2010 International Conference on Distributed Computing Systems

Divide and Conquer Algorithms for Publish/Subscribe Overlay Design

Chen Chen
Department of Electrical and Computer Engineering
University of Toronto
chen@msrg.utoronto.ca

Hans-Arno Jacobsen
Department of Electrical and Computer Engineering
Department of Computer Science
University of Toronto
jacobsen@eecg.toronto.edu

Roman Vitenberg
Department of Informatics
University of Oslo, Norway
romanvi@ifi.uio.no

***Abstract*—Overlay network design for topic-based publish/subscribe systems is of primary importance because the overlay directly impacts the system's performance. Determining a topic-connected overlay, in which for every topic the graph induced by nodes interested in the topic is connected, is a fundamental problem.**

Existing algorithms for this problem suffer from three key drawbacks: (1) prohibitively high running time cost, (2) requirement of full system knowledge and centralized operation, and (3) constructing overlay from scratch. From a practical point of view, these are all significant limitations.

To address these concerns, in this paper, we develop novel algorithms that efficiently solve the problem of dynamically joining two or more topic-connected overlays. Inspired from the divide-and-conquer character of our approach, we derive an algorithm that solves the original problem at a fraction (up to 1.7%) of the running time cost of alternative solutions, but at the expense of an empirically insignificant increase in the average node degree.

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. Many applications report benefits from using this form of interaction, such as application integration [1], financial data dissemination [2], RSS feed distribution and filtering [3], [4], and business process management [5]. As a result, many industry standards have adopted pub/sub as part of their interfaces. Examples of such standards included WS Notifications, WS Eventing, OMG's Real-time Data Dissemination Service, and the Active Message Queuing Protocol.

In pub/sub, subscribers convey their interests in receiving messages and publishers disseminate publication messages. The language and data model to subscribe and publish vary among systems. In this paper, we focus on the topic-based pub/sub model. In a topic-based system, publication messages are associated with topics and subscribers register their interests in receiving all messages published to topics of interest. Many commercial systems are based on this design. For example, TIBCO RV [2] has been used extensively for market data feed dissemination and Google's GooPS [1] constitutes the distributed message exchange for Web-based applications operating worldwide.

In a distributed pub/sub system, so called pub/sub brokers, often connected in a federated manner as an application-level overlay network, efficiently route publication messages from data sources to sinks. The distributed design was introduced to address pub/sub system scalability. The overlay of a pub/sub system directly impacts the system's performance and the message routing cost. Constructing a high-quality broker overlay is a key challenge and fundamental problem for distributed pub/sub systems that has received attention both in industry [1] and academia [6]–[9].

Chockler *et al.* [6] introduced the notion of *topic-connectivity*, which informally speaking means that all nodes interested in the same topic are organized in a connected dissemination sub-overlay. This notion abstracts from data sources and sinks, and equates the client roles with that of a pub/sub broker. Also, publisher and subscriber client roles are equated and simply referred to as nodes interested in topics. This abstraction simplifies the presentation for a theoretical and algorithmic treatment of the problem, while fully preserving its practical characters. We adopt these notions in this paper. Also, from here on forward, we refer to a pub/sub broker simply as a node.

A number of existing approaches to broker overlay construction build separate dissemination overlays on a per-topic (or, more generally, per-interest) basis [10]–[12], thereby attaining topic-connectivity. While such construction methods are efficient, the resulting overlays exhibit prohibitively high average node degrees.

In order to address this issue, Chockler *et al.* posed the problem of determining the topic-connected overlay with the minimum average node degree, proved the problem is NP-complete, and developed a greedy algorithm as solution [6]. Other variations of this problem have also been considered [7]. While these approaches give rise to overlays with quite small node degrees, their running time cost is significant [6], [7]. Furthermore, they require global knowledge of all nodes participating in the pub/sub system as well as their interests. Their solutions are based on centralized algorithms that do not lend themselves to decentralized execution and do not allow for incremental and dynamic changes to nodes, interests and topics without requiring to run the overlay construction from scratch.

1063-6927/10 $26.00 © 2010 IEEE
DOI 10.1109/ICDCS.2010.87

From a practical point of view, these constraints are not satisfactory as system failures inevitably lead to network partitions, which could motivate the need for joining existing and correctly functioning sub-overlays of nodes. Similarly, partitions may be the result of administrative or scheduled maintenance tasks that affect part of the system. A tear-down of all connections and renewed setup of the overlay would be excessively expensive and grow with the size of the overlay. Solutions that do not support dynamics of change to topics, interests, and nodes exhibit the same shortcomings. In practice, especially topics and interests change a lot. While global system knowledge is not an absolute requirement as large organizations often employ various forms of highly available, centralized directory services to reliably track configuration information, a solution to the overlay construction problem that can get by with partial system information may operate faster. Similar concerns apply to the need for decentralizing a solution.

To address these concerns, we propose a number of novel techniques to solve the overlay construction problem. First, we formulate the topic-connected overlay join problem and establish its complexity properties before proposing algorithms for solving the problem. Given two or more topic-connected overlays, we aim to construct a single topic-connected overlay that includes all nodes of the sub-overlays and respects all interest relations. We establish that solving this problem by adding the minimal number of edges is NP-complete. We develop and analyze algorithms for solving this problem and empirically evaluate their performances. Based on our solution, we develop a divide-and-conquer (DC) algorithm that is based on the intuition of dividing an overlay into smaller pieces, solving each piece separately and then recombining the separate solutions into a single, global solution. We show and analyze several key advantages of this divide-and-conquer solution for the original topic-connected overlay construction problem.

Both theoretical analysis and experimental evaluations show that it takes the DC algorithm significantly less time to construct an overlay from scratch than previous algorithms, at the expense of an insignificant increase in the node degree of the resulting overlay. In our experiments, based on typical workloads, we show that the DC algorithm runs at up to 1.7% of the running time cost of an alternative solution constructing overlays with an on average 2.12 larger node degree. In addition, the DC algorithm only requires partial information about nodes, topics and interests. Moreover, the DC algorithm is better able to accommodate dynamic node joins that tend to affect small portions of the overlay.

This paper is organized as follows. In Section II we put our work in the context of related approaches. In Section III we introduce some notations and background information required to keep this paper self-contained. In Section IV we formally introduce the topic-connected overlay join problem. In Section V we discuss and analyze novel solutions to this problem. In Section VI we generalize our approach resulting in the design of a divide-and-conquer algorithm that substantially outperforms earlier solutions. In Section VII we elaborate on several characteristics and optimizations of the new algorithm. In Section VIII we present a thorough experimental analysis of all algorithms developed in this paper, including a comparison of our work to earlier approaches.

II. 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 so that publications and subscriptions are sent most efficiently across the overlay network and (2) the construction of the overlay topology such that network traffic is minimized. Much attention has focused on the first direction [12]–[15] with some recent work addressing the second direction [6]–[9]. The approaches discussed in this paper are part of the second direction.

In [12], [13] efficient protocols were proposed for building pub/sub dissemination trees on top of peer-to-peer substrates. Other than the work in this paper, these approaches assumed an infrastructure-less peer-to-peer context with greatly different constraints from what is the starting point for our work.

