Subscriber Assignment for Wide-Area Content-Based Publish/Subscribe

Albert Yu [1], Pankaj K. Agarwal [2], Jun Yang [3]

Department of Computer Science, Duke University

{ [1]syu, [2]pankaj, [3]junyang}@cs.duke.edu

***Abstract*— We study the problem of assigning subscribers to brokers in a wide-area content-based publish/subscribe system. A good assignment should consider both subscriber interests in the event space and subscriber locations in the network space, and balance multiple performance criteria including bandwidth, delay, and load balance. The resulting optimization problem is NP-complete, so systems have turned to heuristics and/or simpler algorithms that ignore some performance criteria. Evaluating these approaches has been challenging because optimal solutions remain elusive for realistic problem sizes. To enable proper evaluation, we develop a Monte Carlo approximation algorithm with good theoretical properties and robustness to workload variations. To make it computationally feasible, we combine the ideas of linear programming, randomized rounding, coreset, and iterative reweighted sampling. We demonstrate how to use this algorithm as a yardstick to evaluate other algorithms, and why it is better than other choices of yardsticks. With its help, we show that a simple greedy algorithm works well for a number of workloads, including one generated from publicly available statistics on Google Groups. We hope that our algorithms are not only useful in their own right, but our principled approach toward evaluation will also be useful in future evaluation of solutions to similar problems in content-based publish/subscribe.**

I. Introduction

A wide-area publish/subscribe system typically consists of an overlay network of *brokers*. *Events* originate from *publishers*, and are delivered by the brokers to interested *subscribers*. Traditional publish/subscribe is *topic-based*, where subscribers subscribe to a set of predefined topics such as "Apple news" or "American Idol." *Content-based* publish/subscribe, on the other hand, allows a subscriber to express an interest as a Boolean predicate against values of attributes inside events. For example, a subscriber may subscribe to eBay antique auctions with seller rating higher than 90% and starting bid between \$100 and \$200. Only events matching the predicate will be delivered to the subscriber. Content-based publish/subscribe is of interest to both database and networking communities [1], [2], [3], [4], because it must address the dual challenges of subscription matching in an event space and event dissemination in the network space.

An important problem in content-based publish/subscribe is *subscriber assignment*. Each subscriber needs to be assigned a broker responsible for forwarding matching events to this subscriber. Intuitively, we would like to assign subscribers with similar interests to the same broker, so that an event delivered to the broker could serve multiple subscribers simultaneously. If all subscribers assigned to the broker have similar interests, then only a subset of all possible events needs to go through the broker. At the same time, we may not want to assign a subscriber to a broker located far away in the network, because doing so increases delivery latency and communication cost. Finally, we should not assign too many subscribers to a single broker, which creates a performance bottleneck and delays event delivery. Balancing these considerations—similarity of interests in the event space, proximity of locations in the network space, and balance of load across brokers—is a difficult optimization problem.

The Need for a Yardstick. There is a good amount of previous work on subscriber assignment and related problems; see Section VII for details. Most approaches ignore some aspects of the problem or employ heuristic algorithms. For example, Aguilera et al. [1] assign subscribers to their closest brokers in the network, ignoring subscriber interests. On the other hand, Diao et al. [2] make assignment based on similarity of interests, without considering network latency. Papaemmanouil et al. [5] present a general optimization framework that considers multiple performance criteria, but relies on an iterative method to explore the solution space through local adjustments of dissemination trees.

It is understandable and often necessary to employ heuristics for subscriber assignment, because the problem in general is NP-complete. Evaluating these heuristics, however, is frustratingly difficult. How close are their solutions to the optimal? How well do they work on large, realistic workloads? Because of the problem's inherent complexity, optimal solutions for realistic problem sizes are computationally elusive and often unavailable for comparison. What would be a good yardstick then? Could yardsticks be solutions to simpler problems that ignore some performance constraints, since they are easier to compute and can act as lower bounds for the optimal solution?

Our Contributions. A main goal of this paper is to propose a better yardstick for evaluating the performance of various algorithms for the subscriber assignment problem. Our proposal is an algorithm called *SLP*, a shorthand for ***S**ubscriber Assignment by **L**inear **P**rogramming*. SLP jointly considers both subscriber interests in the event space and subscriber locations in the network space, and balances multiple performance criteria including bandwidth, delay, and load balance. While SLP's solution is not guaranteed to be optimal, it has provable properties that make it robust to workload variations, and reasonable as a yardstick for evaluating other algorithms.

Moreover, a by-product of running SLP (the LP fractional solution) gives us another useful indicator of how close a solution is to the optimal.

We also present $Gr^\star$, a simple offline greedy algorithm for subscriber assignment that presorts the subscribers in a particular way before assigning them one by one. Using SLP as a yardstick, we evaluate $\mathsf{Gr}^\star$ and a number of other algorithms. With the help of SLP, we are able to conclude, with confidence, that $\mathsf{Gr}^\star$ works very well for most (but not all) of the workloads tested. Our evaluation also reveals that simpler algorithms that ignore one performance criterion or another are poor yardsticks, because their solution cannot offer meaningful bounds on what can be realistically achieved when considering all constraints.

Another major obstacle for evaluation is the lack of publicly available, realistic workloads for content-based publish/subscribe. Information about user subscriptions (interests and locations) is rarely disclosed because of privacy concerns and commercial interests. Lack of widely deployed systems with powerful subscription languages also contributes to the difficulty. Thus, researchers have often resorted to synthesized workloads. However, simplistic workload generators run the risk of missing interesting patterns of clustering and overlap among subscriber interests, and correlations between subscriber interests and locations, which may influence the evaluation of subscriber assignment algorithms. Therefore, beyond simple synthetic workloads used for evaluation by previous work, we also evaluate our algorithms using workloads we generate [6] from publicly available statistics on Google Groups, which we believe to be closer to (at least one) reality.

SLP is computationally feasible on realistic problem sizes; we have run it on workloads consisting of hundreds of brokers and a million subscribers. We make SLP scalable by combining a suite of techniques, including randomized rounding, coreset, and iterative reweighted sampling. While SLP is slower than the simpler algorithms, its solution quality makes it well worthwhile in some settings, such as initial subscriber assignment, periodical re-optimization, and especially comparison with and evaluation of other algorithms.

II. Problem Statement

Let $\mathbb{N}$ denote the *network space*. Although our algorithm works on any metric space, for simplicity, we assume that $\mathbb{N}$ is a multi-dimensional Euclidean space, obtained by standard Internet embedding techniques [7], [8], [9]; Euclidean distance between two points approximates the network latency between them. Let $P \in \mathbb{N}$ be the *publisher* and $\mathcal{S} = \{S_1, \cdots, S_m\} \subseteq \mathbb{N}$ be a set of m *subscribers*.

P publishes *events*, each of which is represented as a point in the *event space* $\mathbb{E}$. We assume $\mathbb{E}$ to be the d-dimensional Euclidean space $\mathbb{R}^d$. Each subscriber S_i has a *subscription* $\sigma_i \subseteq \mathbb{E}$,[1] which we assume to be a d-dimensional rectangle. S_i receives an event $e \in \mathbb{E}$ if $e \in \sigma_i$.

[1]Without loss of generality, we assume one subscription per subscriber; an individual with multiple subscriptions can be modeled as multiple subscribers located at the same point in $\mathbb{N}$.

To disseminate events, we use a set $\mathcal{B} = \{B_1, \cdots, B_n\} \subseteq \mathbb{N}$ of n *brokers*. P and $\mathcal{B}$ together form a *dissemination network*, which we assume to be a tree $\mathcal{T}$ rooted at P. A leaf of $\mathcal{T}$ is called a *leaf broker*. A *subscriber assignment* $\Sigma : \mathcal{S} \to \mathcal{B}$ connects each subscriber to a leaf broker.

