = \bigcup_{d=1}^{p} V_d$, $Int = \bigcup_{d=1}^{p} Int_d$, and $\bigcup_{d=1}^{p} E_d \subseteq E_{\text{SM}}$

1: $V \leftarrow \bigcup_{d=1}^{p} V_d$; $Int \leftarrow \bigcup_{d=1}^{p} Int_d$; $nodesInts \leftarrow \emptyset$
2: **for** $d = 1$ to p **do**
3: $T_d \leftarrow \{t \in T | \exists v \in V_d \text{s.t.} Int(v, t)\}$
4: $T_{\text{out\_}d} \leftarrow T_d \bigcap (\bigcup_{i \neq d} T_i)$
5: $S_d \leftarrow$ **getStarSet**$(V_d, T_{\text{out\_}d}, Int_d)$
6: add $(S_d, Int_d|_{S_d})$ to $nodesInts$
7: $E_{\text{inSM}} \leftarrow \bigcup_{d=1}^{p} E_d$
8: $E_{\text{outSM}} \leftarrow$ **greedyConnect**$(V, T, Int, nodesInts)$
9: $E_{\text{SM}} \leftarrow E_{\text{inSM}} \cup E_{\text{outSM}}$
10: **return** $TCO(V, T, Int, E_{\text{SM}})$

getStarSet (V, T, Int)

1: $S \leftarrow \emptyset$ and $Rest \leftarrow \{t \in T | \exists v \in V \text{s.t.} Int(v, t)\}$
2: **while** $Rest \neq \emptyset$ **do**
3: $s \leftarrow \arg\max_{v \in V - S} |\{t | t \in Rest \wedge Int(v, t)\}|$
4: $S \leftarrow S \cup \{s\}$; $Rest \leftarrow Rest - \{t | Int(s, t)\}$
5: **return** S

We still need to consider, how to efficiently determine a minimum star set given (V, T, Int). The problem of computing a minimum star set turns out to be precisely equivalent (through a linear reduction) to the classic *minimum set cover* problem, which is NP-complete but has a logarithmic approximation. Note that the size of the star set only affects the performance of our algorithm rather than its correctness. In Algorithm 3, `getStarSet()` provides a standard greedy implementation that attains a provable logarithmic approximation. It starts with an empty star set and continues adding nodes to the star set one by one until all topics of interest are covered. At each iteration, the algorithm selects a node that is interested in the largest number of uncovered topics.

Algorithm 3 presents our Star Merge (SM) algorithm. SM operates in two phases. First, SM determines a star set for each sub-TCO. Note that the star set for TCO_d does not need to cover all of T_d. It suffices to cover $T_{\text{out\_}d} = T_d \bigcap (\bigcup_{i \neq d} T_i)$. Second, SM connects all the nodes in the star sets into a TCO greedily by always selecting an edge with the maximum contribution.

Below, we establish the properties of SM.

Lemma 4: Algorithm 3 outputs a TCO for the input (V, T, Int).

Let E'_{out} and T_d follow the same definitions as for Lemma 2. Let E_{outSM} be the edge set computed by Line 8 of Algorithm 3. This edge set connects star nodes in R_d $(d = 1, \ldots, p)$ across all TCOs. Let E'_{outSM} be the minimum edge set to connect R_d $(d = 1, \ldots, p)$. Then, the following properties for Algorithm 3 hold.

Lemma 5: The output of Algorithm 3 has a log approximation ratio as compared to the optimal solution of just connecting stars: $|E_{\text{outSM}}|/|E'_{\text{outSM}}| = O(\log(\sum_{d=1}^{p} T_d))$.

Let $S_d^* \subseteq V_d$ be the star set of minimum size for $(V_d, T_{\text{out\_}d}, Int_d)$. Then, $|S_d| = O(\log(|T_{\text{out\_}d}|)|S_d^*|), \forall d = 1, \ldots, p$, and the following result holds.

Lemma 6: Algorithm 3 has the following approximation ratio: $|E_{\text{outSM}}|/|E'_{\text{out}}| = O(\log(\sum_{d=1}^{p} T_d) \cdot (\log|T|)^2 \cdot \sum_{d=1}^{p} |S_d^*|)$.

Lemma 7: The running time of Algorithm 3 is

$$\mathbb{T}_{\text{SM}} = O\left(|T| \cdot \left(\sum_{d=1}^{p} |V_d| \log|V_d| + (\log|T|)^2 \sum_{i=1}^{p} \sum_{j>i}^{p} |S_i^*| \left|S_j^*\right|\right)\right).$$

With Lemmas 3 and 7, we can see that SM improves the running time significantly because for most cases $|S_d^*| \ll |V_d|$ holds. We also show this experimentally in Section VIII-B.

The same idea to reduce the number of nodes by choosing from star nodes can be applied to solve MinAvg-TCO, which we discuss in Section V.

V. Divide-and-Conquer for MinAvg-TCO

We extend SM for MinAvg-TCO-Join to tackle the original MinAvg-TCO problem in a divide-and-conquer manner. We first develop the divide-and-conquer algorithm in Section V-A, and then present its main analytical properties in Section V-B. We show analytically that the proposed algorithm leads to a significantly improved running time cost as compared to GM. In Section VIII, we quantify these improvements empirically.

A. DC Algorithm

According to [4], the number of nodes is a dominant factor for the running time of the GM algorithm (i.e., $O(|V|^2|T|)$) and a serious drawback to the algorithm's performance and scalability. We can improve the running time of GM by reducing the size of the node set. This suggests a divide-and-conquer strategy for MinAvg-TCO: 1) *divide* the input MinAvg-TCO instance into several subinstances with smaller node sets; 2) *conquer* each sub-MinAvg-TCO instance independently and build sub-TCOs; and then 3) *combine* these sub-TCOs by adding cross-TCO edges and return one TCO as an output for the original instance.

Following this design, we present the Divide-and-Conquer (DC) algorithm in Algorithm 4. We employ GM to *conquer* the sub-MinAvg-TCO instances by determining inner edges used for the construction of the suboverlays. The SM algorithm can tackle the *combine* phase by adding cross-TCO edges (i.e., outer edges) in a greedy manner. To *divide* nodes, the algorithm uses a random partitioning. The number of partitions p is obtained by function `chooseP()` in Line 2 of Algorithm 4, for which we give a detailed algorithm and analysis in Section VI. Each partition contains $k = |V_d| = |V|/p$ nodes, where $d = 1, \ldots, p$.

There are two methods to *divide* the nodes: 1) node clustering, and 2) random partitioning. Node clustering is partitioning the original node set into groups such that nodes with similar interests are placed in the same group while nodes with diverging interests belong to different groups. Random partitioning assigns each node in the given node set to one of the partitions based on a uniformly random distribution.

Algorithm 4: Divide-and-Conquer algorithm for MinAvg-TCO

DivideAndConquerForTCO (V, T, Int)
Input: V, T, Int
Output: A topic-connected overlay $TCO(V, T, Int, E_{\text{DC}})$
1: $E_{\text{DC}} \leftarrow \emptyset$
2: $p \leftarrow \textbf{chooseP}(V, T, Int)$
3: $L_{\text{TCO}} \leftarrow \emptyset$
4: Randomly divide V into p partitions V_d, $d = 1, 2, \ldots, p$
5: **for** $d = 1$ to p **do**
6: $\quad Int_d \leftarrow Int|_{V_d}$
7: $\quad TCO_d(V_d, T, Int_d, E_d) \leftarrow \textbf{GM}(V_d, T, Int_d)$
8: $\quad$ add TCO_d to L_{TCO}
9: $TCO(V, T, Int, E_{\text{DC}}) \leftarrow \textbf{StarMerge}(L_{\text{TCO}})$
10: **return** $TCO(V, T, Int, E_{\text{DC}})$

Node clustering seems appealing because well-clustered nodes with strongly correlated interests would result in GM producing lower average node degrees. The problem with this approach is its relative inefficiency. Clustering algorithms tend to exhibit high runtime cost. Additionally, they require the computation of a *distance* metric among nodes. In our case, this translates to calculating pairwise correlations among node interests with significant runtime cost implications. It is challenging to fit node clustering into the DC algorithm such that the latter is still superior to the GM algorithm in terms of runtime cost. Furthermore, it is difficult to devise an effective decentralized algorithm for node clustering that would not require complete knowledge of (V, T, Int). Besides, node clustering by interests may yield clusters that vary in size depending on the clustering algorithm used.