Li *et al.* [14] developed algorithms that support general overlays, especially cyclic overlays to introduce redundancy that could be exploited in case of failure, load, or congestion of overlay nodes. IndiQoS [15] supports QoS for rendezvous-based pub/sub approaches. QoS information is collected among neighboring nodes and QoS-capable paths are chosen for message routing. Both approaches target the very different content-based pub/sub model, where it is common place that subscriber interests are fine-grained going beyond the limits of topics. The approach of using an interest matrix employed in this work does not directly extend to content-based approaches.

Topic-connectivity is a required property in approaches such as [11], [16] and is an implicit, not directly specified, requirement in approaches such as [9], [12], [17], [18], which all aim to diminish the number of unrelated intermediate overlay hops in one way or another.

Baldoni *et al.* [9] proposed a self-organizing algorithm to connect brokers matching similar events, which aims 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 should be closer to each other. The approach is content-based, which is a different model from the one underlying the work in this paper. Chockler *et al.* [16] showed empirically that on many practical workloads exhibiting well-correlated subscription patterns, a simple distributed heuristic could be effective for constructing topic-connected overlays with a small average node degree. [16] emphasized on empirical results, but it serves as a good base for further theoretical exploration.

The theoretical formulation of the MinAvg-TCO problem originated in [6]. The problem is the construction of a topic-connected overlay network with minimal edges. The authors proved the problem's NP-completeness and presented a greedy

algorithm for solving the problem. Their algorithm is able to determine an overlay with a logarithmic approximation ratio on the minimum average node degree by using full topology and topic interest information. As the authors pointed out, the GM algorithm might produce an overlay with very uneven node degrees, and thus result in some nodes with unreasonably high node degrees. Onus and Richa [7] defined the MinMax-TCO problem which aims to minimize the maximum degree of the overlay network.

Both approaches [6], [7] borrow ideas from standard greedy algorithm design and proof techniques from the classical minimum set cover problem [19]. To the best of our knowledge, applying divide-and-conquer to do overlay construction has not been directly addressed in prior work.

III. Background

In this section we present some 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. The interest function Int is defined as $Int : V \times T \rightarrow \{true, false\}$. Since the domain of the interest function is a Cartesian product, we can refer to this function as an interest matrix. Given an interest function Int, we say that a node v is interested in some topic t if and only if $Int(v,t) = true$.

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 $t \in T$, we say that 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 called *topic-connected* if for each topic $t \in T$, the subgraph G_t of G induced by t contains at most one connected component. Figure 1 gives an example. A *topic-connected overlay* (TCO) is denoted as $TCO(V,T,Int,E)$.

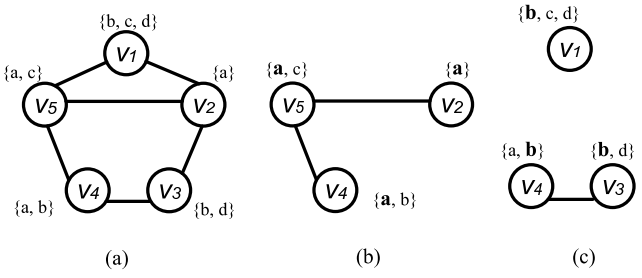

Fig. 1. (a) An overlay G. (b) Subgraph G_a induced by topic a is topic-connected. (c) Subgraph G_b induced by topic b is not topic-connected.

Chockler *et al.* [6] introduced the parametrized family of *Scalable Overlay Construction* (SOC) design problems for pub/sub that captures the trade-off between the overlay scalability and the cost of message dissemination. Their work focused on the MinAvg-TCO problem for minimizing the number of links needed to create a TCO for a given interest function. A formal definition of the problem is as follows.

Definition 1. *MinAvg-TCO(V,T,Int): Given a set of nodes V, a set of topics T, and the interest function Int, construct a topic-connected overlay network $TCO(V,T,Int,E)$ which has the least possible total number of edges (i.e., the least possible average node degree).*

Algorithm 1 Greedy Merge algorithm for MinAvg-TCO.

greedyMerge(V, T, Int)
Input: V, T, Int
Output: A topic-connected overlay $TCO(V,T,Int,E_{GM})$

```
1: for all v ∈ V do
2:     for all t ∈ T such that Int(v,t) do
3:         Nodes[v][t] ← {v}
4: for all e = (v,w) do
5:     contrib ← |{t ∈ T | Int(v,t) ∧ Int(w,t)}|
6:     if contrib > 0 then
7:         add e to LinkContrib[contrib]
8: E_GM ← constructOverlayEdges(V, T, Int, Nodes, LinkContrib)
9: return TCO(V, T, Int, E_GM)
```

Algorithm 2 Overlay construction for the GM algorithm.

constructOverlayEdges(V, T, Int, $Nodes$, $LinkContrib$)

Input: V, T, Int, $Nodes$, $LinkContrib$
/* $Nodes$: a 2-dimensional array over $V \times T$ whose elements are subsets of V such that for each $v \in V$, $t \in T$: (1) $Int(v,t) = true$, and (2) for each $w \in Nodes[v][t]$: $Int(w,t)$ and both w and v belong to the same topic-connected component for t.
$LinkContrib$: an array of size $|T|$ with elements being sets of edges chosen from $V \times V$. If edge $e \in LinkContrib[i]$, then $e \notin E_{GM}$, and adding e to the overlay at the current iteration will reduce the number of topic-connected components by i (where $1 \leq i \leq |T|$). */
Output: A set E_{GM} of edges that form a topic-connected overlay

```
1: HighestContrib ← max{i | LinkContrib[i] ≠ ∅}
2: E_GM ← ∅
3: while HighestContrib > 0 do
4:     e ← some edge (v,w) from LinkContrib[HighestContrib]
5:     E_GM ← E_GM ∪ {e}
6:     delete e from LinkContrib[HighestContrib]
7:     for all t such that Int(v,t) ∧ Int(w,t) do
8:         for all v'∈Nodes[v][t], w'∈Nodes[w][t], (v',w')≠(v,w) do
9:             locate i such that (v',w') ∈ LinkContrib[i]
10:            delete (v',w') from LinkContrib[i]
11:            if i > 1 then
12:                add (v',w') to LinkContrib[i − 1]
13:        new_connected_component_list←Nodes[v][t]∪Nodes[w][t]
14:        for all u ∈ new_connected_component_list do
15:            Nodes[u][t] ← new_connected_component_list
16:    while HighestContrib>0 ∧ LinkContrib[HighestContrib]=∅ do
17:        HighestContrib ← HighestContrib − 1
18: return E_GM
```

Chockler *et al.* [6] proved the decision problem for MinAvg-TCO to be NP-complete and proposed a greedy algorithm that achieves a logarithmic approximation for the problem. The algorithm is centralized and the overlay computation requires complete knowledge of V and Int. The running time for MinAvg-TCO is $O(|V|^2|T|)$. The approach is only capable of building an overlay from scratch, whereas in practice due

to network partitioning, for instance, the combination of sub-overlays is an important concern.