Filters. Each broker B_i is associated with a *filter* $f_i \subseteq \mathbb{E}$ such that if a broker B_j (resp. subscriber S_j) is a descendant of B_i, then $f_j \subseteq f_i$ (resp. $\sigma_j \subseteq f_i$). We call this condition the *nesting condition*. An event e is passed to a broker B_i from its parent if $e \in f_i$. To ensure simplicity and efficiency in implementing this forwarding logic, we require f_i to be the union of at most α rectangles, for some small constant α which we call *filter complexity*. In the special case of $\alpha = 1$, $\mathcal{T} \cup \Sigma$ becomes a bounding box hierarchy like an R-tree. We will, however, allow $\alpha > 1$.

Bandwidth. We are interested in minimizing $Q(\mathcal{T})$, the *expected total bandwidth consumption* (or *bandwidth* for short) of $\mathcal{T}$. $Q(\mathcal{T}) = \sum_{B_i \in \mathcal{B}} Q(B_i)$, where $Q(B_i)$ is the expected bandwidth *into* broker B_i. (We ignore the bandwidth required for leaf brokers to deliver events to subscribers because the total does not depend on the subscriber assignment.) When events are uniformly distributed, $Q(B_i) = \mathrm{Vol}(f_i)$. Our approach can be extended to a non-uniform event distribution π, in which case $Q(B_i) = \int_{f_i} \pi(e) de$.

Choosing $\alpha > 1$ can reduce bandwidth into a broker, as multiple rectangles can summarize child filters or subscriptions more precisely than a single rectangle, at the cost of increasing storage and processing overhead at the broker.

Latency. We want to bound the latency of delivering events to each subscriber S_j. We make a natural requirement in this paper: for a subscriber assignment Σ to be valid, the network latency of the path in $\mathcal{T} \cup \Sigma$ from the publisher to each subscriber S_j must not exceed the user-defined *maximum allowable latency* δ_j for S_j. Here, the path latency is the sum of distances in $\mathbb{N}$ between consecutive points on the path.

Our approach can be extended to handle other form of latency constraints, such as one that bounds only the last-hop latency to each subscriber (from the broker it is assigned to). More sophisticated constraints that account for broker processing delays can be enforced by additionally imposing load balance constraints described below.

Load Balance. We also want to ensure that not too many subscribers are assigned to one leaf broker. Without loss of generality, assume that $B_1, \cdots, B_l$ are the l leaf brokers in $\mathcal{B}$. Each leaf broker B_i is associated with a user-defined *capacity fraction* $\kappa_i \in [0, 1]$, such that $\sum_{i=1}^{l} \kappa_i = 1$. Perfect load balance happens when each B_i is assigned $\kappa_i m$ subscribers, but it is unnecessary and often undesirable as it may sacrifice other performance measures. Let m_i be the number of subscribers assigned to leaf broker B_i; we call $\max_{1 \le i \le l} \frac{m_i}{\kappa_i m}$ the *load balance factor* (*lbf*) of the assignment. We allow the user to cap the lbf at $\beta_{\max}$ and specify a *desired lbf* $\overline{\beta}$, where

$\beta_{\max} > \overline{\beta} > 1$. We try to find an assignment with lbf within $\overline{\beta}$; failing that, we try to find an assignment with lbf within $\beta_{\max}$ and as close to $\overline{\beta}$ as possible. The pair $(\overline{\beta}, \beta_{\max})$ allows the user to encourage load balance towards the desired level without rewarding assignments that "over-balance."

The Problem. The *subscriber assignment problem* (*SA*) is defined as follows: Given P, $\mathcal{B}$, $\mathcal{S}$, $\mathcal{T}$, maximum allowable latencies $\boldsymbol{\delta} = \{\delta_1, \ldots, \delta_m\}$, leaf broker capacity fractions $\boldsymbol{\kappa} = \{\kappa_1, \ldots, \kappa_l\}$, as well as parameters α, $\overline{\beta}$, and $\beta_{\max}$, compute an assignment $\Sigma : \mathcal{S} \to \mathcal{B}$ and filters for all brokers, such that the latency constraint is satisfied at each subscriber, the nesting condition is satisfied by all filters (each with no more than α rectangles), and the load balance factor is no more than $\overline{\beta}$ (or as close to $\overline{\beta}$ as possible and no more than $\beta_{\max}$). The assignment with the minimum expected total bandwidth $Q(\mathcal{T})$ will be returned. By reducing the standard set cover problem [10] to SA, we can show that SA is NP-complete.

III. Two Greedy Algorithms

We first present two simple greedy algorithms for SA, both aimed at minimizing bandwidth while meeting latency and load balance constraints.

The first algorithm, ***Online Greedy (Gr)***, assigns subscribers sequentially to leaf brokers, without having the entire set of subscribers available from the start. It considers the effect of incorporating the new subscription into existing filters in the event space, in a way similar to R-tree splitting heuristics. For each subscriber $S_j \in \mathcal{S}$, we define the *cost* of assigning S_j to a leaf broker B_i to be the sum of least volume enlargement of filters over the path in $\mathcal{T}$ from the publisher to B_i, such that the nesting condition is preserved. Gr identifies a set of *candidate brokers* (defined below) for S_j, and then greedily assigns S_j to the candidate broker with the minimum cost. We break a tie by choosing the least loaded broker (i.e., one with the minimum $\frac{m_i}{\kappa_i|\mathcal{S}|}$, where m_i is the number of subscribers already assigned to it).

B_i is a *candidate broker* for S_j if the following conditions are met: 1) Assigning S_j to B_i satisfies the user-defined latency constraint; 2) B_j will not be overloaded by this assignment; i.e., $\frac{m_i+1}{\kappa_i|\mathcal{S}|}$, is no more than a user-specified lbf. (This lbf can be set initially to $\overline{\beta}$; it can be increased if no feasible solution is found, eventually to $\beta_{\max}$.)

The second algorithm, ***Offline Greedy (Gr*)***, is an offline and more expensive variant of Gr. Each subscriber is processed in the exact same way as Gr. However, Gr* first sorts and then processes the set of subscribers in ascending order of the cardinality of their candidate broker sets. Intuitively, by deferring the processing of subscribers with more choices, we reduce the chance that Gr* will be forced into a costly decision due to lack of choices. Note that the assignment of earlier subscribers may restrict the choices available to later subscribers; hence, Gr* updates the ordering of remaining subscribers whenever a broker becomes fully loaded. As we will see in Section VI, Gr* not only consumes lower bandwidth than Gr but also produces much more balanced loads than Gr.

IV. One-Level SA

We now turn to a more sophisticated algorithm, SLP. In this section we describe SLP_1, the one-level version of SA, in which all brokers are directly connected to the publisher in $\mathcal{T}$. In Section V, we extend our solution to a multi-level $\mathcal{T}$.

Although SA can be written as an integer program, solving it directly is not computationally tractable even for the one level version. Realistic workloads involving hundreds of thousands of subscribers easily overwhelm the most sophisticated solvers. To tame the complexity of the problem, we first solve a carefully simplified problem to obtain a preliminary, but nonetheless good, assignment of filters to brokers; we then use it to derive the final solution to the full problem. The three-step strategy, illustrated in Figure 1, is as follows.

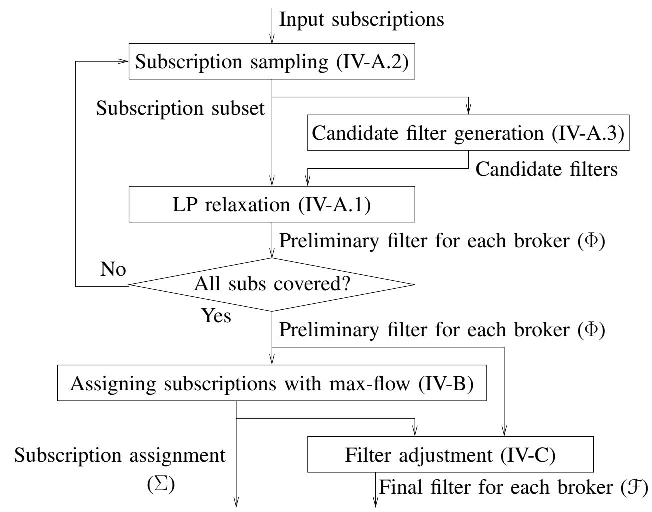


Fig. 1. Overview of SLP_1.