We choose random partitioning for our DC algorithm because it is extremely fast, more robust than node clustering, and supports decentralized implementation. Also, it is not difficult to obtain equal-sized partitions by random partitioning, which is optimal with respect to both average node degree and running time. Furthermore, the construction of inner edges for each sub-TCO only requires knowledge of node interests within the suboverlay. Hence, random partitioning can be oblivious to the composition of nodes and their interests. The shortcoming of using random partitioning for the DC algorithm is the potentially higher average node degree because random partitioning may place nodes with diverging interests into the same partition thereby reducing the amount of correlation that is present in the original node set. In Section VIII, we validate the effect of the increase in the average node degree empirically and show that this effect is insignificant.

In order to diminish the risk of suboptimality due to random partitioning and to guarantee the quality of the output TCO, in Section VII, we design a more refined partitioning method in which the nodes that subscribe to large number of topics are replicated in each partition.

B. DC Algorithm Analysis

In this section, we prove correctness, approximation ratio, and running time properties for the DC algorithm.

Lemma 8: Algorithm 4 outputs a TCO for the input (V, T, Int).

Given an instance of the MinAvg-TCO problem with (V, T, Int), suppose $TCO_{\mathrm{OPT}}(V, T, Int, E_{\mathrm{OPT}})$ is an optimal solution for it. As defined before, p denotes the number of partitions provided for Algorithm 4. There are two types of edges that form the TCO produced by Algorithm 4: 1) E_{inDC}, the inner edges that are computed for each suboverlay using the GM algorithm in Lines 5–8; 2) E_{outDC}, the outer edges to connect star nodes across different sub-TCOs using the SM algorithm in Line 9. The following lemma is based on the assumption that the optimal overlay for $(V', T, Int|_{V'})$ where $V' \subset V$ has fewer edges than the optimal overlay TCO' for (V, T, Int).

Lemma 9: Algorithm 4 has the following approximation ratio for the total number of edges in the TCO: $|E_{\mathsf{DC}}|/|E_{\mathsf{OPT}}| = O(p \cdot \log(|V||T|))$.

Next, we look at the running time of the DC algorithm. $\mathbb{T}_{\mathrm{inDC}}$ denotes the running time to build E_{inDC} in the loop in Lines 5–8. $\mathbb{T}_{\mathrm{outDC}}$ denotes the running time to build E_{outDC} in Line 9, and let $\mathbb{T}_{\mathsf{DC}}$ be the total running time cost of Algorithm 4.

Lemma 10: The running time of Algorithm 4 is

$$\mathbb{T}_{\mathsf{DC}} = O\left(|T| \cdot \left(\frac{1}{p}|V|^2 + (\log|T|)^2 \sum_{i=1}^{p}\sum_{j>i}^{p} |S_i^*||S_j^*|\right)\right)$$

$$\mathbb{T}_{\mathrm{inDC}} = O\left(|T| \cdot \sum_{d=1}^{p} |V_d|^2\right)$$

$$\mathbb{T}_{\mathrm{outDC}} = O\left(|T| \cdot (\log|T|)^2 \cdot \sum_{i=1}^{p}\sum_{j>i}^{p} |S_i^*|\,|S_j^*|\right).$$

Let us denote the average size of a star set for a sub-TCO as $\tilde{k}$. In Section VIII, we evaluate $\tilde{k}$'s statistical properties

$$\tilde{k} = \overline{|S_d|} = \frac{\sum_{d=1}^{p} |S_d|}{p}.$$

Lemma 11 is given to simplify the analysis of the running time cost of the DC algorithm presented in Section VI.

Lemma 11: The running time of Algorithm 4 is

$$\mathbb{T}_{\mathsf{DC}} = O\left(|T| \cdot \left(\frac{1}{p}|V|^2 + p^2\tilde{k}^2\right)\right).$$

C. Decentralizing the DC Algorithm

The DC algorithm (Algorithm 7) is fully centralized. We can decentralize DC in the following ways: 1) nodes autonomously organize themselves into random partitions; 2) different partitions construct inner edges in parallel, i.e., the nodes within each partition exchange their interests and execute GM; 3) different partitions compute star sets in parallel; 4) members of all star sets exchange their interests and compute outer edges. This decentralized implementation has several important advantages: The parallel calculation reduces the total running time and distributes the computational load. Furthermore, decentralization eliminates the need for a central entity that must have full knowledge of all nodes and their interests, as we further elaborate on in Section VI. Note, the original GM algorithm does not lend itself to such decentralization.

VI. Selecting the Size of Partitions for DC

Our DC algorithm is parameterized with p, the number of partitions. This parameter impacts all important characteristics of DC. When $p = 1$, the performance is dominated by the invocation of the GM algorithm at the *conquer* phase. In this case, average node degree, running time, and other performance characteristics are identical to the performance of the GM algorithm when applied to MinAvg-TCO. At the other end of the spectrum where p is close to $|V|$, the invocation of SM at the *combine* phase dominates the performance of DC. In this case, as we observed in Section IV, SM is once again identical to GM when applied to MinAvg-TCO. Thus, it is the intermediate values of p that represent the interesting balance between the *conquer* and *combine* phases.

With respect to the average node degree, it is desirable to maintain p reasonably small because p bounds the approximation ratio, as Lemma 9 shows. We will present further evaluation in Section VIII.

With respect to the running time, the analysis in Section V-A establishes that the contribution of both partitioning the nodes and computing the star sets is of a small order of magnitude in comparison to the time needed for computing inner edges at the *conquer* phase and outer edges at the *combine* phase. The time needed for computing inner edges is bounded by $(1/p)|V|^2|T|$ (under the assumption that inner edges for different suboverlays are computed sequentially rather than in parallel), while the time required for computing outer edges is proportional to $(\tilde{k}/k)^2|V|^2|T|$. Approximately speaking, the combined sum of these two values is minimal if $1/p + (\tilde{k}/k)^2$ is minimal, which occurs if $\mathrm{d}/\mathrm{dp}(1/p + (\tilde{k}/k)^2) = 0$.

If, on the other hand, inner edges for different suboverlays are computed in parallel in a decentralized fashion as discussed in Section V-C, then this computation requires time proportional to $1/p^2|V|^2|T|$. Then, the total running time is minimal if $1/p^2 + (\tilde{k}/k)^2$ is minimal, or $\mathrm{d}/\mathrm{dp}(1/p^2 + (\tilde{k}/k)^2) = 0$.

A practical application of this running time analysis is complicated by the fact that it is difficult to assess $\tilde{k}$ analytically. However, our analysis suggests an adaptive way for selecting p. Since partitioning the nodes and computing the star sets is relatively cheap, we can try partitioning for different p values. Each time, we only compute the star sets and thus obtain $\tilde{k}$ without running the expensive calculation of inner and outer edges. Then, we use fast numerical methods to approximately determine the minimum of $1/p + (\tilde{k}/k)^2$ or $1/p^2 + (\tilde{k}/k)^2$.

Another important performance characteristic of the TCO design algorithms is the *potential neighbor set*. We define the *potential neighbor set* for a node v as the set of nodes that are considered candidates for becoming a neighbor of v when executing an algorithm. We denote by $pn\text{-}size(v)$, the size of v's potential neighbor set, also referring to it as *potential neighbor size*. Note that $pn\text{-}size(v)$ is the maximum fan-out that an overlay design algorithm may yield for node v. According to DC, the potential neighbor set for v consists of two subsets: 1) nodes in the same partition as s: $\{u|u(\neq v) \in V_d \wedge v \in V_d\}$; 2) all other star set nodes: $\{s|s \in S_i(\neq S_d) \wedge v \in S_d\}$ (if v belongs to the star set S_d). Consequently, there are two classes of nodes with respect

to the potential neighbor set

$$pn\text{-}size(v) = \begin{cases} (k-1), & \text{if } v \text{ is not a star node} \\ (k-1)+(p-1)\tilde{k}, & \text{if } v \text{ is a star node.} \end{cases}$$

The significance of this characteristic is manyfold. First of all, the cardinality of the *potential neighbor set* directly impacts the running time for computing inner edges and outer edges of DC because it implies how many nodes are processed when choosing a link to add (see empirical results in Section VIII). Second, the maximum node degree has an importance of its own. While the GM algorithm strives to minimize the average node degree, it may be severely suboptimal with respect to the degree of individual nodes by producing star-like overlays as it has been observed in [5]. Therefore, minimizing the potential neighbor size has a desirable effect from this point of view. Third, the potential neighbor size has an additional important meaning for the decentralized implementation of the DC algorithm discussed in Section V-C. For a node v, $pn\text{-}size(v)$ is precisely equal to the number of nodes whose interests v may need to learn about in the course of an execution. This is important because gathering nodes' interests in a scalable and robust decentralized manner is a problem in its own right.