Algorithm 1 specifies the Greedy Merge algorithm (GM algorithm), developed by Chockler *et al.* [6]. The algorithm works as follows: It starts with an empty set of links and iteratively adds carefully selected edges one by one until topic-connectivity is attained. At each iteration, the algorithm greedily selects an edge whose addition to the overlay would maximally reduce the total number of connected components for all the topics. In this paper, we use techniques from this algorithm both to devise an algorithm for the overlay join problem in Section V and from there derive a more efficient solution for the MinAvg-TCO problem in Section VI-A. The algorithm also serves as baseline in our experimentation.

IV. The Topic-connected Overlay Join Problem

In this section we introduce the TCO join problem and discuss similarities and differences to solving the MinAvg-TCO problem.

The joining of existing TCOs over the same set of topics but disjoint sets of nodes can be achieved by the known solutions 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 solutions for MinAvg-TCO to rebuild the overlay from scratch. However, for practical considerations, it can often be better to preserve existing edges and only incrementally add edges as needed to achieve topic-connectivity in the overlay combined from existing overlays. This is because establishing a new edge is a relatively costly operation and incremental computation of additional edges is more efficient than the re-computation of the entire overlay. Also, to re-build 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 an overlay. The formal definition of the TCO join problem, referred to as MinAvg-TCO-Join is as follows.

Definition 2. *MinAvg-TCO-Join*(V, T, Int, p)*: Given* p *topic-connected overlays:* $TCO_d(V_d, T, Int_d, E_d)$, d=$1, ..., p$, *which are node-disjoint, construct a topic-connected overlay* $TCO(V, T, Int, E)$ *where* $V = \bigcup_{d=1}^{p} V_d$, $Int = \bigcup_{d=1}^{p} Int_d$, *and* $\bigcup_{d=1}^{p} E_d \subseteq E$, *with the least possible total number of edges.*

We define the corresponding Avg-TCO-Join decision problem.

Definition 3. *Avg-TCO-Join*(V, T, Int, p, k)*: Given an integer* k *and* p *topic-connected overlays:* $TCO_d(V_d, T, Int_d, E_d)$, d=$1, ..., p$, *which are node-disjoint, determine if there exists a topic-connected overlay* $TCO(V, T, Int, E)$ *where* $V = \bigcup_{d=1}^{p} V_d$, $Int = \bigcup_{d=1}^{p} Int_d$, *and* $\bigcup_{d=1}^{p} E_d \subseteq E$, *such that* $|E| = k$.

MinAvg-TCO is a special case of MinAvg-TCO-Join when considering that each joining overlay contains only one node. Hence, all impossibility results about MinAvg-TCO (NP-completeness for the decision problem and impossibility of linear approximation unless P=NP) apply to Avg-TCO-Join and MinAvg-TCO-Join as well. For the practical concerns raised above, the previously known greedy algorithms are not directly applicable to solve MinAvg-TCO-Join.

Also, based on the considerations from above, given a number of topic-connected overlays, it is often more beneficial to solve MinAvg-TCO-Join incrementally rather than solving MinAvg-TCO from scratch. There is, however, a drawback, as the total number of edges resulting from incremental addition of edges can become much larger than the total number of edges resulting from overlay construction from scratch, as illustrated in the following example.

Assume a n-node set $V = \{v_1, v_2, ..., v_n\}$ and a n^2-topic set $T = \{t_{ij} | i, j = 1, 2, ..., n\}$. T is divided into n groups $T_i = \{t_{ij} | j = 1, 2, ..., n\} = \{t_{i1}, t_{i2}, ..., t_{in}\}, i = 1, 2, ...n$. 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 i-th topic t_{ji} from another topic group $T_j (\neq T_i)$. Therefore,

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

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

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

As an input to the MinAvg-TCO problem, a $n \times n$ complete overlay TCO_0 is generated for this instance. This is because each v_i has to link to all other nodes in order to achieve topic-connectivity for topics in T_i.

Assume a new node v_{all}, which is interested in all topics in the entire topic set T, requests to join. If the new topic-connected overlay TCO_1 is constructed incrementally by adding edges to TCO_0, then the number of edges in TCO_1 is still $\Theta(n^2)$, as illustrated in Figure 2(b). However, if we delete all existing edges and re-construct the TCO from scratch, it is sufficient to simply connect v_{all} to all others $\{v_1, v_2, ..., v_n\}$, which results in a TCO_2 with $\Theta(n)$ edges (cf. Figure 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 overlay from scratch. However, our experimental findings in Section VIII show that for typical pub/sub workloads, this situation virtually never occurs in practice.

In the next section, we introduce several solutions for the MinAvg-TCO-Join problem. Our solutions are based on determining cross-TCO edges that are added incrementally.

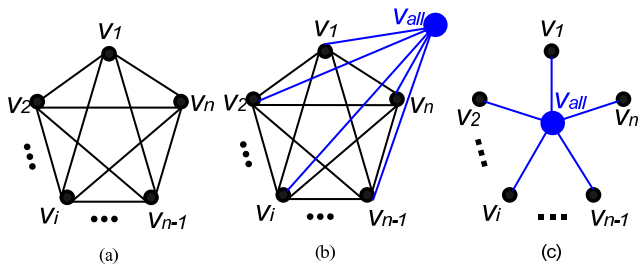

Fig. 2. (a) TCO_0 with $\Theta(n^2)$ edges. (b) TCO_1 built by incremental join has $\Theta(n^2)$ edges. (c) TCO_2 built from scratch has $\Theta(n)$ edges.

V. Solving The MinAvg-TCO-Join Problem

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

The intuition behind our solution is that we need to determine a set of cross-TCO links, that in conjunction with the already existing links internal to the TCOs, produce a combined TCO. Below we refer to inner links as E_{innNM} and outer links as E_{outNM}. E_{innNM} is easily obtained by the union of all edges within TCOs. The algorithm starts with an empty E_{outNM} and iteratively adds carefully selected edges one by one until topic-connectivity is attained. At each iteration, the algorithm selects a cross-TCO edge whose addition to the overlay would maximally reduce the total number of connected components for all topics. Algorithm 3 specifies our solution.

Algorithm 3 Naive Merge algorithm for MinAvg-TCO-Join.

NaiveMergeAlgorithm(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, ..., p

Output: A topic-connected overlay $TCO(V, T, Int, E_{NM})$ where $V = \bigcup_{d=1}^{p} V_d$, $Int = \bigcup_{d=1}^{p} Int_d$, $\bigcup_{d=1}^{p} E_d \subseteq E_{NM}$

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

Algorithm 3 follows the design of the GM algorithm given in Section III. The below correctness, approximation ratio, and running time properties can be established by adapting the respective proofs for the GM algorithm. For brevity we only focus on the key differences and omit other proof elements.

Lemma 1. (Correctness) *Algorithm 3 is correct: it yields an overlay such that for every topic t, all the nodes interested in t are organized in a single connected component.*

Algorithm 4 Compute cross-TCO links.

greedyConnect($V, T, Int, nodesAndInterests$)

Input: $V, T, Int, nodesAndInterests$
// $nodesAndInterests$: A list of p (V_d, Int_d) pairs

Output: A set E_{out} of cross-TCO edges that along with inner overlay edges form a TCO

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 $LinkContrib[weight]$
12: return **constructOverlayEdges**$(V, T, Int, Nodes, LinkContrib)$

Line 7 in Algorithm 3 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 topic-connected components for TCO_d and thus $\sum_{d=1}^{p} T_d$ is the total number of topic-connected components for the input of all TCOs. With proof techniques similar to Lemma 6.5 from [6], Lemma 2 is established.