1) *Preliminary filter assignment.* The heart of SLP_1, this step produces a preliminary filter assignment $\Phi = \{\varphi_1, \ldots, \varphi_m\}$, where each broker B_i is assigned a filter φ_i. As motivated, this step considers all factors simultaneously in optimization—bandwidth, latency, and load balance—using LP relaxation and randomized rounding. To keep the size of the LP under control, instead of optimizing directly with all subscriptions and all possible filters, we choose representative sets (coresets) of subscriptions and candidate filters to consider in an iterative fashion.
2) *Subscription assignment.* Given the preliminary filter assignment Φ, this step considers the full set of subscriptions and computes the subscriber assignment $\Sigma : \mathcal{S} \to \mathcal{B}$. Since the filters are already given, this step focuses on load balancing while meeting latency constraints, using a max-flow algorithm.
3) *Filter adjustment.* Given Φ and Σ, this step further refines the filters and enforces the maximum filter complexity. Let $\mathcal{F} = \{f_1, \cdots, f_n\}$ be the resulting set of filters. The algorithm returns Σ and $\mathcal{F}$.

Algorithm 1: Preliminary filter assignment algorithm.

```
FilterAssign(𝓑, 𝓢) begin
    g ← 4;
    while g ≤ |𝓢| do
        foreach S ∈ 𝓢 do w(S) ← 1;
        q ← 10g ln g;
        for i ← 1 to 4g ln(|𝓢|/g) do
            repeat
                𝓠 ← Random(𝓢, w, q);
                Φ ← FilterAssignHelper(𝓠, 𝓑, 𝓢);
                if Φ = ⊥ then return ⊥;
                if Violate((1 + ε)Φ, 𝓑, 𝓢) = ∅ then
                    return (1 + ε)Φ;
                𝓥 ← Violate(Φ, 𝓑, 𝓢);
            until Σ_{S∈𝓥} w(S) ≤ ε Σ_{S∈𝓢} w(S);
            foreach S ∈ 𝓥 do w(S) ← 2w(S);
        g ← 2g;
    return ⊥;
FilterAssignHelper(𝓠, 𝓑, 𝓢) begin
    for j ← 0 to ln |𝓢| do
        𝓢_b ← Random(𝓢, 1, 10|𝓑|);
        𝓢_a ← 𝓠 ∪ 𝓢_b;
        𝓡 ← FilterGen(𝓢_a);
        Φ ← LPRelax(𝓑, 𝓡, 𝓢_a, 𝓢_b);
        if Φ ≠ ⊥ then return Φ;
    return ⊥;
```

A. Preliminary Filter Assignment

We present the first step of SLP_1, $\mathsf{FilterAssign}(\mathcal{B}, \mathcal{S})$ (Algorithm 1). We begin in Section IV-A.1 by describing LPRelax, a subroutine for computing a filter assignment using LP relaxation. Calling this subroutine with all subscriptions and all possible filters is impractical. Therefore, in Section IV-A.2, we use iterative reweighted sampling to obtain a coreset of subscriptions to run LPRelax with. In Section IV-A.3, we present a method for choosing a good subset of candidate filters to be considered by LPRelax.

*1) **LP Relaxation:*** We first describe $\mathsf{LPRelax}(\mathcal{B}, \mathcal{R}, \mathcal{S}_a, \mathcal{S}_b)$, which assigns each broker $B_i \in \mathcal{B}$ a filter consisting of rectangles in $\mathbb{E}$ drawn from a given set $\mathcal{R} = \{R_1, \cdots, R_u\}$. $\mathcal{S}_a$ denotes the subset of $\mathcal{S}$ considered by LPRelax; $\mathcal{S}_b \subseteq \mathcal{S}_a$ denotes the subset for which LPRelax enforces the load balance constraint (see (C3) below). Intuitively, we would like $\mathcal{S}_a = \mathcal{S}_b = \mathcal{S}$ and let $\mathcal{R}$ contain the minimum enclosing box of each non-empty subset of the subscriptions, but this would make the algorithm quite expensive in practice. We carefully choose a subset $\mathcal{S}_a \subseteq \mathcal{S}$ so that a filter assignment with respect to $\mathcal{S}_a$ is also good with respect to the entire set $\mathcal{S}$, and choose a subset $\mathcal{S}_b \subseteq \mathcal{S}_a$ to facilitate load balancing. We address how to choose $\mathcal{S}_a$ and $\mathcal{S}_b$ (and why to distinguish them) in Section IV-A.2, and how to choose $\mathcal{R}$ in Section IV-A.3.

For each subscriber $S_j \in \mathcal{S}_a$, let $\mathcal{B}_j \subseteq \mathcal{B}$ be the subset of brokers that satisfy the user-defined latency constraint for S_j if S_j is assigned to them; let $\mathcal{R}_j = \{R_k \mid \sigma_j \subseteq R_k \in \mathcal{R}\}$, i.e., the subset of given rectangles that contain S_j's subscription.

We formulate SA as a mixed integer program. We introduce two sets of Boolean variables x_{ij}, $y_{ik} \in \{0, 1\}$ for $i \in [1, n]$, $j \in \{j \mid S_j \in \mathcal{S}_a\}$, and $k \in [1, u]$, where

- $x_{ij} = 1$ iff subscriber S_j is assigned to broker B_i, and
- $y_{ik} = 1$ iff rectangle R_k is assigned to B_i as part of its filter.

The objective is to minimize $\sum_{B_i \in \mathcal{B}, R_k \in \mathcal{R}} \mathrm{Vol}(R_k) y_{ik}$,[2] subject to the following constraints:

(C1) [Filter complexity] Each broker is assigned a filter consisting of at most α rectangles:
$$\textstyle\sum_{R_k \in \mathcal{R}} y_{ik} \le \alpha \quad \forall B_i \in \mathcal{B}.$$

(C2) [Assignment and latency] Each subscriber is assigned to at least one broker meeting the latency constraint:
$$\textstyle\sum_{B_i \in \mathcal{B}_j} x_{ij} \ge 1 \quad \forall S_j \in \mathcal{S}_a.$$

(C3) [Load balance] The load balance factor is at most $\overline{\beta}$:
$$\textstyle\sum_{S_j \in \mathcal{S}_b} x_{ij} \le \overline{\beta} \kappa_i |\mathcal{S}_b| \quad \forall B_i \in \mathcal{B}.$$

(C4) [Nesting] A subscription can only be assigned to a broker whose filter contains it:
$$\textstyle\sum_{R_k \in \mathcal{R}_j} y_{ik} \ge x_{ij} \quad \forall S_j \in \mathcal{S}_a, \forall B_i \in \mathcal{B}_j.$$

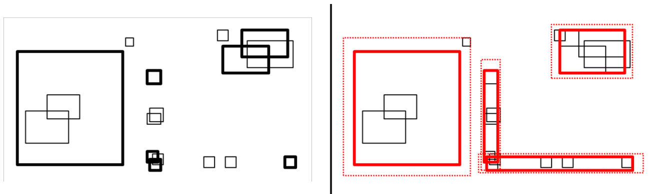
By relaxing the values of Boolean variables to be real numbers (i.e., $x_{ij}, y_{ik} \in [0, 1]$), the above mixed integer program can be reduced to an LP. Using an LP algorithm, we compute the optimal fractional solution, and then apply randomized rounding [10] to construct a solution to the filter-assignment problem. Specifically, for each y_{ik}, suppose $\hat{y}_{ik}$ is its value in the optional fractional solution. We set $\overline{y}_{ik}$ to 1 with probability $1 - (1 - \hat{y}_{ik})^{2 \ln |\mathcal{S}_a|}$, or 0 otherwise. The resulting filter assignment is $\Phi = \{\varphi_1, \ldots, \varphi_n\}$, where $\varphi_i = \{R_k \mid \overline{y}_{ik} = 1\}$.