We extend the definitions of potential neighbor size to apply to the entire node set

$$pn\text{-}size(V) = \max_{v \in V} pn\text{-}size(v) = (k-1)+(p-1)\tilde{k}.$$

If we consider $pn\text{-}size(V)$ as a function of p, it becomes minimal when $k+(p-1)\tilde{k}$ is minimal. Similar to the analysis of the running time, further development of this derivation requires to assess $\tilde{k}$ analytically. It is also possible, however, to select p adaptively so as to minimize the potential neighbor size by partitioning the nodes and computing the star sets and using efficient numerical methods to approximately determine the minimum.

To this end, we consider $pn\text{-}size$ as a function of p and search for p^*, the value of p that approximately yields the minimum $pn\text{-}size$. As shown in `estimatePNSize()` in Algorithm 5, given p, we estimate $pn\text{-}size$ as a function of p by performing random partitioning of V, computing the star sets for each partition, and then summing the sizes of these star sets for this particular collection of partitions. This method produces reasonable estimations because the sizes of star sets (and therefore $pn\text{-}size$) stay stable with a small deviation across different trials of random partitioning (see Section VIII-C). Besides, we can implement `estimatePNSize()` efficiently, and the cumulative time complexity of all invocations in Algorithm 5 is negligible as long as we only call this function a few times. The brute-force method requires $\Theta(|V|)$ invocations: We can estimate $pn\text{-}size$ with all values of p in the domain $[2, |V|]$ and choose p^* that yields the minimum. Fortunately, the shape of dependency of $pn\text{-}size$ on p allows us to find p^* more efficiently with only $\Theta(\log p^*)$ invocations. Section VIII-C shows that $pn\text{-}size$, as a function of p, roughly exhibits a convex profile: The curve of the function lies below the line segment joining two endpoints at $p=2$ and $p=|V|$. Starting from $p=2$, as the number of partitions increases, $pn\text{-}size$ would first decrease and then increase; the point of transition is close to the leftmost endpoint, where $p=2$. With this in mind, we use a *one-sided binary search* to find p^* that approximately yields the minimum $pn\text{-}size$, as specified in Algorithm 5. First, Lines 2–5 check `estimatePNSize(q)` repeatedly at larger intervals ($q = 2, 4, 8, 16, \ldots$) until we find a q such that `estimatePNSize(q)` stops decreasing. We obtain a window containing the target p^*, i.e., $[q/2, q]$, with $\leq \log 2p^*$ invocations of `estimatePNSize()`. Second, Lines 6–15 try to locate $p^* \in [q/2, q]$ with a binary search, which takes $\leq \log p^*$ invocations of `estimatePNSize()`. In total, Algorithm 5 runs `estimatePNSize()` at most $(2\log p^* + O(1))$ times, which is of smaller order of magnitude as compared to the runtime cost of computing TCO edges in DC (see Lemma 10). Our evaluation in Section VIII further validates that the running time Algorithm 5 is practically negligible in DC.

Algorithm 5: Choose the Number of Partitions by $pn\text{-}size$

choosePbyPNSizeEstimation (V, T, Int)
Input: V, T, Int
Output: p^*: the number of partitions, $2 \leqslant p^* \leqslant |V|$
1: $q \leftarrow 2$; $pn\text{-}size_{\text{new}} \leftarrow$ **estimatePNSize**(q)
2: **repeat**
3: $\quad pn\text{-}size_{\text{old}} \leftarrow pn\text{-}size_{\text{new}}$
4: $\quad q \leftarrow 2q$; $pn\text{-}size_{\text{new}} \leftarrow$ **estimatePNSize**(q)
5: **until** $pn\text{-}size_{\text{new}} >= pn\text{-}size_{\text{old}}$
6: $low \leftarrow (q/2)$; $high \leftarrow q$
7: $pn\text{-}size_{\min} \leftarrow pn\text{-}size_{\text{old}}$
8: $p^* \leftarrow 0$, $pn\text{-}size_{\text{est}} \leftarrow 0$
9: **while** $low < high$
10: $\quad p^* \leftarrow \frac{low+high}{2}$
11: $\quad pn\text{-}size_{\text{est}} \leftarrow$ **estimatePNSize**(p^*)
12: $\quad$ **if** $pn\text{-}size_{\text{est}} < pn\text{-}size_{\min}$ **then**
13: $\quad\quad low \leftarrow p^*$; $pn\text{-}size_{\min} \leftarrow pn\text{-}size_{\text{est}}$
14: $\quad$ **else**
15: $\quad\quad high \leftarrow p^*$
16: **return** p^*

estimatePNSize(p)
Input: p, the number of partitions
Output: $pn\text{-}size_{\text{est}}$, an estimate of $pn\text{-}size$ as a function of p
1: $pn\text{-}size_{\text{est}} \leftarrow |V|/p$
2: Randomly divide V into p partitions V_d, $d = 1, 2, \ldots, p$
3: **for** $d = 1$ to p **do**
4: $\quad S_d \leftarrow$ **getStarSet**(V_d, T_d, Int_d)
5: $\quad pn\text{-}size_{\text{est}} \leftarrow pn\text{-}size_{\text{est}} + |S_d|$
6: **return** $pn\text{-}size_{\text{est}}$

VII. Divide-and-Conquer With Bulk Clones

As both the example in Fig. 2 and the evaluation in Section VIII indicate, DC may produce much higher node degrees than GM because of partitioning. Fig. 2 further suggests that the node degrees of a TCO are highly sensitive to the placement of *bulk subscribers*—nodes that subscribe to a lot of topics. The impact of bulk subscribers is exacerbated by the fact that representative pub/sub workloads follow the "Pareto 80–20" rule [16]: Most nodes subscribe to a small number of topics. A partition without bulk subscribers would

Algorithm 6: Divide-and-Conquer with Bulk Clones

DCBulkClone (V, T, Int)
Input: V, T, Int
Output: $TCO(V, T, Int, E_{\mathsf{DCBC}})$
1: $\eta \leftarrow \mathbf{chooseEta}(V, T, Int)$
2: $B \leftarrow \{v \in V : |T(v)| > \eta\}$; $L \leftarrow V - B$
3: $T_{\text{restricted}} \leftarrow T - \bigcup_{v \in B} T(v)$
4: $S \leftarrow \mathbf{getStarSet}(L, T_{\text{restricted}}, Int|_L)$
5: $B \leftarrow B \cup S$; $L \leftarrow L - S$
6: $TCO_B(B, T, Int|_B, E_B) \leftarrow \mathbf{GM}(B, T, Int|_B)$
7: $p \leftarrow \mathbf{chooseP}(V, T, Int)$
8: Randomly divide L into p partitions L_d, $d = 1, 2, \ldots, p$
9: **for** $d = 1$ to p
10: $\quad BL_d \leftarrow B \cup L_d$; $Int_d \leftarrow Int|_{BL_d}$; $nodesInts_d \leftarrow \emptyset$
11: $\quad$ add $(B, Int|_B)$ to $nodesInts_d$
12: $\quad$ **for** $v \in L_d$ **do**
13: $\quad\quad$ add $(v, Int|_v)$ to $nodesInts_d$
14: $\quad E_d \leftarrow \mathbf{greedyConnect}(BL_d, T, Int_d, nodesInts_d)$
15: $E_{\mathsf{DCBC}} \leftarrow E_B \cup (\bigcup_{d=1}^{p} E_d)$
16: **return** $TCO(V, T, Int, E_{\mathsf{DCBC}})$

require significantly more links to attain the TCO property; it is therefore desirable to have bulk subscribers in each partition.

In view of this observation, we design Algorithm 6, the Divide-and-Conquer with Bulk Clones (DCBC) algorithm. We apply random partitioning merely to lightweight subscribers, while replicating bulk subscribers in addition to every lightweight partition. As a result, DCBC preserves the subscription correlation for each partition and thus achieves an improved average node degree as compared to DC. DCBC works as follows: First, Lines 1–5 perform a bulk-lightweight partitioning of the input. Second, Line 6 greedily builds a suboverlay TCO_B for all bulk subscribers. Third, Lines 7–8 impose random partitioning upon lightweight nodes. Finally, Lines 9–14 add to each lightweight partition the bulk nodes, and then conquer each partition separately, where we reuse all edges in TCO_B.

To tune the bulk-lightweight partitioning, DCBC introduces an additional parameter, the *bulk subscription threshold*, η, $0 < \eta \leq |T|$. Lines 1–2 rely on η to determine the subscription size of bulk subscribers: $B \supseteq \{v : |T(v)| > \eta\}$, where $T(v)$ denotes the topic set to which node v subscribes, and $|T(v)|$ represents the *subscription size* of node v. Lines 3–5 further extend B to make sure that it covers all topics in T.

Bulk subscribers tend to attract more links for constructing TCOs. Even without granting them a more prominent role, bulk subscribers still need to be able to sustain a high load in order to process publications on their subscribed topics. In practice, bulk subscribers are powerful machines whose users might be monitoring a large portion of the traffic.