Lemma 2. (Approximation Ratio) *The output of Algorithm 3 has a log approximation ratio as compared to the optimal solution:* $\frac{|E_{outNM}|}{|E'_{out}|} = O(\log(\sum_{d=1}^{p} T_d))$

Lemma 3. (Running Time) *The running time of Algorithm 3 is* $O(|T| \cdot \sum_{i=1}^{p} \sum_{j>i}^{p} |V_i||V_j|)$.

Proof: Let the set of all potential cross-TCO edges be:

$$E_{potNM} = \{(v, w) | v \in V_i \wedge w \in V_j, i \neq j\}$$

The running time cost of Algorithm 3 is dominated by the loop to compute initial edge weights in Line 9 of Algorithm 4 and the loop to update edge weights in Lines 8-12 of Algorithm 2. In particular, it takes $O(1)$ time to determine the edge with maximum weight in Line 4 of Algorithm 2 so that the total time of selecting all the edges in the solution is of lower complexity.

Initial weight computation has to be done for each of E_{potNM} edges resulting in $O(\sum_{(v,w) \in E_{potNM}} |\{t \in T | Int(v, t) \wedge Int(w, t)\}|)$ time. It takes $O(1)$ time to decrement the weight of an edge by 1. Therefore, the number of weight updates and hence the running time to build E_{outNM}, denoted as $\mathbb{T}_{NM}$, is

determined by the total initial weight of all E_{potNM} edges:

$$\mathbb{T}_{NM} = O(\sum_{(v,w)\in E_{potNM}} |\{t \in T | Int(v,t) \wedge Int(w,t)\}|)$$
$$= O(|T| \cdot |E_{potNM}|) = O(|T| \cdot \sum_{i=1}^{p}\sum_{j>i}^{p} |V_i||V_j|)$$

□

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

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

Definition 4. (Star set) *Given V, T and Int, a star set is a subset $S \subseteq V$ such that*

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

A node s is referred to as a star node *(or a* star*) if $\exists$ a star set S s.t. $s \in S$.*

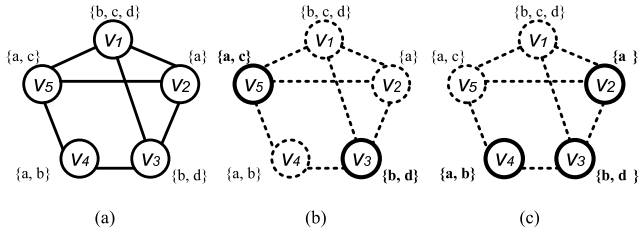

Fig. 3. (a) A TCO. (b) $\{v_3, v_5\}$ is a star set which covers all topics $\{a, b, c, d\}$. (c) $\{v_2, v_3, v_4\}$ is not a star set; it only covers $\{a, b, d\}$.

As illustrated in Figure 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 nodes of different star sets. 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 will include a node interested in t. By connecting nodes from different star sets we can achieve topic-connectivity for t.

Since minimal star sets tend to be substantially smaller than the total number of nodes, considering only star nodes as candidates for cross-TCO links gives rise to a number of advantages. One, the running time of the overlay construction algorithms considered in this paper is roughly proportional to the square of the number of nodes, therefore our algorithm that only considers star nodes runs much faster. Two, star sets of different TCOs can be computed in parallel in a fully decentralized fashion. Three, calculation of cross-TCO links no longer requires complete knowledge of V and Int, only a partial view of star nodes from star sets and their interests is required.

We still need to consider, how to efficiently determine a minimal star set given V, T, and Int. The problem of computing a minimal star set turns out to be precisely equivalent (through a linear reduction) to the classic set cover problem, which is NP-complete but has a logarithmic approximation [19]. Note that the size of the star set only affects the performance of our algorithm rather than its correctness. Algorithm 6 provides a standard greedy implementation that attains a provable logarithmic approximation. The algorithm 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 5 presents our Star Merge (SM) algorithm. The algorithm operates in two phases. First, it determines a star set for each sub-TCO. Second, it connects all the nodes in the star sets into a topic-connected overlay in a greedy manner by always selecting an edge with maximum contribution.

Algorithm 5 Star Merge algorithm for MinAvg-TCO-Join.

StarMergeAlgorithm(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, ..., p
Output: A topic-connected overlay $TCO(V, T, Int, E_{SM})$ where $V = \bigcup_{d=1}^{p} V_d$, $Int = \bigcup_{d=1}^{p} Int_d$, $\bigcup_{d=1}^{p} E_d \subseteq E_{SM}$

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

Algorithm 6 Determine star set for a TCO.

getStarSetFromNodes(V_d,T_{out_d},Int_d)
Input: V_d, T_{out_d}, Int_d
Output: S_d: A star set for V_d

1: Start with $R = T_{out\_d}$ and $S_d = \emptyset$
2: **while** $R \neq \emptyset$ **do**
3: $s \leftarrow \text{argmax}_{v\in V_d - S_d}(|\{t | t \in R \wedge Int(v,t)\}|)$
4: $S_d \leftarrow S_d \cup \{s\}$
5: $R \leftarrow R - \{t | Int(s,t)\}$
6: Return S_d

Below, we establish correctness, approximation ratio and running time properties for the SM algorithm.

Lemma 4. (Correctness) *Algorithm 5 is correct: it yields an overlay such that for every topic t, all the nodes interested in t are organized in a single connected component.*

Let E'_{out} and T_d follow the same definitions as for Lemma 2. Let E_{outSM} be the set of edges computed by Line 10 of Algorithm 5. This edge set connects star nodes in S_d $(d = 1, ..., p)$ across all TCOs. Let E'_{outSM} be the minimal edge set to connect S_d $(d = 1, ..., p)$. Then the following properties for Algorithm 5 hold.

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

Let S_d^* be the minimal star set of TCO_d for T_{out_d}, then:

Lemma 6. (Approximation Ratio) *The output of Algorithm 5 has the following approximation ratio as compared to the optimal solution:*
$\frac{|E_{outSM}|}{|E'_{out}|} = O(\log(\sum_{d=1}^{p} T_d) \cdot (\log |T|)^2 \cdot \sum_{d=1}^{p} |S_d^*|)$.

Lemma 6 shows that the output of Algorithm 5 is bounded by the sum of the sizes of minimal star sets of the given input.

Lemma 7. (Running Time) *The running time of Algorithm 5 is* $O(|T| \cdot (\sum_{d=1}^{p} |V_d| \log |V_d| + (\log |T|)^2 \sum_{i=1}^{p} \sum_{j>i}^{p} |S_i^*||S_j^*|))$.