Before returning Φ as a preliminary filter assignment, LPRelax further verifies that Φ *covers* $\mathcal{S}_a$. More precisely, we say that a subscriber S_j is *covered* by a filter assignment if there exists a broker B_i with assigned filter φ_i such that S_j's subscription σ_j is contained in one of the rectangles of φ_i, and the assignment of S_j to B_i satisfies the latency constraint for S_j. A set of subscribers is *covered* by a filter assignment if every subscriber in the set is covered. If it happens that Φ does not cover $\mathcal{S}_a$, LPRelax performs randomized rounding again for the y_{ik}'s to generate a new Φ. The scheme guarantees to produce a Φ covering $\mathcal{S}_a$ with probability at least $1/2$.

Remark. Because of rounding, φ_i may contain more than α rectangles; this violation is okay for now—recall from the beginning of Section IV that the goal of our first step in SLP_1 is not the *final* filter assignment, but a good, preliminary assignment to guide the reminding steps; in Section IV-C we will fix such violations.

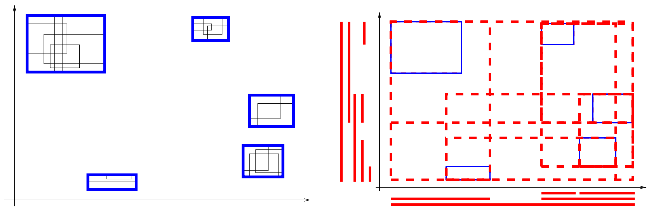
Note that we could also apply randomized rounding to x_{ij}'s and obtain a subscriber assignment for $\mathcal{S}_a$, but the resulting assignment may violate constraints due to rounding, and it is not the goal of this step of our algorithm.

[2] If filters consist of more than one rectangle ($\alpha > 1$), this objective function computes the sum of volumes of these rectangles instead of the volume of their union. We choose this function because it is simpler and discourages choosing overlapping rectangles for filters.



(a) (b)

Fig. 2. (a) Coreset members are drawn with thick outlines; (b) filters covering the coreset are ε-expanded to cover all subscriptions.



(a) Super-subscriptions (b) Rectangle generation

Fig. 3. Illustration of candidate filter generation.

*2) **Subscription Sampling:*** If we input all subscribers as $\mathcal{S}_a$ and $\mathcal{S}_b$ to LPRelax, the size of LP in Section IV-A.1 will be too large even for moderate number of subscribers. Therefore, we present a method to reduce the number of subscribers to input to LPRelax. This method combines two ideas:

- *Coreset*: For a wide range of geometric optimization problems, there exists a small subset (*coreset*) of the input objects such that the solution for this subset is a good approximation of the solution for the entire input [11]. Here, we show that for filter assignment, a small coreset of $\mathcal{S}$ exists and can be computed quickly.
- *Iterative reweighted sampling*: This idea has been previously used for problems such as linear programming [12], set cover [13], and computing coresets [14]. Here, we apply it to coreset computation for filter assignment.

We begin with a few definitions. For a rectangle $R = \prod_{i=1}^{d}[l_i, h_i]$, the *$\varepsilon$-expansion* of R, denoted by $(1+\varepsilon)R$, is $\prod_{i=1}^{d}[l_i - \varepsilon(h_i - l_i)/2, h_i + \varepsilon(h_i - l_i)/2]$. Similarly, the *$\varepsilon$-expansion* of a filter $\varphi = \{R_1, \ldots, R_\alpha\}$ is $(1+\varepsilon)\varphi = \{(1+\varepsilon)R_1, \ldots, (1+\varepsilon)R_\alpha\}$. Let $\Phi = \{\varphi_1, \cdots, \varphi_n\}$ be a filter assignment to $\mathcal{B}$, with φ_i being the filter associated with B_i, and let $(1+\varepsilon)\Phi = \{(1+\varepsilon)\varphi_1, \ldots, (1+\varepsilon)\varphi_n\}$. We call a subset $\mathcal{Q} \subseteq \mathcal{S}$ an *ε-certificate* if, for any filter assignment Φ that covers $\mathcal{Q}$, $(1+\varepsilon)\Phi$ covers $\mathcal{S}$ (recall the definition of "cover" from Section IV-A.1). We illustrate the notion of coreset in Figure 2. Lemma 1 in the appendix shows that there is always an ε-certificate whose size is independent of $|\mathcal{S}|$ although the worst case bound is exponential in $|\mathcal{B}|$. The size of an ε-certificate is likely to be much smaller in practice—as evident from our empirical results.

We now describe FilterAssign$(\mathcal{B}, \mathcal{S})$ (Algorithm 1), for computing a preliminary filter assignment using the ideas above. If we know there exists an ε-certificate of size g, then an iterative reweighted sampling scheme computes an ε-certificate of size $O(g \ln g)$ in $O(g \ln |\mathcal{S}|)$ iterations (Lemma 2 in the appendix). Without knowing g in advance, FilterAssign performs an exponential search on g, running $O(g \ln |\mathcal{S}|)$ iterations for a value of g and then doubling g.

Each stage of the search targets a specific g and consists of multiple *valid* iterations.[3] We maintain a weight for each subscriber in $\mathcal{S}$, initialized to 1 at the beginning of the stage. Each iteration chooses a random subset $\mathcal{Q} \subseteq \mathcal{S}$ of size $O(g \ln g)$, where each subscriber is chosen with probability proportional to its weight. We compute a filter assignment for $\mathcal{Q}$ using a helper procedure FilterAssignHelper described below. If the procedure finds an assignment Φ (by calling LPRelax), we check whether $(1+\varepsilon)\Phi$ covers the entire $\mathcal{S}$. If yes, FilterAssign stops and returns $(1+\varepsilon)\Phi$. Otherwise, we double the weight of each subscriber not covered by Φ, and begin a new iteration. An example is shown in Figure 4. If the number of valid iterations for the stage exceeds $4g\ln(|\mathcal{S}|/g)$, we conclude that the ε-certificate has size larger than g (by Lemma 2), and we move on to the next stage.

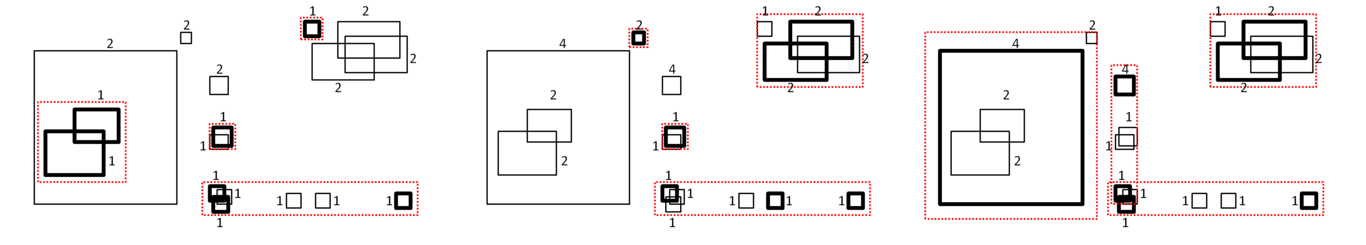
FilterAssignHelper, invoked by the inner loop of FilterAssign, further prepares the input for and calls LPRelax. The ε-certificate $\mathcal{Q}$ that we look for in FilterAssign is intended for the problem of covering $\mathcal{S}$, but since LPRelax considers coverage and load balance jointly, we must also ensure that our input to LPRelax properly reflects the properties of $\mathcal{S}$ relevant to load balancing. To this end, we choose a random subset $\mathcal{S}_b \subseteq \mathcal{S}$ of size proportional to $|\mathcal{B}|$ (we use $10|\mathcal{B}|$ for the practical sizes of $\mathcal{B}$ we consider). We call LPRelax with $\mathcal{S}_a = \mathcal{Q} \cup \mathcal{S}_b$, and $\mathcal{R} =$ FilterGen$(\mathcal{S}_a)$, where FilterGen is the candidate filter generation procedure to be described in Section IV-A.3. To guard against the small possibility that a random choice of $\mathcal{S}_b$ makes the otherwise feasible optimization problem infeasible, we repeat with a new choice of $\mathcal{S}_b$ (up to a small number of times) if LPRelax fails to find a feasible solution.