Before considering how to choose key parameters for the algorithm (i.e., the implementation of `chooseEta`() and `chooseP`()), we first establish correctness, approximation ratio, and running time cost of the DCBC algorithm.

Lemma 12: Algorithm 6 outputs a TCO for the input (V, T, Int).

Following the notations for DC, for the dth partition with node set $BL_d \subseteq V$, we denote by E_{d_OPT} the minimum edge set that satisfies topic-connectivity for the instance $(BL_d, T, Int|_{BL_d})$ and by $E_{\mathrm{OPT}}|_{BL_d} (\subseteq E_{\mathrm{OPT}})$ the edge subset induced by BL_d. DCBC does not construct edges across different lightweight partitions because lightweight nodes always connect to bulk nodes to attain topic-connectivity. This might introduce additional suboptimality. Fortunately, this suboptimality is not so profound due to the small subscription size of lightweight nodes.

Lemma 13: Algorithm 6 has the following approximation ratio for the number of edges in the TCO: $|E_{\mathsf{DCBC}}|/|E_{\mathrm{OPT}}| = O(\eta \cdot \log(|V||T|))$.

The approximation ratio of DCBC given in Lemma 13 is better than that of DC given in Lemma 9. In particular, η is relatively small and can be considered as a constant factor for typical pub/sub workloads, while p is a higher-order variable in DC, which relies on the input instance.

If we define $\beta = |B|/|V|$, then Lemma 14 establishes the running time for the DCBC algorithm.

Lemma 14: The running time of Algorithm 6 is

$$\mathbb{T}_{\mathsf{DCBC}} = O\left(\frac{(\beta(p-1)+1)^2}{p}|V|^2|T|\right).$$

The selection of the bulk subscriber threshold η captures the tradeoff between node degree and running time. On the one hand, small threshold values result in treating the majority of nodes as bulk subscribers, which favors node degree over running time. As shown in Lemma 13, threshold η bounds the approximation ratio on the average node degree (regardless of the number of partitions p). When η is close to 0, all nodes are treated as bulk subscribers, and DCBC turns out to be equivalent to GM. On the other hand, large threshold values are favorable for the running time at the expense of an increased node degree. Keeping the threshold value low is important for reducing the risk of poor partitioning in DCBC, and thus providing a better bound on the average node degree of the output TCO. Fortunately, even relatively small threshold values result in small bulk subscriber sets for typical pub/sub workloads that follow the "Pareto 80–20" principle (see Section VII). Therefore, in our implementation of `chooseEta`() in Line 1 of Algorithm 6, we choose an η that causes $\leq$20% of all nodes to be considered as bulk subscribers. Since $\beta \leq 0.2$ in this case, this leads to a significantly improved running time of DCBC compared to that of GM according to Lemma 14.

To select of the number of partitions in `chooseP`(), we use Algorithm 5 that estimates $pn\text{-}size$, just as we do for DC. The considerations discussed in Section VI still apply, and Algorithm 5 provides a reasonable value of p^* for DCBC (see Section VIII-D).

VIII. Evaluation

A. Pub/Sub Workloads

We implement SM, DC, DCBC, and other auxiliary algorithms in Java. We use GM as a baseline because it produces

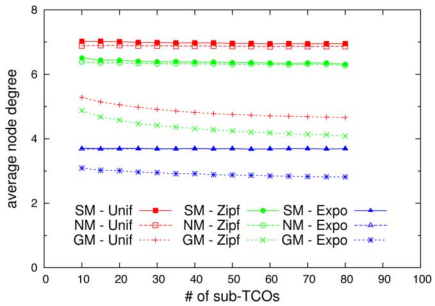

Fig. 4. Average degree—SM versus NM versus GM.

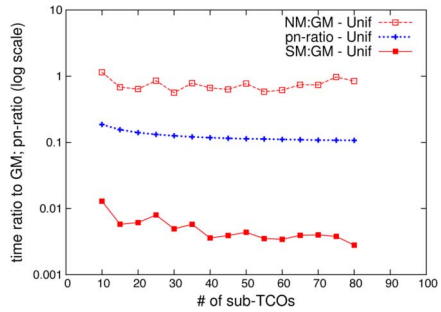

Fig. 5. Running time—SM versus NM versus GM.

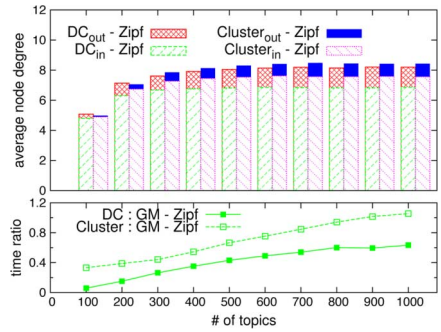

Fig. 6. Random partitioning versus clustering.

the lowest average node degree of all known polynomial algorithms. We denote by TCO_{ALG} the TCO produced by ALG, by d_{ALG} the average node degree in TCO_{ALG}, and by $\mathbb{T}_{\mathsf{ALG}}$ the running time of ALG, where ALG stands for any of the discussed algorithms. Given $TCO(V, T, Int, E)$, the *topic diameter* for $t \in T$ is the maximum shortest distance between any two nodes in $G(t)(V(t), E(t))$. We denote by $Diam_{\mathsf{ALG}}$ the maximum topic diameter across all topics in TCO_{ALG}.

We synthetically generate three types of topic popularities: uniform, Zipfian, and exponential. We also extract workloads from real-world social networks, i.e., Facebook and Twitter.

1) Synthetic Workloads: Our inputs have the following ranges: $|V| \in [1000, 10000]$, $|T| \in [100, 10000]$, and $\overline{|T(v)|} \in [50, 150]$, where we define the average node subscription size as $\overline{|T(v)|} = \sum_{v \in V} |T(v)|/|V|$. Each topic $t \in T$ is associated with probability $q(t)$, $\sum_t q(t) = 1$, such that each node subscribes to t with a probability $q(t)$. The value of $q(t)$ is distributed according to either a uniform, a Zipf (with $\alpha = 2.0$), or an exponential distribution, which we call Unif, Zipf, or Expo, for short. These distributions are representative of actual workloads used in industrial pub/sub systems today [16]. Stock-market monitoring engines use Expo for the study of stock popularity in the New York Stock Exchange [17]. Zipf faithfully describes the feed popularity distribution in RSS feeds [3]. Most experiment results presented in Section VIII are based on these synthetic workloads, unless stated otherwise.

2) Facebook Dataset: We use a public Facebook dataset [18], with over 3 million distinct user profiles and 28.3 million social relations as a second workload for our evaluations. In Facebook, when a user performs an activity (e.g., updating status information, sharing photos, or commenting on a blog), all the user's friends receive a notification. Therefore, we model each user, say Alice, as a topic, and all of Alice's friends as the respective subscribers. Likewise, the friend set of Alice forms her subscription set. Facebook relations are bidirectional: Friends in Alice's social graph subscribe to notifications from Alice, and vice versa, which we capture in our modeling.

3) Twitter Dataset: We also use a public Twitter dataset [19], containing 41.7 million distinct user profiles and 1.47 billion social followee–follower relations. Similarly to Facebook, we model users as topics and subscribers. However, relations in Twitter are unidirectional, i.e., Alice following Bob does not imply Bob to also follow Alice.

We extract the workloads from the original Facebook and Twitter social graphs with a methodology inspired from [20]. More specifically, starting with a random set of a few users as seeds, we traverse the social graph via breadth first search until we reach the targeted number of nodes, and our sample includes all edges among visited nodes. The size of our samples is 1 K and 10 K, i.e., $|V| \approx 1$K, $|T| \approx 1$K or $|V| \approx 10$K, $|T| \approx 10$K. We denote the instances of our samples by FB 1 K, FB 10 K, TW 1 K, and TW 10 K, respectively. Reference [21] uses these social network datasets and similar sampling methods for the pub/sub system evaluation.

B. *Star Merge Algorithm for* MinAvg-TCO-Join

We evaluate the output and performance of two incremental TCO-join algorithms: SM and NM. We compare these two to GM, which builds the TCO from scratch.

We generate sub-TCOs as follows: Each suboverlay TCO_d has $|V_d| = 100$ nodes ($d = 1, \ldots, p$, where p is the number of sub-TCOs), and all suboverlays share the same topic set T with $|T| = 100$. We construct these sub-TCOs using GM and feed them to the TCO-join algorithms as the input.

Fig. 4 shows that under different distributions, SM and NM produce similar TCOs with a slightly higher number of edges compared to GM. However, the increase in the average node degree is insignificant ($\leqslant 2.5$).