Proof: First, suppose S_d is the star set found for TCO_d by Line 8 in Algorithm 5, then the set of all potential edges among star nodes is:

$$E_{potSM} = \{(v, w) | v \in S_i \wedge w \in S_j, i \neq j\}$$

Using the same reasoning as in Lemma 3, the running time cost to build E_{outSM} by Line 10 of Algorithm 5 is:

$$\begin{aligned} \mathbb{T}_{mergeStars} &= O(|T| \sum_{i=1}^{p} \sum_{j>i}^{p} |S_i||S_j|) \\ &= O(|T| (\log |T|)^2 \sum_{i=1}^{p} \sum_{j>i}^{p} |S_i^*||S_j^*|) \end{aligned}$$

Second, time cost to determine the star set for each sub-TCO is determined by Lines 6- 9 in Algorithm 5. If we assume that the topic set for each node is represented by a $|T|$-bit Boolean array, then the set difference operation takes $O(|T|)$ time. The running time to determine a star set for TCO_d with $|V_d|$ nodes is $O(|T||V_d| \log |V_d|)$ [20]. Thus, for all p sub-TCOs,

$$\mathbb{T}_{getStarSets} = O(|T| \sum_{d=1}^{p} |V_d| \log |V_d|)$$

Therefore, the running time for the Star Merge algorithm is:

$$\begin{aligned} \mathbb{T}_{SM} &= O(\mathbb{T}_{getStarSets} + \mathbb{T}_{mergeStars}) \\ &= O(|T| \cdot (\sum_{d=1}^{p} |V_d| \log |V_d| + (\log |T|)^2 \sum_{i=1}^{p} \sum_{j>i}^{p} |S_i^*||S_j^*|)) \end{aligned}$$

□

With Lemma 3 and Lemma 7, we can see that our SM algorithm improves the running time performance significantly because $|S_d^*| \ll |V_d|$ holds for most cases. We also show this experimentally.

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 the next section.

VI. Solving MinAvg-TCO By Divide-and-Conquer

In this section we show how our SM algorithm for MinAvg-TCO-Join can be extended to solve the MinAvg-TCO problem in a divide-and-conquer manner. We also show analytically that the resulting algorithm leads to a significantly improved running time cost as compared to the GM algorithm. In Section VIII, we quantify these improvements empirically. We first develop the divide-and-conquer algorithm in Section VI-A and then present its main analytical properties in Section VI-B.

A. The Divide-and-Conquer Algorithm

According to Chockler *et al.* [16], 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. An improvement to the running time cost could result from reducing the size of the node set when executing the GM algorithm. This suggests a divide-and-conquer strategy to solve the problem: (1) *divide* the MinAvg-TCO problem into several sub-overlay construction problems that are similar to the original overlay but with a smaller node set, (2) *conquer* the sub-MinAvg-TCO problems independently and build sub-overlays into sub-TCOs, and then (3) *combine* these sub-TCOs to one TCO as a solution to the original problem.

The existing GM algorithm can be employed to *conquer* the sub-MinAvg-TCO problems by determining inner edges used for the construction of the sub-overlays. The SM algorithm can take care of the *combine* phase by adding cross-TCO edges (i.e., outer edges) in a greedy manner. The DC algorithm that follows this design is presented in Algorithm 7. To *divide* nodes the algorithm uses a random partitioning. The number of partitions p is given as an input parameter and each sub-overlay has $k = |V_d| = \frac{|V|}{p}$ nodes, where $d = 1, ..., p$.

Basically 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 so 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.

The idea of node clustering is appealing because well-clustered nodes with strongly correlated interests would result in the GM algorithm producing lower average node degrees. The problem with this approach is its relative inefficiency. Clustering algorithms tend to exhibit high running time 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 run

time cost implications. It is challenging to fit node clustering into the DC algorithm so that the latter is still superior to the GM algorithm in terms of running time cost. Furthermore, it is difficult to devise an effective decentralized algorithm for node clustering that would not require complete knowledge of V and Int.

We choose random partitioning for our DC algorithm because it is extremely fast, more robust than node clustering, and it can be done in a decentralized manner. Furthermore, the construction of inner edges for each overlay only requires knowledge of node interests within the overlay. 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. We validate the effect of the increase in the average node degree empirically in Section VIII and show that this effect is insignificant.

Note that node clustering by interests may yield clusters that vary in size depending on the clustering algorithm used. For random partitioning, on the other hand, equal-sized partitions, that are optimal with respect to average node degree and running time, are simple to obtain.

Algorithm 7 Divide-and-Conquer algorithm for TCO.

DivideAndConquerForTCO(V, T, Int, p)
Input: V, T, Int, p
// p: the number of partitions, $1 \leqslant p \leqslant |V|$.
Output: A topic-connected overlay $TCO(V, T, Int, E_{DC})$

1: $E_{DC} \leftarrow \emptyset$
2: $L_{TCO} \leftarrow \emptyset$
3: Randomly divide V into p partitions V_d, d=1, 2, ..., p
4: **for** $d = 1$ to p **do**
5: $Int_d \leftarrow Int|_{V_d}$
6: $TCO_d(V_d, T, Int_d, E_d) \leftarrow$**greedyMerge**($V_d$,$T$,$Int_d$)
7: add TCO_d to L_{TCO}
8: $TCO(V, T, Int, E_{DC}) \leftarrow$ **StarMergeAlgorithm**(L_{TCO})
9: return $TCO(V, T, Int, E_{DC})$

B. DC Algorithm Analysis

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

Lemma 8. (Correctness) *Algorithm 7 is correct: it yields an overlay such that for every topic t, all nodes interested in t are organized in a single connected component.*

The proof for this lemma follows directly from the definition of star sets and has been omitted.

Given an instance of the MinAvg-TCO problem with (V, T, Int), suppose $TCO'(V, T, Int, E')$ is an optimal solution for it. The following analysis is based on the assumption that the optimal overlay for $(V_0, T, Int|_{V_0})$ where $V_0 \subset V$ has fewer edges than the optimal overlay TCO' for (V, T, Int).

Lemma 9. (Approximation Ratio) *Algorithm 7 for MinAvg-TCO produces an overlay with a total number of edges that has an approximation ratio of at most $O(p \cdot \log(|V||T|))$ as compared to the optimal solution: $\frac{|E_{DC}|}{|E'|} = O(p \cdot \log(|V||T|))$*

The proof of this statement is omitted from the paper because of the page limit. Please see [21] for the complete analysis.

Next, we look at the running time the DC algorithm. As defined before, p denotes the number of partitions provided for Algorithm 7. There are two types of edges that form the TCO produced by this algorithm: (1) E_{inDC}, the inner edges which are computed for each sub-overlay using the GM algorithm in Lines 4-7, $E_{inDC} = \bigcup_{d=1}^{p} E_d$; (2) E_{outDC}, the outer edges to connect star nodes across different sub-TCOs using the Star Merge algorithm in Line 8. Algorithm 7 produces the set of output edges $E_{DC} = E_{inDC} \bigcup E_{outDC} = (\bigcup_{d=1}^{p} E_d) \bigcup E_{outDC}$. $\mathbb{T}_{inDC}$ denotes the running time to build E_{inDC} in the loop in Lines 4-7. $\mathbb{T}_{outDC}$ denotes the running time to build E_{outDC} in Line 8 and let $\mathbb{T}_{DC}$ be the total running time cost of Algorithm 7. Then,

Lemma 10. (Running Time) *Algorithm 7's running time is:*
$\mathbb{T}_{DC} = O(|T| \cdot (\frac{1}{p}|V|^2 + (\log|T|)^2 \sum_{i=1}^{p} \sum_{j>i}^{p} |S_i^*||S_j^*|))$,
$\mathbb{T}_{inDC} = O(|T| \cdot \sum_{d=1}^{p} |V_d|^2)$,
$\mathbb{T}_{outDC} = O(|T| \cdot (\log|T|)^2 \cdot \sum_{i=1}^{p} \sum_{j>i}^{p} |S_i^*||S_j^*|)$.