*3) **Candidate Filter Generation:*** We now describe the procedure FilterGen for constructing the set $\mathcal{R}$ of rectangles to be used by LPRelax to form filters. Without loss of generality, let $\mathcal{S} = \{S_1, \cdots, S_m\}$ denote the set of subscribers given as input to FilterGen (in reality, a subset may be given instead), and let σ_i denote S_i's subscription (a rectangle in $\mathbb{R}^d$). Each rectangle in $\mathcal{R}$ is intended to contain a subset of $\mathcal{S}$. There are $\Omega(m^{2d})$ rectangles, each of which contains a distinct subset.[4] However, this many rectangles make LPRelax impractical.

Therefore, we take two steps (see Figure 3) to ensure that $\mathcal{R}$ is small yet provides good coverage. The first step is optional. Here, we replace the input subscriptions with a set $\Xi = \{\xi_1, \cdots, \xi_k\}$ of k *super-subscriptions*, where k is proportional to the number of brokers (we set $k = 5|\mathcal{B}|$).

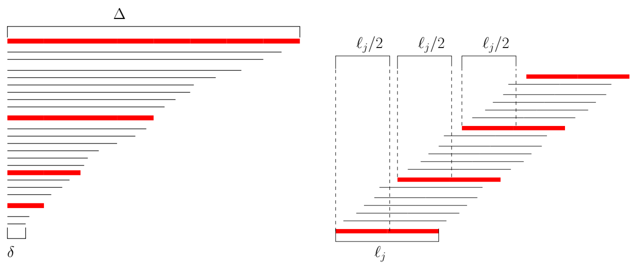
[3]This validity condition is needed to establish the termination condition of an iteration (Line 14 of Algorithm 1). A *valid* iteration is one where the ratio of the total weight of uncovered subscribers to that of all subscribers is no more than ε. By random sampling theory (Lemma 3 in the appendix), an iteration is valid with probability at least $1/2$, so we can simply repeat an iteration until it is valid.

[4]This lower bound is tight. In the case of $d = 1$, each subscription is an interval. Any interval I containing a subset of the m intervals can be shrunk so that the endpoints of I coincide with the endpoints of some of the m intervals. Hence, there are $O(m^2)$ candidate intervals. Generalizing this argument to higher dimensions, we can generate $O(m^{2d})$ candidate rectangles in $\mathcal{R}$.



(a) (b) (c)

Fig. 4. Three steps of iterative reweighted sampling: Choose a subset $\mathcal{S}_a$; find a filter assignment Φ of $\mathcal{S}_a$; (a, b) double the weight of all uncovered $S \in \mathcal{S}$, (c) The expansion of Φ covers $\mathcal{S}$.



(a) Consider only $\log_2(\Delta/\delta)$ different lengths.

(b) No two intervals of length ℓ_j overlap by more than $\ell_j/2$.

Fig. 5. Two main ideas for the rectangle generation step.

We obtain these super-subscriptions by partitioning $\mathcal{S}$ into k clusters and choosing the minimum enclosing box (MEB) of the subscriptions in each cluster. This clustering is done in a joint network-event space, and captures geographical and topical concentration of interests. In the second step, instead of generating $O(k^{2d})$ rectangles, we use a hierarchical procedure that generates fewer rectangles. The intuition is that if latency and load balancing constraints are not too tight, there is some flexibility in assigning subscribers to brokers and the filters can be "loose." The first step is relatively straightforward; see the technical report version [15] of this paper for details. We now describe the second step in more detail.

In the second step, for each dimension $i \in [1, d]$, we construct a set $\mathcal{J}_i$ of intervals on the x_i-axis. We set $\mathcal{R}$ to be the Cartesian product of these sets, i.e., $\mathcal{R} = \{J_1 \times \cdots \times J_d \mid \forall i \in [1, d] : J_i \in \mathcal{J}_i\}$. It thus remains to describe the construction of $\mathcal{J}_i$. Let $\mathcal{I}_i$ be the set of k intervals that are the projection of Ξ onto the x_i-axis. Let Δ be the length of the smallest interval containing $\mathcal{I}_i$, and let δ be the length of the smallest interval in $\mathcal{I}_i$. For $1 \le j \le \log_2(\Delta/\delta)$, let $\ell_j = 2^j\delta$. (If Δ/δ is large, we choose ℓ_j's more carefully.) For each j, let $\mathcal{I}_{ij} \subseteq \mathcal{I}_i$ be the set of intervals of length at most $\ell_j/2$; our goal is generate a set of intervals $\mathcal{J}_{ij}$ of length at most ℓ_j such that every interval of $\mathcal{I}_{ij}$ is contained by one in $\mathcal{J}_{ij}$, and no two intervals in $\mathcal{J}_{ij}$ overlap by more than $\eta\ell_j$ ($1/2 \le \eta < 1$; we use $\eta = 1/2$). Figure 5 illustrates the ideas.

To avoid two intervals in $\mathcal{J}_{ij}$ overlapping by more than $\eta\ell_j$, let $\mathcal{L}$ be the set of left endpoints of intervals in $\mathcal{I}_{ij}$, sorted in increasing order. We scan $\mathcal{L}$ from left to right and do the following. We take the first point, say p, of $\mathcal{L}$ and remove all points from $\mathcal{L}$ that are within distance $(1-\eta)\ell_j$ from p. Let J be the interval of length ℓ_j with p as its left endpoint. We shrink J to the smallest possible interval that still contains the same subset of intervals in $\mathcal{I}_{ij}$. We then add J to $\mathcal{J}_{ij}$ and repeat the above step, until $\mathcal{L}$ becomes empty, at which point we add $\mathcal{J}_{ij}$ to $\mathcal{J}_i$ and move on to the next j. In the worst case, $|\mathcal{J}_i| = O(k \log_2 \Delta/\delta)$, but in practice we expect it to be closer to $O(k)$ or even smaller. Hence, the size of the filter candidate set is $O(k^d)$, but it can be further reduced by working in high dimension directly if $\mathbb{E}$ has high dimensionality. FilterGen shrinks each rectangle $R \in \mathcal{R}$ to the MEB of subscriptions contained by R and returns $\mathcal{R}$ to FilterAssignHelper.

B. Subscription Assignment

The second step of SLP_1 takes as input the preliminary filter assignment Φ produced by FilterAssign in Section IV-A, and computes the subscriber assignment $\Sigma : \mathcal{S} \to \mathcal{B}$, for the entire set of subscribers. Since the filters are already given, we are not concerned with minimizing bandwidth here; instead, we focus on load balance while ensuring that subscribers are only assigned to brokers that *cover* them (recall the definition of "cover" from Section IV-A.1, which considers both nesting and latency constraints). Also, recall from Section II that $\overline{\beta}$ and $\beta_{\max}$ are user-defined desired and maximum load balance factors (lbfs), resp.; our goal is to find a Σ whose lbf is no more than $\overline{\beta}$, or else, close to $\overline{\beta}$ and no more than $\beta_{\max}$.