Although SM and NM produce quite close average node degrees (the difference is $\leqslant 0.15$), SM runs considerably faster than NM. Fig. 5 shows that, under Unif, it takes SM less than 1% of $\mathbb{T}_{\mathsf{GM}}$ on average, while NM costs as much as 75% of $\mathbb{T}_{\mathsf{GM}}$. We can explain this by the different $pn\text{-}size$'s between SM and NM. To illustrate the dependence, we normalize $pn\text{-}size$ by the total number of nodes $|V|$ and define the *potential neighbor ratio* as $pn\text{-}ratio = pn\text{-}size/|V|$. Note that the $pn\text{-}ratio$ that we present refers to $pn\text{-}ratio(V)$, the potential neighbor ratio for the node set V. SM achieves much lower $pn\text{-}ratio$, which is $\leqslant$18.6% in the worst case under Unif. NM's $pn\text{-}ratio$ is 100%, the same as GM.

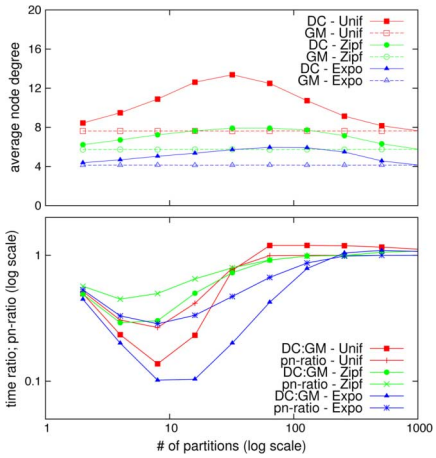

Fig. 7. DC versus GM wrt p.

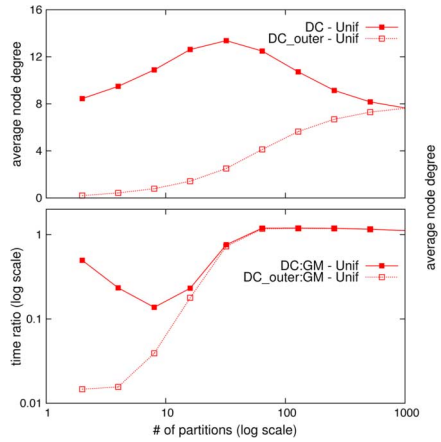

Fig. 8. Inner versus outer of DC wrt p.

TABLE I
RANDOM PARTITIONING UNDER UNIFORM DISTRIBUTION

average node degree, time ratio, size of star set				
	mean	variance	min	max
d_{DC}	6.50	0.000578	6.42	6.56
d_{GM}	5.09	0.000175	5.07	5.13
$\mathbb{T}_{\mathsf{DC}}/\mathbb{T}_{\mathsf{GM}}$	0.0634	0.000143	0.0205	0.150
$\tilde{k} = \overline{\lvert R_d \rvert}$	9.54	0.297	8	11

As the number of sub-TCOs that request to join increases, the average node degrees of the SM output remain steady while the running time ratio of SM to GM decreases considerably, which is caused by the decline in the *pn-ratio*. Notably, it only takes 0.28% of GM's running time for SM to incrementally construct a TCO with 8000 nodes. This further attests to SM's scalability with respect to the number of nodes.

C. Random Partitioning for Divide-and-Conquer

First, we evaluate the effects of random partitioning for the DC algorithm. We run DC 400 times for the same settings (namely, Unif, $|V| = 1000$, $|T| = 100$, and $|T(v)| = 20$) so that the only difference between different runs is due to the random node interest generation according to the given distribution parameters and due to random node partitioning. Table I reports the statistics pertaining to the average node degree, running time ratio, and the average size of a star set. In Table I, all the values are stable with negligible variance values across different experiments. Besides, the results validate our assumption that $\tilde{k} \ll k$. We conclude that when the number of partitions is reasonable, random partitioning is an efficient and robust way to implement the *divide* phase of DC and results in small star sets that could be computed efficiently, which is vital to the performance of DC.

Second, we study the impact of p on the output and performance of DC. Given input V, T, and Int, where $|T| = 200$, the DC algorithm is executed with all possible values of p ranging from 1 to $|V|$. We already know that DC exhibits identical behavior to GM at the two ends of p's range where $p = 1$ and $p = |V|$. Fig. 7 shows that as the number of partitions p grows from 1 to $|V|$, d_{DC} first increases gracefully, and then it starts to decrease until it becomes equal to d_{GM}. Note that d_{DC} never moves far from the horizon line of d_{GM}. Under the uniform distribution, the difference in average node degree between DC and GM is less than 6 even when d_{DC} reaches its peak. It is less than 3.5 when the running time of DC is minimal. On the other hand, DC's running time (ratio to GM) first slides down sharply and then climbs up as p increases. It touches the lowest point when p is around 10, which is relatively close to the left end of p's spectrum. Aligned with our proposed Algorithm 8 for choosing p^*, $\mathbb{T}_{\mathsf{DC}}$ (and consequently $\mathbb{T}_{\mathsf{DC}}/\mathbb{T}_{\mathsf{GM}}$) forms a similar U-shape as *pn-ratio*, and their lowest values are located close to each other. This further verifies our approach to choose p^* by minimizing *pn-size* (which is equivalent to minimizing *pn-ratio*).

Fig. 8 looks inside the operation of the DC algorithm and shows how the "inner" (*divide* and *conquer*) phases and the "outer" (*combine*) phase contribute to the studied metrics for different values of p. As shown in Fig. 8, increasing p leads to the increase of E_{outDC} and $\mathbb{T}_{\text{outDC}}$. However, we would like E_{inDC} and $\mathbb{T}_{\text{inDC}}$ to be the dominant terms because we expect the total runtime cost to be as low as possible. Besides, in some pub/sub systems (e.g., [20]), interpartition routing takes an order of magnitude more time than inside-partition message distribution, so small E_{outDC} is desired for efficient event dissemination. Thus, instead of having many small partitions, it is better to have fewer bigger and well-connected overlays. A well-selected p^* should be relatively small and stand far away from the endpoint $p = |V|$, which is again aligned with the value of p^* output by Algorithm 5.

Third, we compare two methods for the *divide* phase: random partitioning and clustering. We choose random partitioning for DC as discussed in Section V. As an alternative, we implement the k-means clustering algorithm with the following optimization techniques: We compute the Hamming distance between two nodes, i.e., $dist(v, w) = |\{t | Int(v, t) \neq Int(w, t)\}|$, and we use random initialization and the elbow method to determine the number of clusters.

Fig. 6 compares DC and divide-and-conquer with k-means clustering under Zipf—we collect similar results for other distributions. DC achieves better time efficiency in all experiments. Under some input instances, k-means clustering might yield slightly smaller average node degrees than DC because grouping similar nodes would cause fewer inner edges. However, as the number of topics increases, DC outperforms

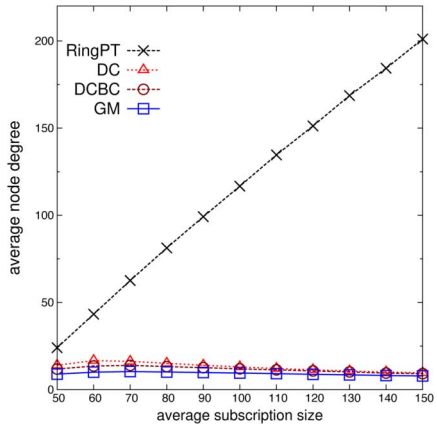

Fig. 9. DC versus DCBC versus GM versus RingPT.

TABLE II
EXPERIMENT RESULTS UNDER FB AND TW

d_{ALG}, $Diam_{\mathsf{ALG}}$	FB1K	FB10K	TW1K	TW10K
DC	4.40, 10	8.47, 18	8.61, 14	16.63, 24
DCBC	2.94, 6	7.43, 8	4.98, 8	11.55, 7
GM	2.74, 8	5.48, 12	4.74, 11	8.33, 16

k-means in the average node degree because the placement of *bulk nodes* becomes increasingly important. The k-means clustering tends to isolate *bulk nodes* from *lightweight nodes*. The lack of bulk nodes results in a significant loss of correlation in the lightweight node partition, which requires many more links to build the sub-TCO.

Overall, these experiments validate our choice of random partitioning for the divide-and-conquer algorithm. We opt not to explore clustering techniques in greater depth, e.g., by using other distance metrics or applying more sophisticated clustering algorithms. There is not much potential for improvement by using clustering in the *divide* phase because divide-and-conquer with random partitioning already achieves good performance both theoretically and empirically (see Section VIII-D).