Proof: First, the for-loop in Lines 4-7 of Algorithm 7 uses the GM algorithm to build a sub-TCO for each sub-overlay. The d-th sub-overlay has $|V_d|(= k)$ nodes, so the running time for the d-th TCO is: $\mathbb{T}_d = O(|V_d|^2|T|) = O(k^2|T|)$. To construct all p sub-TCOs, we have:

$$\mathbb{T}_{inDC} = \sum_{d=1}^{p} \mathbb{T}_d = O(p \cdot k^2|T|) = O(\frac{1}{p}|V|^2|T|)$$

Second, based on Lemma 7, we have:

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

Observe that all other time spent by the algorithm (such as the time of partitioning the set of nodes) is of lower complexity. Then,

$$\begin{aligned} \mathbb{T}_{DC} &= O(\mathbb{T}_{inDC} + \mathbb{T}_{outDC}) \\ &= O(|T| \cdot (\frac{|V|^2}{p} + (\log|T|)^2 \cdot \sum_{i=1}^{p} \sum_{j>i}^{p} |S_i^*||S_j^*|)) \end{aligned}$$

□

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.

$$\tilde{k} = \overline{|S_d|}, d = 1, ..., p$$

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

Lemma 11. (Running Time) *The running time of Algorithm 7 for MinAvg-TCO is $O(|T| \cdot (\frac{1}{p}|V|^2 + p^2\tilde{k}^2))$.*

C. Decentralizing the DC algorithm

Note that the DC algorithm as presented above is fully centralized. It is possible to decentralize it in the following way: (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 run the GM algorithm, (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: 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 VII. Note, the original GM algorithm does not lend itself to such decentralization.

VII. Selecting the Size of Partitions for DC

Our DC algorithm is parametrized with the number of partitions p. This parameter impacts all important characteristics of the algorithm's performance. When the number of partitions is 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. When the number of partitions is close to $|V|$ at the other end of the spectrum, the performance is dominated by the invocation of the SM algorithm at the *combine* phase. As we observed in Section V, the performance of the SM algorithm in this case is once again identical to that of the GM algorithm 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 value of the approximation ratio, as shown in Lemma 9. Further evaluation will be presented in Section VIII.

With respect to the running time, the analysis in Section VI-A establishes that the contribution of both partitioning the nodes and computing the star sets is small 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 proportional to $\frac{1}{p}|V|^2|T|$ (under the assumption that inner edges for different sub-overlays are computed sequentially rather than in parallel) while the time required for computing outer edges is proportional to $(\tilde{k}/k)^2|V|^2|T|$. Therefore, the combined sum of these two times is minimal if $1/p + (\tilde{k}/k)^2$ is minimal, which occurs when $\frac{\mathrm{d}}{\mathrm{dp}}(1/p + (\tilde{k}/k)^2) = 0$.

If, on the other hand, inner edges for different sub-overlays are computed in parallel in a decentralized fashion as discussed in Section VI-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 $\frac{\mathrm{d}}{\mathrm{dp}}(1/p^2 + (\tilde{k}/k)^2) = 0$.

Practical application of this analysis of the running time is complicated by the fact that it is difficult to assess $\tilde{k}$ analytically. However, this 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 can only compute the star sets and thus obtain $\tilde{k}$ without running the expensive calculation of inner and outer edges. Then, we can 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 algorithm is size of the *potential neighbor set*, that is the maximal fan-out that the DC algorithm may yield for a single node. We normalize this characteristic by the total number of nodes $|V|$. We define the *potential neighbor ratio* for a node v, denoted as $pn\text{-}ratio(v)$, as the percentage of nodes that are considered candidates for becoming a neighbor of v when executing DC. According to the DC algorithm, 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 ratio:

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

The significance of this characteristic is twofold. First of all, the maximal 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 [7]. Therefore, minimizing the potential neighbor ratio has a desirable effect from this point of view. Secondly, the potential neighbor ratio has additional principal meaning for the decentralized implementation of the DC algorithm discussed in Section VI-C. For a node v, $pn\text{-}ratio(v)$ 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 definition of a potential neighbor ratio to apply to the entire node set:

$$pn\text{-}ratio(V) = \max_{v \in V} pn\text{-}ratio(v) = \frac{k-1}{|V|} + \frac{(p-1)\tilde{k}}{|V|}$$

If we consider $pn\text{-}ratio(V)$ 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 ratio by partitioning the nodes and computing the star sets and using efficient numerical methods to approximately determine the minimum.

VIII. Evaluation

We implemented the Star Merge and Divide-and-Conquer algorithms, and other auxiliary algorithms in Java. These

algorithms are compared to the GM algorithm under varying experimental conditions, such as average node degree and running time ratio. The GM algorithm is used as a baseline because it produces the lowest average node degree of all known algorithms that run in polynomial time.

In all our experiments, the size of the entire node set $|V|$ ranges from 1 000 to 8 000 and the number of topics $|T|$ ranges from 100 to 1 000. The average number of topics that a node is interested in, denoted as $\overline{NodeIntSize}$, varies from 10 to 100. We used $|V| = 1000$, $|T| = 100$, $\overline{NodeIntSize} = 20$, and $p = 10$ for most of the experiments unless specified otherwise. Each topic $t_i \in T$ is associated with probability q_i, $\sum_i q_i = 1$, so that each node subscribes to t_i with a probability q_i. The value of q_i is distributed according to either a uniform, a Zipf (with $\alpha = 2.0$), or an exponential distribution. According to [16], these distributions are representative of actual workloads used in industrial pub/sub systems today. For some evaluations we only present results under the uniform topic popularity distribution and report on results and analysis according to other distributions. We plotted average node degree on a linear scale and ratios (time ratio and $pn\text{-}ratio$) on a log scale. Note that $pn\text{-}ratio$ presented in our experiment figures refers to $pn\text{-}ratio(V)$, the *potential neighbor ratio* for the node set V.

A. Star Merge for MinAvg-TCO-Join

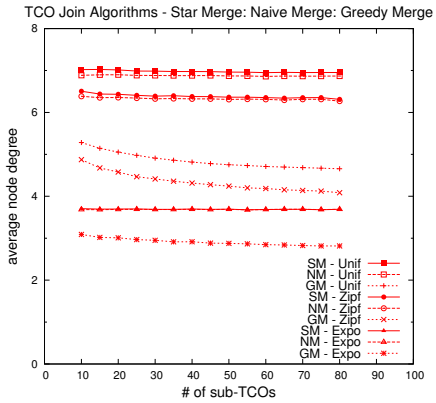

Fig. 4. Average Node Degree for MinAvg-TCO-Join Algorithms.

We evaluated the output and performance of two incremental TCO-join algorithms: Star Merge and Naive Merge. Also, these two algorithms are compared to GM which builds the TCO non-incrementally.