We formulate the computation of Σ as a max-flow problem. We construct a bipartite graph $G = (V, E)$, where $V = \mathcal{S} \cup \mathcal{B} \cup \{s, t\}$, $E = E_1 \cup E_2 \cup E_3$, $E_1 = \{(s, B) \mid B \in \mathcal{B}\}$, $E_2 = \{(S, t) \mid S \in \mathcal{S}\}$, and $E_3 = \{(B_i, S_j) \mid B_i \text{ covers } S_j\}$. We set the capacity of every edge in $E_2 \cup E_3$ to 1, and the capacity of an edge (s, B_i) in E_1 to $\lfloor \beta\kappa_i|\mathcal{S}| \rfloor$. Initially, we let $\beta = \overline{\beta}$, but it may increase over time to $\beta_{\max}$.

We compute the maximum flow from s to t. Let f be the value of the maximum flow. If $\mathsf{f} = |\mathcal{S}|$, then every subscriber in $\mathcal{S}$ is assigned to a broker, which can be identified by the edge into the subscriber with flow of 1. We return the resulting subscriber assignment, which by construction has a lbf of no more than β. If $\mathsf{f} < |\mathcal{S}|$ and $\beta = \beta_{\max}$, we conclude that the load balance constraint is too tight, and we stop. If $\mathsf{f} < |\mathcal{S}|$ and $\beta < \beta_{\max}$, we increase the value of β by a small factor, update the capacity of the edges in E_1, and recompute the maximum flow from s to t. Depending on the maximum flow algorithm employed, as an optimization, we can reuse the current flow as the starting flow for the increased value of β.

C. Filter Adjustment

The third and last step of SLP_1 further adjusts the preliminary filter assignment $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ made by FilterAssign. Based on the subscriber assignment $\Sigma : \mathcal{S} \to \mathcal{B}$

made by the second step, this step opportunistically tightens the filters, and enforces the filter complexity constraint (that each φ_i consists of no more than α rectangles). Consider each broker B_i with preliminary filter φ_i. Let $\mathcal{S}_i \subseteq \mathcal{S}$ be the set of subscribers assigned to B_i. We want to replace φ_i by f_i, a set of no more than α rectangles, such that $\bigcup_{S_j \in \mathcal{S}_i} \sigma_j \subseteq \bigcup_{R \in f_i} R$ and $\mathrm{Vol}(\bigcup_{R \in f_i} R)$ is minimized. The problem is NP-hard [16] in general, so we use a simple heuristic. Roughly speaking, for each preliminary filter, we cluster its subscriptions in the event space into α groups, and construct an alternative filter consisting of α MEBs, one for each group. See [15] for details.

D. Discussion

SLP_1 involves solving an LP given $\mathcal{B}$, $\mathcal{S}_a$, $\mathcal{S}_b \subseteq \mathcal{S}_a$, and $\mathcal{R}$ (Section IV-A.1). The optimal LP fractional solution provides a lower bound for the optimal bandwidth of subscription assignment with respect to $\mathcal{S}_a$ and $\mathcal{R}$. Randomized rounding for the y_{ik}'s would increase the expected bandwidth of $\mathcal{T}$ and expected filter complexity of each broker by a factor of $2\ln|\mathcal{S}_a|$ in the worst case, but since the size of an ε-certificate is independent of $|\mathcal{S}|$, $|\mathcal{S}_a|$ is likely much smaller than $|\mathcal{S}|$, so the blow-up is closer to a small constant factor—as evident from our empirical results.

With Theorem 1 in the appendix, we show that there exists a rounding scheme for x_{ij}'s such that the latency and nesting constraints are strictly enforced, and the expected load balance of a broker can be increased by a factor of at most $2\ln|\mathcal{S}_a|$ with respect to the random subset $\mathcal{S}_b$. If $|\mathcal{S}_b|$ is large enough, the expected load is balanced with the entire set of subscribers by using existing theoretical results on ϵ-approximation. Since our max-flow based algorithm optimizes load balancing, the resulting subscriber assignment is better than the one obtained from the rounding scheme in terms of load balancing.

The optimal filter assignment for $\mathcal{S}$ is also a filter assignment for $\mathcal{S}_a \subseteq \mathcal{S}$; therefore, the LP fractional solution, optimal with respect to $\mathcal{S}_a$, must be a lower bound for the optimal solution with respect to $\mathcal{S}$. However, decreasing the cardinality of $\mathcal{R}$ can increase the fractional value. Note that the two steps in candidate filter generation are orthogonal to one another. We can prove that given the set of super-subscriptions, the pruning of filters in the interval generalization step only degrades the final fractional solution by a constant factor because for any rectangle R excluded from the candidate filter set $\mathcal{R}$, there exists a filter $R' \in \mathcal{R}$ such that $R \subseteq R'$ and $\mathrm{Vol}(R) \approx \mathrm{Vol}(R')$. More precisely, if the first step is skipped, i.e. every subscription is a super-subscription, the fractional solution matches the lower bound of the optimal solution up to a small constant factor by Lemma 4. However, we cannot bound the blow-up due to the super-subscription clustering step; it is a necessary trade-off between performance and effectiveness.

V. Multi-Level SA

We now describe an algorithm for SA when the broker tree $\mathcal{T}$ has multiple levels of brokers. One possible approach is to first run the one-level algorithm SLP_1 (Section IV) over all leaf brokers, and then compute the filters at the interior nodes of $\mathcal{T}$ in a bottom-up manner. This approach has two drawbacks. First, sibling brokers in $\mathcal{T}$ may be assigned very different subscriptions, forcing a large filter at their parent which consumes a lot of bandwidth. Second, solving SLP_1 on a large set of brokers is computationally expensive. In practice, broker trees often follow the topology of the underlying network, so a top-down hierarchical approach will be effective.

Our algorithm works by recursively applying the one-level algorithm SLP_1 to subtrees in $\mathcal{T}$ in a top-down manner. At each non-leaf broker B of $\mathcal{T}$, we invoke SLP_1 to distribute the subscribers among B's children, deciding in which subtree of B each subscriber will be assigned. We then recursively process each child with the set of subscriptions assigned to the corresponding subtree.

To invoke SLP_1 over a set of non-leaf sibling brokers, we still need to address the issues of determining appropriate latency and load balance constraints for assigning a subscriber to these brokers—recall from Section II that the actual latency to a subscriber depends on its leaf broker assignment, which we have not made yet because of top-down processing; the load balance constraints have only been defined for leaf brokers. See [15] for how to address these two issues.

VI. Evaluation

Other Algorithms Tested. For comparison with Gr, $\mathsf{Gr}^\star$, SLP_1, and SLP, we also consider other algorithms. The first one is a variant of Gr that ignores latency. (Note that it is less sensible to ignore load balance, because there would be a strong incentive to assign every subscriber to the same broker.)

- ***Online Greedy without Latency Consideration*** *($\mathsf{Gr}_{\neg l}$)*. This algorithm works exactly like Gr, except that it drops the latency constraint in defining candidate broker sets. The answer produced by $\mathsf{Gr}_{\neg l}$ is useful in understanding how latency constraints affect attainable bandwidth.

We additionally consider other algorithms that ignore bandwidth and instead focus on some other performance metrics. As we will see, like $\mathsf{Gr}_{\neg l}$, these algorithms do poorly on the metrics they ignore, but they help illustrate the importance of considering multiple metrics jointly in optimization.

- ***Closest Broker without Load Balance*** *($\mathsf{Closest}_{\neg b}$)*. This algorithm resembles the one in [1]. It assigns each subscriber to its closest leaf broker in the network space (hence minimizing the last-hop latency). Ties are broken arbitrarily.
- ***Closest Broker*** *($\mathsf{Closest}$)*. Like $\mathsf{Closest}_{\neg b}$, this algorithm assigns each subscriber to its closest leaf broker. However, once a broker has already been assigned the maximum number of subscribers allowed by the user-specified maximum lbf $\beta_{\max}$, $\mathsf{Closest}$ drops it from further consideration.
- ***Best Load-Balanced Assignment*** *($\mathsf{Balance}$)*. This algorithm finds the assignment with the best possible lbf (possibly less than the user-specified desired lbf $\overline{\beta}$) by solving a max-flow problem. The graph construction is a variant of the one in Section IV-B.