D. DC *and* DCBC *for* MinAvg-TCO

This experiment evaluates the performance and scalability of DC and DCBC with respect to different input variables. Given any MinAvg-TCO instance, we determine the algorithm parameters based on the analysis in Sections VI and VII: 1) we select the number of partitions p for both DC and DCBC by Algorithm 5; 2) we choose the threshold η for DCBC s.t. $\leq$20% of all nodes to be treated as bulk subscribers.

1) Running on Social Network Workloads: We execute DC, DCBC, and GM under real-world social network workloads as described in Section VIII-A. Table II shows that $d_{\mathsf{GM}} < d_{\mathsf{DCBC}} < d_{\mathsf{DC}}$ and $Diam_{\mathsf{DCBC}} < Diam_{\mathsf{GM}} < Diam_{\mathsf{DC}}$ under all inputs. As the instances scale up, DCBC shows more substantial advantages over DC in both average node degrees and topic diameters. Under TW10K, $d_{\mathsf{DCBC}} - d_{\mathsf{GM}} = 3.22$, but d_{DC} is about twice of d_{GM}; $Diam_{\mathsf{DCBC}} < (1/2)Diam_{\mathsf{GM}} < (1/3)Diam_{\mathsf{DC}}$. These benefits of DCBC stem from our special treatment for *bulk nodes*. First, the replicated bulk nodes contribute to minimize the number of edges: The replication of bulk nodes in each partition greatly helps DCBC preserve the subscription correlation, which might be reduced in the random partitioning phase of DC. Second, bulk nodes serve as the "connecting spots" for the TCO, which are beneficial to diminish the topic diameters.

2) Comparison to Ring-Per-Topic: This experiment compares the average node degrees of DC, DCBC, GM, and *Ring-Per-Topic* (RingPT). RingPT mimics the common practice of building a separate overlay for each topic (usually a tree, but we use a ring that has the same average node degree) [9], [11], [22], [23]. RingPT forms a ring for all the nodes interested in the same topic, and rings for different topics are merged into a single TCO. Fig. 9 shows experiments under Unif. First, the average node degrees output by DC and DCBC are quite close to those of GM, the average difference between GM and DCBC is 2.45, and DCBC is slightly better than DC (by 1.35 on average). As the subscription size increases, d_{DCBC}, d_{DC}, and d_{GM} decrease, and the differences (i.e., $d_{\mathsf{DCBC}} - d_{\mathsf{GM}}$ and $d_{\mathsf{DC}} - d_{\mathsf{GM}}$) shrink. Second, the average node degree of RingPT grows almost linearly with the subscription size; d_{RingPT} roughly equals twice the subscription size, which is over 10 times the value of d_{DCBC} (or d_{DC}) on average. This demonstrates the scalability of DC and DCBC as compared to RingPT.

3) Impact of $|V|$: Fig. 10(a) compares the performance of DC, DCBC, and GM with regard to the number of nodes under Unif. DC and DCBC output similar average node degrees, which are slightly higher than d_{GM}. Both DC and DCBC run considerably faster than GM. DCBC obtains a marginally better average node degree as compared to DC, i.e., $d_{\mathsf{DCBC}} \approx d_{\mathsf{DC}} - 0.99$ on average, because DCBC uses a more refined partitioning mechanism. As the number of nodes scales up, this marginal improvement is more profound: Both d_{GM} and d_{DCBC} decrease, and d_{DCBC} becomes closer to d_{GM}. However, d_{DC} stands relatively stable, and thus $d_{\mathsf{DC}} - d_{\mathsf{DCBC}}$ is increasing. At $|V| = 10\,000$, $d_{\mathsf{GM}} = 4.18$, $d_{\mathsf{DCBC}} = 4.60$, and $d_{\mathsf{DC}} = 10.0$. However, DCBC has an extra runtime cost because of the additional calculation imposed by replicating the bulk nodes: Under Unif, $\mathbb{T}_{\mathsf{DCBC}}$ is $164\% \cdot \mathbb{T}_{\mathsf{DC}}$ and $2.6\% \cdot \mathbb{T}_{\mathsf{GM}}$ on average. DCBC outperforms both DC and GM in terms of topic diameters. As the number of nodes increases, $Diam_{\mathsf{DCBC}}$ is decreasing but $Diam_{\mathsf{DC}}$ is increasing; at $|V| = 10\,000$, $Diam_{\mathsf{DC}} > 3 \cdot Diam_{\mathsf{DCBC}}$. The root cause lies in the subscription correlation in the partitions. Increasing the number of nodes increases the correlation—the node has a higher chance to find a neighbor with a larger interest overlap. DCBC preserves the subscription correlation by cloning bulk nodes in each partition, and thus both average node degrees and topic diameters decrease thanks to the increased correlation. However, random partitioning in DC leads to the loss of correlation in some division, especially for a large node set. The size of each random partition in DC is more or less the same, so both average node degree and topic diameters for each partition stay relatively stable. As the number of nodes increases, DC randomly divides the node set into more partitions, the interpartition links becomes the major factor for the quality of the output TCO, and both average node degrees and topic diameters tend to increase.

4) Impact of $|T|$: Fig. 10(b) depicts how DC and DCBC perform compared to GM as the number of topics scales up under Unif. First, the average node degrees of all algorithms

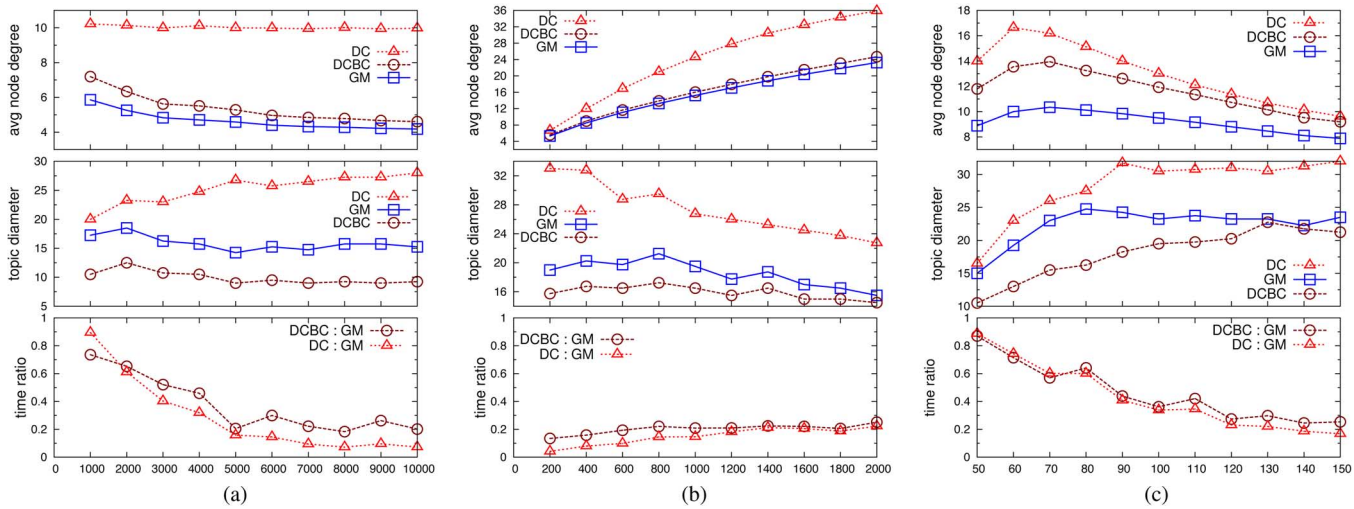

Fig. 10. DCBC versus DC versus GM under Unif. (a) Number of nodes $|V|$. (b) Number of topic $|T|$. (c) Average subscription size.

increase with the number of topics. This is because increasing the number of topics leads to reduced correlation among the nodes. The gap in the average node degree between DC and GM becomes more noticeable as the number of topics increases: $d_{\mathsf{DC}} - d_{\mathsf{GM}} = 14.74$ when the number of topics reaches 2000, which is an increase of over 69.0%. DCBC achieves a lower average node degree as compared to DC, and the difference against GM is insignificant, i.e., $d_{\mathsf{DCBC}} - d_{\mathsf{GM}} = 1.94$ when $|T| = 2000$. This is in line with Lemma 13, which shows the improvement in the average node degree produced by DCBC as compared to that of DC. The results suggest that bulk nodes play a more important role in capturing the correlation among a node set when there is a large number of topics, and that the placement of bulk nodes has a more profound impact on the average node degree of the divide-and-conquer algorithms. Second, DCBC outputs the lowest topic diameters among all algorithms: $Diam_{\mathsf{DCBC}} = 15.9$, $Diam_{\mathsf{GM}} = 18.5$, and $Diam_{\mathsf{DC}} = 22.3$ on average under Unif. The topic diameters decrease as the number of topics increases since the overlays have more edges. Third, both DCBC and DC run more efficiently than GM. DCBC exhibits a higher speedup as compared to DC when the number of topics increases. The time ratios of $\mathbb{T}_{\mathsf{DCBC}}/\mathbb{T}_{\mathsf{GM}}$ and $\mathbb{T}_{\mathsf{DC}}/\mathbb{T}_{\mathsf{GM}}$ increase at a low pace proportionally to the number of topics. Still, $\mathbb{T}_{\mathsf{DCBC}} \leq 5.7\% \cdot \mathbb{T}_{\mathsf{GM}}$ and $\mathbb{T}_{\mathsf{DCBC}} \leq 10.5\% \cdot \mathbb{T}_{\mathsf{GM}}$ in the worst case when the number of topics is as high as 2000.