We generate several sub-TCOs as follows: Each sub-overlay TCO_d has $|V_d| = 100$ nodes ($d = 1, ..., p$, where p is the number of sub-TCOs), and all sub-overlays share the same topic set T with $|T| = 100$. These sub-overlays are constructed by the GM algorithm and fed to the TCO-join algorithms as input.

Figure 4 depicts that under different distributions, Star Merge and Naive Merge produce similar output overlays with a slightly higher number of edges compared to GM. However,

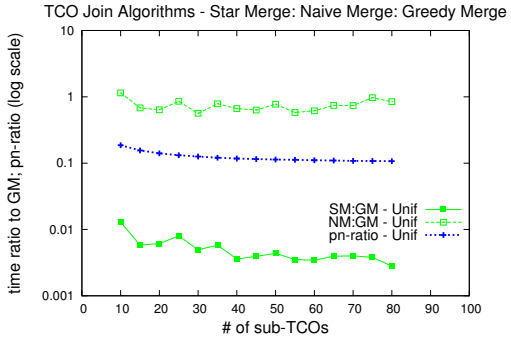

Fig. 5. Running Time Ratio for MinAvg-TCO-Join Algorithms.

the difference in the average node degree is insignificant ($\leqslant 2.5$).

Although Star Merge and Naive Merge produce quite close average node degrees (the difference is $\leqslant 0.15$), Star Merge runs considerably faster than Naive Merge. As it is shown in Figure 5, under the uniform distribution, it takes Star Merge less than 1% of the running time of GM on average, while the cost for Naive Merge is as much as around 75% of that for GM. This could be explained by different $pn\text{-}ratio$s between Star Merge and Naive Merge. Star Merge achieves much lower $pn\text{-}ratio$, which is $\leqslant 18.6\%$ in the worst case in our experiments under the uniform distribution. Naive Merge's $pn\text{-}ratio$ remains 100%, the same as for GM.

As the number of sub-TCOs that request to join increases, the average node degrees of the Star Merge output remains steady while the running time ratio of Star Merge to GM decreases considerably, which is caused by the decline in the $pn\text{-}ratio$. Notably, it only takes 0.28% of GM's running time for Star Merge to incrementally achieve a TCO with $8\,000$ nodes. This further attests to SM's scalability with respect to the number of nodes in the network.

B. DC for MinAvg-TCO

a) Random node partitioning for DC: We first evaluate the effects of random partitioning for the DC algorithm. We run the algorithm 400 times for the same settings (namely, the default experimental settings discussed above, the uniform distribution, and $p = 10$) 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. The statistics pertaining to the average node degree, running time ratio, and the average size of a star set are reported in Table I. As the table illustrates, all the values are quite 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 p 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 the DC algorithm.

Next, we study the impact of p on output and performance

TABLE I
RANDOM PARTITIONING UNDER UNIFORM DISTRIBUTION.

average node degree, time ratio, size of star set				
	mean	variance	min	max
avg node degree DC	6.50	0.000578	6.42	6.56
avg node degree GM	5.09	0.000175	5.07	5.13
time ratio DC:GM	0.0634	0.000143	0.0205	0.150
size of star set (k)	9.54	0.297	8	11

of DC. Given input V, T and Int, where $|T| = 200$, The DC algorithm is executed with all possible p values 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 $p = |V|$. Figure 6 shows that as the number of partitions p grows from 1 to $|V|$, DC's average node degree first increases gracefully and then it starts to decrease until it becomes equal to that of GM. Note that DC's average node degree never moves far from the horizon line of GM's output. Under the uniform distribution, the difference in average node degree between DC and GM is less than 6 even when the size of DC's output TCO 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 end of p's spectrum.

Figure 7 looks inside the DC algorithm and shows how "inner" (*divide* and *conquer*) phases and "outer" (*combine*) phase contribute to the algorithm for different p values. As we show, increasing p leads to the increase of E_{outDC} and $\mathbb{T}_{outDC}$. Under a reasonable partitioning (i.e., $pn\text{-}ratio < 20\%$) where the total running time cost is expected to be as low as possible, E_{inDC} and $\mathbb{T}_{inDC}$ are the dominant terms. This implies that a well-selected p should be relatively small and stand far away from $|V|$.

b) The Effects of $|V|$: Figure 8 depicts the comparison between DC and GM as the number of nodes increases. The number of partitions for DC is chosen based on both theoretical and experimental analysis. More specifically, we set the size of each partition to $k = |V_d| = 100$ and thus $p = \frac{|V|}{100}$. The figure shows that as the total number of nodes increases, DC has a larger impact on the running time while keeping the average node degree steadily low. Under the uniform distribution, for example, DC takes on average 1.70% of GM's running time to construct a high-quality TCO. The average node degree is around 6.97, which is less than 2.12 higher as compared to GM's.

c) The Effects of $|T|$: Figure 9 depicts how DC performs compared to GM as the number of topics changes. $|V|$ is set to $4\,000$ in this experiment. As the figure shows, under the uniform distribution, the average node degrees of both DC and GM increase with the number of topics. This is because increasing the number of topics leads to less correlation among nodes. However, DC's average node degree increases at a slow pace while the difference between DC and GM is insignificant: 3.66 for the uniform, 1.24 for Zipf and 1.06 for exponential distributions on average.

Also the running time ratio of DC to GM slightly increases with the number of topics. Still DC costs less than 28% of GM's running time and is at its worst when $|T|$ is as much as $1\,000$.

d) The Effects of $\overline{NodeIntSize}$: Figure 10 depicts how the number of node interests $\overline{NodeIntSize}$ affects the DC algorithm. We set $|V| = 4000$, $|T| = 400$, and $\overline{NodeIntSize}$ varies from 10 to 100. The figure shows that under the uniform distribution, as the number of topics a node subscribes to increases, the average node degree of both DC and GM decreases, and the difference between DC and GM shrinks. This is due to the increasing topic correlation within the node set when $\overline{NodeIntSize}$ grows. Like GM, DC also chooses each link in a greedy manner (but from a smaller set), so the selected edge is more likely to connect more topic-connected components since nodes share more common interests.

Beside, the running time ratio of DC to GM decreases with the increase of $\overline{NodeIntSize}$. Although more updates for an edge addition would be incurred, DC significantly reduces the time for each update compared to GM, since each edge update in DC only involves a small subset of all nodes.

e) Comparison with Ring-Per-Topic: We compare the average node degree obtained in the experiment by DC, GM and *Ring-Per-Topic (RingPT)*. RingPT is an algorithm that 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). According to RingPT, all the nodes interested in the same topic form a ring for that topic, after which rings for different topics are merged into a single overlay. As Figure 11 shows, the average node degrees of DC and GM are quite close (the average difference is 2.12), and the average node degree of RingPT, which is roughly equal to twice the average subscription size, is much more than 5 times the average node degrees of DC. This further demonstrates the scalability of DC when the number of nodes grows.

IX. CONCLUSIONS

In this paper, we introduce the novel MinAvg-TCO-Join optimization problem that attempts to add the minimal number of connections to join a number of topic-connected sub-overlays into a single TCO. MinAvg-TCO-Join is NP-complete. We develop the efficient and scalable Star Merge algorithm that constructs the TCO with a logarithmic approximation ratio compared to the optimal solution.