Workloads. As discussed in Section I, it is important to base evaluation on realistic workloads, but they are difficult to find. Our earlier work [6] addressed this issue by developing a workload generator based on publicly available statistics on Google Groups. Extrapolating from these statistics, the generator produces a baseline workload consistent with them, and can generate additional workloads that deviate in meaningful ways from the baseline. We use multiple workloads produced by this generator (collectively referred to as ***workload set #1***) for evaluation. The network locations are mapped to points in $\mathbb{N} = \mathbb{R}^5$, and the subscriptions are rectangles in $\mathbb{E} = \mathbb{R}^2$. We vary two factors—*IS*, interest skewness in terms of popularity, and *BI*, number of broad interests (i.e., large rectangles)—between the settings of L(ow) and H(igh). The baseline workload from Google Groups resembles (IS:H, BI:L). The distribution of subscribers across Asia, North America, and Europe is $4 : 1 : 4$. The distribution of brokers across the network space is set to be roughly the same as that of the subscribers.

Workload set #2, designed to reproduce those used for evaluation in [17], [18], [5], is based on observations of the RSS feed popularity. A total of 50 different interests are generated and their popularity follows a Zipf distribution with exponent 0.5. Each interest is mapped to a random unit square in $\mathbb{E}$. Given an interest, subscriber locations are drawn uniformly at random from 10 locations in $\mathbb{N}$. In this workload set, the subscriber interests are essentially topic-based, and no notion of "proximity" is captured in either the event space or the network space.

Workload set #3 is designed to mimic those used in [19], [20], [21]. We partition the event space into 100 grid cells. The center of a subscription is mapped to the center of one of the cells. To create hot spots in $\mathbb{E}$, we rank the cells in random order; the probability of picking a cell as a subscription center follows a Zipf distribution with exponent 0.5. There is also a set of predefined subscription widths. For each dimension, the width of a subscription is chosen from this set according to a Zipf distribution with exponent 0.5. Each subscriber is randomly located at one of the network locations in $\mathbb{N}$; therefore, subscriber interests and locations are independent.

More details on how to generate these workload sets are available in [15]. We focus on results for workload set #1 as it is more realistic; additional results are in [15].

Problem Settings. Unless otherwise specified, we use the following settings for the SA problem. We set filter complexity $\alpha = 3$. Latency constraints are specified using a *maximum delay* of 0.3; the *delay* experienced by a subscriber S under a subscriber assignment Σ is defined to be $\delta/\Delta - 1$, where δ is the latency of the path in $\mathcal{T} \cup \Sigma$ from the publisher to S, and Δ is latency of the shortest path from the publisher to S through $\mathcal{T}$. For the load balance constraints, all leaf brokers have equal capacity fractions. For workload set #1, the desired and maximum load balance factors, $\overline{\beta}$ and $\beta_{\max}$, are 1.5 and 1.8, respectively. For workload set #2, since the subscribers of an interest are restricted to only a few network locations, subscriber distribution is skewed in $\mathbb{N}$ due to interest skewness.

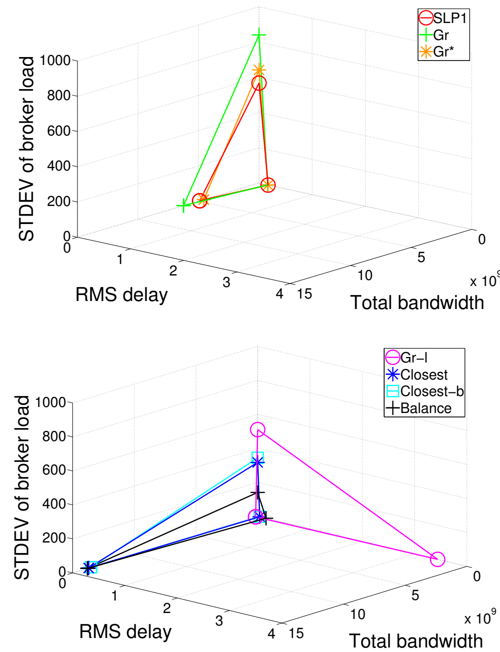


Fig. 6. Overall comparison (one-level network, workload set #1).

Therefore, we set $\overline{\beta}$ and $\beta_{\max}$ to more relaxed values of 2.3 and 2.5, respectively. For workload set #3, since subscriber locations are random, we tighten $\overline{\beta}$ to 1.3 and $\beta_{\max}$ to 1.5.

We compare the two greedy algorithms in Section III and the algorithms described earlier in this section together with SLP_1 (for one-level broker networks) or SLP (for multi-level broker networks). The quality of a solution is measured in terms of total bandwidth, subscriber delays, and broker loads (i.e., number of subscribers assigned to each broker). For non-deterministic algorithms, we do five runs and report the average (when applicable); we have found deviation in results to be insignificant.

Solution Quality for a One-Level Broker Network. In the following, we have 100,000 subscribers to assign to 100 brokers attached directly to the publisher.

Overall Comparison: Figure 6. To get a quick overview, we plot the result quality of each algorithm on workload set #1 as a triangle whose vertices correspond to total bandwidth, root mean square (RMS) of delay across subscribers, and standard deviation (STDEV) of broker loads. The numbers reported are averaged over four workloads: (IS:L, BI:L), (IS:H, BI:L), (IS:L, BI:H), and (IS:H, BI:H).

We see that SLP_1 and $\mathsf{Gr}^\star$ do well in minimizing bandwidth while bounding delay and load balance. Gr is worse: not only it incurs higher bandwidth, but it also produces very unbalanced loads (while SLP_1 and $\mathsf{Gr}^\star$ stay right within the maximum lbf). In fact, for all four workloads, Gr fails to find a feasible solution that satisfies the load balance constraints; nonetheless, we report the best-effort solutions found by Gr. We also tried variants of Gr: whenever we cannot assign a subscriber S_j because all its candidate brokers are fully loaded, we randomly remove some subscribers from these brokers to make room for S_j, and either reassign the removed subscribers next, or append them to the list of subscribers to be processed later. These variants still failed to find feasible solutions, even when given longer time to run than SLP_1.

On the bottom, we see algorithms that ignore one performance criterion or another do poorly. By failing to consider subscriptions in the event space, $\mathsf{Closest}_{\neg\mathsf{b}}$, Closest, and Balance incur huge bandwidth. By ignoring latency constraints in the network space, $\mathsf{Gr}_{\neg\mathsf{l}}$ produces unacceptable delays. $\mathsf{Closest}_{\neg\mathsf{b}}$ does okay with load balance in this case only because the broker and subscriber distributions are similar; in general $\mathsf{Closest}_{\neg\mathsf{b}}$'s load imbalance can be arbitrarily bad.

One question that we set out to answer with these experiments is whether, in practice, we could use the solution to a more tractable optimization problem that ignores some constraints as a (lower-bound) yardstick for gauging the quality of the solution to the full optimization problem. Here it is clear that $\mathsf{Gr}_{\neg\mathsf{l}}$ is not a good yardstick—compared with the other algorithms, its bandwidth is just too low and too unrealistic to serve as a meaningful yardstick.

But then, how could we conclude that a solution is "good enough" with respect to the optimal? The solution of SLP_1, though not guaranteed to be optimal, serves as a reasonable indicator because of SLP_1's theoretical properties. Next, we will see how a by-product of running SLP_1, namely the LP fractional solution (Section IV-D), can further help.