5) Impact of subscription size: Fig. 10(c) depicts how the subscription size affects the DC and DCBC algorithms. We set $|V| = 1000$, $|T| = 400$, and $\overline{|T(v)|}$ varies from 50 to 150. Under Unif, as the subscription size increases, both d_{DC} and d_{DCBC} decrease, and the gaps of ($d_{\mathsf{DC}} - d_{\mathsf{GM}}$ and $d_{\mathsf{DCBC}} - d_{\mathsf{GM}}$) shrink. We can explain this by the increase in topic correlation within the node set when the subscription size grows. Like GM, both DC and DCBC also choose each link in a greedy manner (but from a smaller set), so that the selected edge is more likely to connect a higher number of TC-components since the nodes share more common interests.

Topic diameters increase as the subscription size increases since the number of edges increases. $Diam_{\mathsf{DCBC}}$ is always lower than $Diam_{\mathsf{GM}}$ and $Diam_{\mathsf{DC}}$, and $Diam_{\mathsf{DCBC}} \approx 0.64 \cdot Diam_{\mathsf{DC}}$ on average.

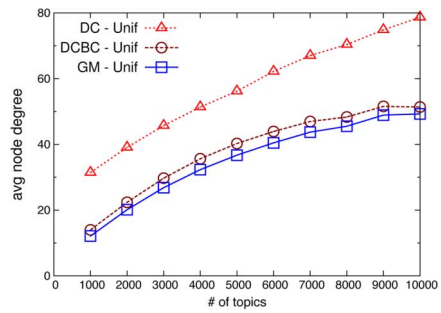

Fig. 11. DCBC versus DC as $|T|$ scales up.

The running time ratios of DC and DCBC compared to GM further improve with the increase in the average subscription size. Although more updates for an edge addition would be incurred, DC and DCBC significantly reduce the time for each update compared to GM since each edge update in DC and DCBC only involves a small subset of all nodes.

Similarly to Fig. 10(a) and (b), Fig. 10(c) shows that DCBC treads the balance between the quality of the output and time efficiency, i.e., both d_{DCBC} and $\mathbb{T}_{\mathsf{DCBC}}$ lie between those of the DC and GM algorithms. DCBC marginally outperforms DC in terms of average node degrees, i.e., the difference $d_{\mathsf{DC}} - d_{\mathsf{DCBC}}$ is 0.97 on average. However, this benefit comes at the expense of additional runtime, i.e., DCBC takes 4.4% more time as compared to DC.

In summary, Fig. 10 shows the following: 1) Both DC and DCBC produce TCOs with low average node degree at significant speedup as compared to GM under a variety of typical pub/sub workloads. 2) DCBC provides a more fine-tuned mechanism to seek the balance between the average node degree and the running time, and this benefit is especially important when the instance scales up; as shown in Fig. 11, as the number of topics increases from 1000 to 10 000, the suboptimality of DC becomes noticeable, and the advantage of DCBC is significant: $d_{\mathsf{DC}} - d_{\mathsf{DCBC}} \approx 25$ at $|T| = 10\,000$. 3) DCBC significantly improves topic diameters as compared to DC.

IX. Related Work

In order to improve distributed pub/sub system performance and scalability, two directions have crystallized in the literature: 1) the design of routing protocols such that publications and subscriptions are sent in a most efficient way across the overlay network (see [10], [24]–[26]); and 2) the construction of the overlay topology such that network traffic is minimized (e.g., [4]–[7], [22], [27]). A number of recent approaches combine both directions [20], [23]. This paper focuses on the second direction.

Topic-connectivity is a required property in approaches such as [9], [11], [16], and [22], and is an implicit, not directly specified, requirement in approaches such as [7], [10], [20], and [27]–[29], which all aim to diminish the number of unrelated intermediate overlay hops in one way or another.

Baldoni *et al.* [7] proposed a self-organizing algorithm to connect brokers matching similar events, which aimed to improve the overall system performance by reducing the overlay hops for event routing. The authors pointed out that an overlay should be constructed in a way such that brokers sharing interests are closer to each other. The approach targeted at content-based pub/sub and acyclic topologies, which is different from this work. Chockler *et al.* [16] showed empirically that on many practical workloads exhibiting well-correlated subscription patterns, and a simple distributed heuristic could effectively reduce the average node degree for the TCO.

The theoretical formulation of pub/sub overlay design originated in [4]. Chockler *et al.* [4] proposed the MinAvg-TCO problem of constructing a TCO with minimum edges. They proved the NP-completeness of MinAvg-TCO and presented the GM algorithm that achieves a logarithmic approximation ratio for the average node degree. References [5] and [30] pointed out that GM might produce a TCO with unbalanced node degrees by effectively creating hotspot nodes with a high node degree. Reference [30] proposed the problem of minimizing the maximum node degree in the TCO. Furthermore, [5] attempted to minimize both the maximum and average node degrees of the TCO.

The ideas underlying [4], [5], and [30] are inspired by the standard greedy algorithm design principles and rooted in the analysis for the classical *minimum set cover* problem. To the best of our knowledge, we are among the first to apply the divide-and-conquer approach to pub/sub overlay design.

We also designed divide-and-conquer algorithms to reduce the maximum node degree of the TCO in [31], which targets at a fundamentally different problem from the work presented in this paper. Our algorithms in [31] do not guarantee any theoretical bound on the total number of edges, and the output TCOs may have significantly higher average node degrees.

Like most work in the area of pub/sub overlay network design (e.g., [4], [5], [7], [9], [10], [16], [21], [27], [28], [30], and [32]), the TCO model and related approaches are oblivious to the low-level network infrastructure. First, incorporating topology awareness is a performance optimization, but is not required for correctness. Second, topology plays a less important role in the targeted environment, where all nodes reside in one large-scale data center. As a result, the TCO model does not consider locality in the underlying network and implicitly assumes that edges between any two nodes are equivalent. The assumption of edge equivalence is reasonable for a pub/sub system in a data center but does not hold for Internet-scale applications that are deployed across multiple data centers. Unfortunately, it has not been widely studied (or hardly at all considered) what effects the underlying (physical) network topology may induce on the pub/sub overlay.

Indeed, building a topology-aware pub/sub overlay is an important and challenging problem; our work (and many other TCO-based approaches, e.g., [4], [5], [16], and [30]) serve as a solid bedrock for this promising direction. To address this problem, the TCO framework could embrace several existing techniques. First, we could reflect network structure and physical locality by measuring node *distances* based on some network-level metrics [7], [11], [20], [33]–[37]. It would be natural to introduce edge weights into the TCO model: Intuitively, we may hope to cluster nodes with similar interests while preserving locality in terms of small overlay edge weights. We could apply the divide-and-conquer strategy, but instead of random partitioning, we need more intelligent mechanisms with respect to topology, e.g., the binning scheme with predefined landmark nodes [33], [36], [38] or clustering of nodes with closer identifiers that encode network structure and geographical positions [27], [37], [39]. Second, besides the TCO, we could build an additional overlay in which each node maintains its proximity-based neighbors [11], [35], [37], [39], [40]. Then, we could design pub/sub routing on the combined overlay. For example, we could integrate small-world networks [13], [41] into the TCO: Build a sub-TCO for each region (e.g., a data center) with only local links and connect different sub-TCOs via long-range links [20]. Third, we could design proximity routing [33], [42] on top of the pub/sub TCO. This approach would construct the overlay regardless of physical topology. However, we would have to develop efficient routing protocols that forward messages to the topologically closest nodes among the next hop candidates in the routing table.

X. Conclusion

We introduce the MinAvg-TCO-Join problem that attempts to add the minimum number of links to join a number of sub-TCOs into a single TCO. MinAvg-TCO-Join is NP-complete. We develop the Star Merge (SM) algorithm that efficiently constructs the TCO with a logarithmic approximation ratio.