Our solution inspires a divide-and-conquer strategy that gives rise to a new algorithm for solving the original MinAvg-TCO problem. Theoretical analysis shows that with a reasonable partitioning of the node set, our DC algorithm is able to construct a high-quality TCO significantly faster than earlier alternative solutions. We motivate a novel random partitioning technique that lends itself to tuning through numerical analysis.

A comprehensive experimental analysis demonstrates the DC algorithm's performance. Under realistic workloads, DC successfully produces low average node degree TCOs with significantly lower running time cost than alternative solutions.

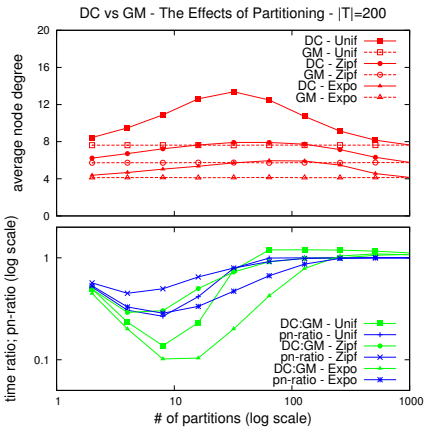

Fig. 6. DC vs GM as # of partitions increases.

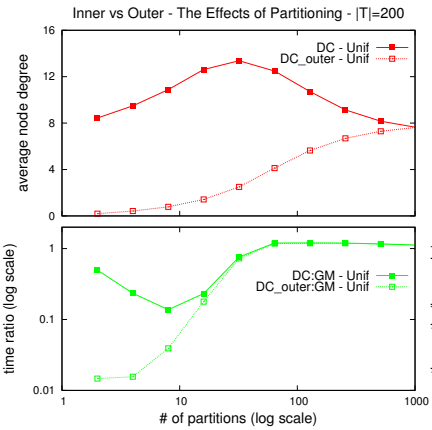

Fig. 7. Inner vs Outer as # of partitions increases.

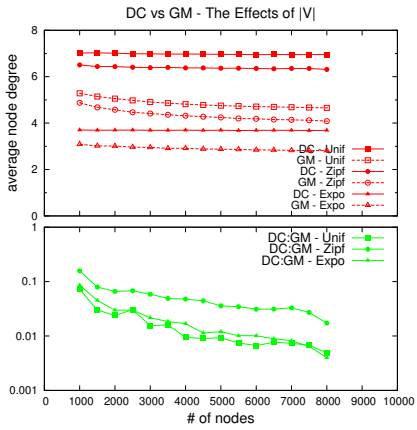

Fig. 8. DC vs GM as # of nodes increases.

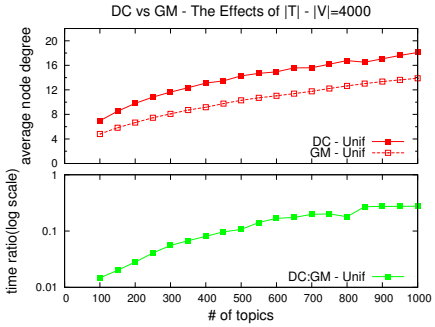

Fig. 9. DC vs GM as # of topics increases.

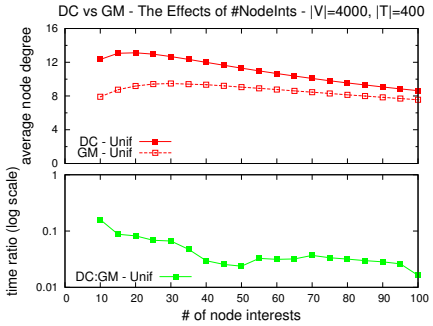

Fig. 10. DC vs GM as # of node interests increases.

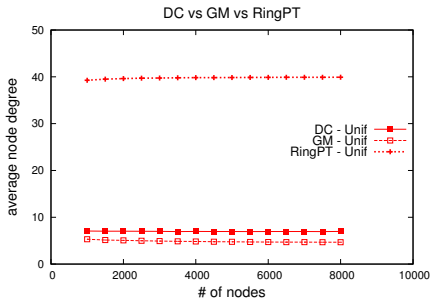

Fig. 11. DC vs GM vs RingPT

Besides, DC is more scalable than GM as the number of nodes, the number of topics and the number of topic interests increases.

Acknowledgments We are thankful for helpful comments and feedback from Patrick Lee, Vinod Muthusamy, Naweed Tajuddin. Chunyang Ye, and Young Yoon. The research presented in this project was supported in part by NSERC, ERA, CFI and the CANOE Collaboration in Higher Education between Norway and Canada.

REFERENCES

[1] J. Reumann, "Pub/Sub at Google," lecture at CANOE Summer School, Oslo, Norway, Aug'09.
[2] "Tibco rendezvous," 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 *IMC'05*.
[4] M. Petrovic, H. Liu, and H.-A. Jacobsen, "G-ToPSS: fast filtering of graph-based metadata," in *WWW'05*.
[5] G. Li, V. Muthusamy, and H.-A. Jacobsen, "A distributed service oriented architecture for business process execution," *ACM TWEB*, 2010.
[6] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Constructing scalable overlays for pub-sub with many topics: Problems, algorithms, and evaluation," in *PODC'07*.
[7] M. Onus and A. W. Richa, "Minimum maximum degree publish-subscribe overlay network design," in *INFOCOM'09*.
[8] M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann, "Self-organizing broker topologies for publish/subscribe systems," in *SAC'07*.
[9] 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, 2007.
[10] S. Voulgaris, E. Rivire, A.-M. Kermarrec, and M. V. Steen, "Sub-2-Sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks," in *IPTPS'06*.
[11] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni, "TERA: topic-based event routing for peer-to-peer architectures," in *DEBS'07*.
[12] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *JSAC*, 2002.
[13] D. Tam, R. Azimi, and H.-A. Jacobsen, "Building content-based publish/subscribe systems with distributed hash tables," in *DBISP2P'03*.
[14] G. Li, V. Muthusamy, and H.-A. Jacobsen, "Adaptive content-based routing in general overlay topologies," in *Middleware'08*.
[15] F. Araujo, L. Rodrigues, N. Carvalho, and N. Carvalho, "Scalable QoS-based event routing in publish-subscribe systems," in *NCA'05*.
[16] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: A scalable interest-aware overlay for topic-based pub/sub communication," in *DEBS'07*, pp. 14–25.
[17] E. Baehni, P. Eugster, and R. Guerraoui, "Data-aware multicast," in *DSN'04*.
[18] R. Chand and P. Felber, "Semantic peer-to-peer overlays for publish/subscribe networks," in *EUROPAR'05*.
[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2001.
[20] S. Chakravarty and A. Shekhawat, "Parallel and serial heuristics for the minimum set cover problem," *J. Supercomput.*, vol. 5, no. 4, 1992.
[21] C. Chen, H.-A. Jacobsen, and R. Vitenberg, "Star merge and divide-and-conquer algorithms for publish/subscribe topic-connected overlay design," U. of Toronto & U. of Oslo, Tech. Rep., 2010, http://msrg.org/papers/TRCJV-DC-2010.