Our approach inspires the divide-and-conquer strategy that gives rise to a number of new algorithms for the original MinAvg-TCO problem. Theoretical analysis shows that with a reasonable partitioning of the node set, our divide-and-conquer algorithms are able to construct high-quality TCOs significantly faster than earlier alternatives. We motivate novel techniques including random partitioning and bulk-lightweight partitioning that capture the tradeoffs between the runtime cost and the quality of the output TCO. We also provide a numerical method to determine the number of partitions as the parameter for any input instance.

Our comprehensive experimental analysis demonstrates the performance benefits for our family of divide-and-conquer algorithms. The analysis also validates important design decisions for our algorithms, such as random partitioning, bulk-lightweight partitioning, determining the number of partitions,

and selecting the bulk subscriber threshold. We show that divide-and-conquer algorithms significantly reduce runtime cost to construct TCOs with guaranteed qualities, e.g., average node degrees and topic diameters.

REFERENCES

[1] J. Reumann, "Pub/sub at Google," Oslo, Norway, Aug. 2009, lecture at CANOE Summer School.

[2] Tibco Software, Inc., Palo Alto, CA, USA, "Tibco Rendezvous," [Online]. Available: http:\www.tibco.com

[3] H. Liu, V. Ramasubramanian, and E. G. Sirer, "Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews," in *Proc. IMC*, 2005, p. 3.

[4] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Constructing scalable overlays for pub-sub with many topics," in *Proc. PODC*, 2007, pp. 109–118.

[5] M. Onus and A. W. Richa, "Parameterized maximum and average degree approximation in topic-based publish-subscribe overlay network design," in *Proc. IEEE ICDCS*, 2010, pp. 644–652.

[6] M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann, "Self-organizing broker topologies for publish/subscribe systems," in *Proc. SAC*, 2007, pp. 543–550.

[7] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito, "Efficient publish/subscribe through a self-organizing broker overlay and its application to SIENA," *Comput. J.*, vol. 50, no. 4, pp. 444–459, 2007.

[8] S. Voulgaris, E. Riviére, A.-M. Kermarrec, and M. V. Steen, "Sub-2-Sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks," in *Proc. IPTPS*, 2006.

[9] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni, "TERA: Topic-based event routing for peer-to-peer architectures," in *Proc. DEBS*, 2007, pp. 2–13.

[10] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 8, pp. 1489–1499, Oct. 2002.

[11] J. A. Patel, E. Riviére, I. Gupta, and A.-M. Kermarrec, "Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems," *Comput. Netw.*, vol. 53, no. 13, pp. 2304–2320, 2009.

[12] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the evolution of peer-to-peer systems," in *Proc. PODC*, 2002, pp. 233–242.

[13] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *Proc. STOC*, 2000, pp. 163–170.

[14] "Google cluster data," [Online]. Available: http://code.google.com/p/googleclusterdata/

[15] C. Chen, H.-A. Jacobsen, and R. Vitenberg, "Star merge and divide-and-conquer algorithms for publish/subscribe topic-connected overlay design," Tech. Rep., 2010 [Online]. Available: http://msrg.org/papers/TRCJV-DC-2010

[16] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: A scalable interest-aware overlay for topic-based pub/sub communication," in *Proc. DEBS*, 2007, pp. 14–25.

[17] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky, "Hierarchical clustering of message flows in a multicast data dissemination system," in *Proc. IASTED PDCS*, 2005, pp. 320–326.

[18] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *Proc. EuroSys*, 2009, pp. 205–218.

[19] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?," in *Proc. WWW*, 2010, pp. 591–600.

[20] F. Rahimian, T. Le Nguyen Huu, and S. Girdzijauskas, "Locality-awareness in a peer-to-peer publish/subscribe network," in *Proc. DAIS*, 2012, pp. 45–58.

[21] V. Setty *et al.*, "The hidden pub/sub of Spotify," in *Proc. DEBS*, 2013, pp. 231–240.

[22] M. Matos, P. Felber, R. Oliveira, J. O. Pereira, and E. Riviere, "Scaling up publish/subscribe overlays using interest correlation for link sharing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 12, pp. 2462–2471, Dec. 2013.

[23] V. Setty, M. van Steen, R. Vitenberg, and S. Voulgaris, "PolderCast: Fast, robust, scalable architecture for P2P topic-based pub/sub," in *Proc. Middleware*, 2012, pp. 271–291.

[24] D. Tam, R. Azimi, and H.-A. Jacobsen, "Building content-based publish/subscribe systems with distributed hash tables," in *Proc. DBISP2P*, 2003, pp. 138–152.

[25] G. Li, V. Muthusamy, and H.-A. Jacobsen, "Adaptive content-based routing in general overlay topologies," in *Proc. Middleware*, 2008, pp. 1–21.

[26] F. Araujo, L. Rodrigues, N. Carvalho, and N. Carvalho, "Scalable QoS-based event routing in publish-subscribe systems," in *Proc. IEEE NCA*, 2005, pp. 101–108.

[27] S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed, "Magnet: Practical subscription clustering for Internet-scale publish/subscribe," in *Proc. DEBS*, 2010, pp. 172–183.

[28] E. Baehni, P. Eugster, and R. Guerraoui, "Data-aware multicast," in *Proc. DSN*, 2004, p. 233.

[29] R. Chand and P. Felber, "Semantic peer-to-peer overlays for publish/subscribe networks," in *Proc. EUROPAR*, 2005, pp. 1194–1204.

[30] M. Onus and A. W. Richa, "Minimum maximum degree publish-subscribe overlay network design," in *Proc. IEEE INFOCOM*, 2009, pp. 882–890.

[31] C. Chen, R. Vitenberg, and H.-A. Jacobsen, "Scaling construction of low fan-out overlays for topic-based publish/subscribe systems," in *Proc. ICDCS*, 2011, pp. 225–236.

[32] F. Rahimian, S. Girdzijauskas, A. H. Payberah, and S. Haridi, "Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks," in *Proc. IEEE IPDPS*, 2011, pp. 746–757.

[33] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. SIGCOMM*, 2001, pp. 161–172.

[34] M. A. Tariq, B. Koldehofe, and K. Rothermel, "Efficient content-based routing with network topology inference," in *Proc. DEBS*, 2013, pp. 51–62.

[35] M. Castro, P. Druschel, Y. C. Hu, and A. I. T. Rowstron, "Topology-aware routing in structured peer-to-peer overlay networks," in *Proc. Future Directions Distrib. Comput.*, 2003, pp. 103–107.

[36] Z. Xu, C. Tang, and Z. Zhang, "Building topology-aware overlays using global soft-state," in *Proc. ICDCS*, 2003, pp. 500–508.

[37] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "SkipNet: A scalable overlay network with practical locality properties," in *Proc. USITS*, 2003, vol. 4, p. 9.

[38] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proc. IEEE INFOCOM*, 2002, vol. 3, pp. 1190–1199.

[39] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, routing for large-scale peer-to-peer systems," in *Proc. Middleware*, 2001, pp. 329–350.

[40] I. Aekaterinidis and P. Triantafillou, "PastryStrings: A comprehensive content-based publish/subscribe DHT network," in *Proc. ICDCS*, 2006, p. 23.

[41] A. Slivkins, "Towards fast decentralized construction of locality-aware overlay networks," in *Proc. PODC*, 2007, pp. 89–98.

[42] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. SIGCOMM*, 2001, pp. 149–160.

[43] S. Chakravarty and A. Shekhawat, "Parallel and serial heuristics for the minimum set cover problem," *J. Supercomput.*, vol. 5, no. 4, pp. 331–345, 1992.

Chen Chen (M'10) received the B.S. and M.S. degrees in computer science from Tsinghua University, Beijing, China, in 2004 and 2007, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada, in 2014.

His Ph.D. research focuses on the design of overlay infrastructure and routing protocols for distributed systems, especially publish/subscribe.

Hans-Arno Jacobsen (S'98–A'00–M'03–SM'08) holds the Bell University Laboratories Chair in Software, and he is a faculty member with the Department of Electrical and Computer Engineering and the Department of Computer Science, University of Toronto, Toronto, ON, Canada, where he leads the Middleware Systems Research Group. His principal areas of research include the design and the development of middleware systems, distributed systems, and information systems. His current research focuses on publish/subscribe, content-based routing, event processing, and aspect orientation.

Roman Vitenberg (M'07) is a faculty member with the Department of Informatics, University of Oslo, Oslo, Norway. His research interests lie broadly in the areas of distributed applications, middleware, and algorithms; they span modeling, design, analysis, software engineering, implementation, and performance evaluation. His specific research focus is on large-scale communication, mobile and dynamic environments, object-oriented and component-based platforms, distributed event-based systems, consistency models, and fault-tolerant distributed computing.