Bandwidth: Figure 7(a), Tables I and II. In Figure 7(a), we take a closer look at total bandwidth consumption across workload set #1. The relative ordering of the algorithms is fairly consistent. SLP_1 and $\mathsf{Gr}^\star$ are good and comparable. Gr is consistently worse (not to mention its solutions also violate load balance constraints). Algorithms that ignore the event space are the worst. Again, $\mathsf{Gr}_{\neg\mathsf{l}}$ (barely visible in the figure) is just too good to be true or useful to the comparison.

Table I additionally shows the total bandwidth of the LP fractional solution obtained by running SLP_1. Recall from Section IV-D that this solution provides a lower bound for the attainable bandwidth (modulo the choice of candidate filters) and the optimal bandwidth up to a small constant factor (if subscriptions are not first clustered into super-subscriptions). We see from the table that such solutions give much more meaningful lower bounds than $\mathsf{Gr}_{\neg\mathsf{l}}$. The fact that SLP_1 and $\mathsf{Gr}^\star$ perform within small factors (between 1.3 and 2.7) from the fractional solution is a good indication that they perform very well with respect to the optimal.

Table II further shows the comparison for workload sets #2 and #3. Here, the bandwidths of the LP fractional solutions indicate that $\mathsf{Gr}^\star$ performs well in both data sets. For workload set #2, the fact that the bandwidth of $\mathsf{Gr}^\star$ is smaller than the LP fractional solution automatically implies that the bandwidth achieved by $\mathsf{Gr}^\star$ matches the lower bound (within a small constant factor).

Delays: Figure 7(b). Here we show scatter plots of delay versus shortest path latency for selected algorithms for (IS:H, BI:H); the results are similar for other workloads in workload

TABLE I

BANDWIDTH COMPARISON (WORKLOAD SET #1)

Workload	Fractional solution	SLP_1	$\mathsf{Gr}^\star$	Gr
(IS:L, BI:L)	3.09E9	7.12E9	6.53E9	9.50E9
(IS:H, BI:L)	1.2E9	1.86E9	1.53E9	2.09E9
(IS:L, BI:H)	3.81E9	8.48E9	7.79E9	1.05E10
(IS:H, BI:H)	1.29E9	2.13E9	2.39E9	2.78E9

TABLE II

BANDWIDTH COMPARISON (OTHER WORKLOAD SETS)

Workload set	Fractional solution	SLP_1	$\mathsf{Gr}^\star$	$\mathsf{Gr}_{\neg\mathsf{l}}$
#2	1.01E7	1.37E7	8.5E6	220
#3	2.48E10	5.4E10	5.3E10	5.09E10

set #1 and for other workload sets. Both SLP_1 and $\mathsf{Gr}^\star$ are able to bound delay at 0.3 as required. $\mathsf{Closest}_{\neg\mathsf{b}}$ is expected to do well on delay, because it focuses exclusively on the network space. However, since $\mathsf{Gr}_{\neg\mathsf{l}}$ ignores the network space, it has trouble satisfying the latency constraints; subscribers near the publisher are especially vulnerable as they may be assigned to faraway brokers that blow up delays significantly.

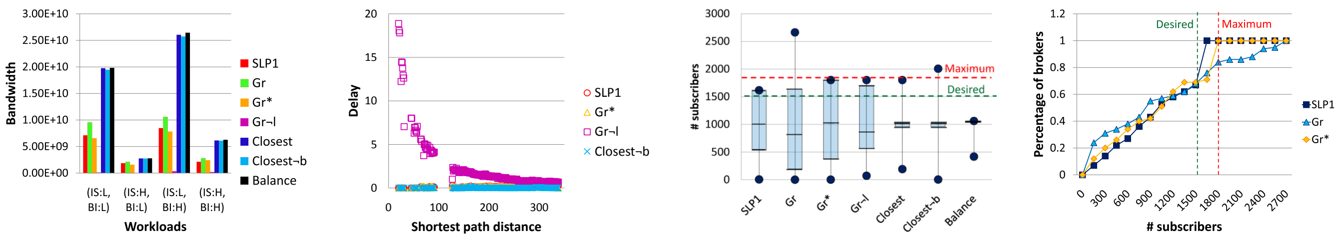
Broker Loads: Figures 7(c) and 7(d). Figure 7(c) shows the boxplot of broker loads for each algorithm for (IS:H, BI:H); the results are similar for other workloads in workload set #1. The two dashed horizontal lines show the maximum and desired load bounds corresponding to $\beta_{\max}$ and $\overline{\beta}$, respectively. As expected, Balance is the best; Closest also does well because the broker distributions roughly follow the subscriber distributions in our workloads; $\mathsf{Closest}_{\neg\mathsf{b}}$ is similar to Closest but some brokers may still be overloaded because $\mathsf{Closest}_{\neg\mathsf{b}}$ does not enforce load balance constraints. Keep in mind, however, that these algorithms achieve good load balance at the expense of huge bandwidth (Figure 7(a)). Other algorithms exhibit wider range of loads. As mentioned earlier, Gr is unable to satisfy the load balance constraints, but SLP_1, $\mathsf{Gr}^\star$, $\mathsf{Gr}_{\neg\mathsf{l}}$ do, with SLP_1 achieving a lbf close to the desired setting.

To have a closer look at the load distributions, we plot the cumulative distribution function (CDF) for selected algorithms in Figure 7(d). Gr, despite its best attempt at enforcing constraints, leaves more than 10% of the brokers overloaded.

The results are also similar for the other two workload sets. The maximum load of Gr exceeds $\beta_{\max}$ by 39% and 58% for workload sets #2 and #3, respectively.

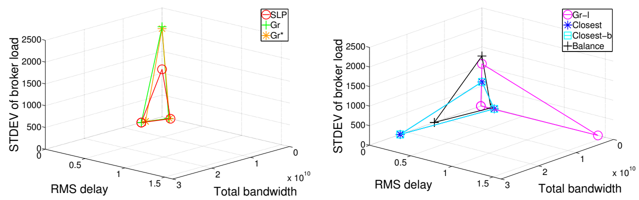
Solution Quality for a Multi-Level Broker Network. In the following, we test workload set #1 and have $100{,}000$ subscribers to assign to a multi-level network of 200 brokers, where each internal broker has a maximum out-degree of 15. We also adjust the constraints to see how well different algorithms cope with them. In the *tight latency* setting, we set the maximum delay to 0.2; to compensate, we set the desired and maximum lbfs to 7 and 8 (the minimum possible lbf is around 6). In the *loose latency* setting, we set the maximum delay to 1, and the desired and maximum lbfs to 1.3 and 1.5.

Overall Comparison: Figures 8(a) and 8(b). Similar to the results for a one-level network, algorithms that ignore the

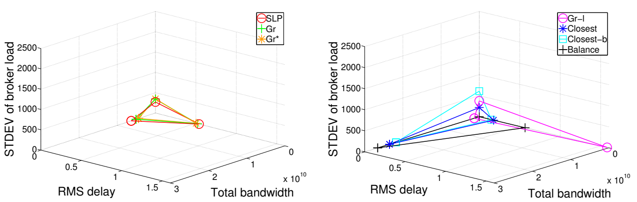


(a) Bandwidth (b) Delay distribution (c) Broker load distribution (d) Broker load CDF

Fig. 7. Detailed comparison (one-level network, workload set #1).

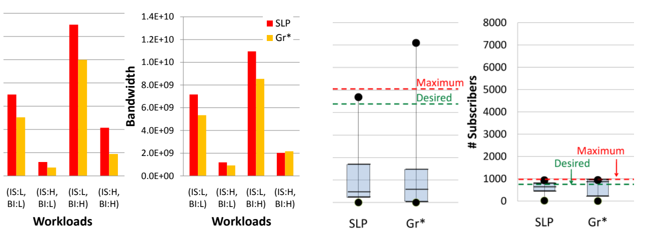


(a) *tight*



(b) *loose*

Fig. 8. Overall (multi-level network, workload set #1); *tight* and *loose* refer to the tight and loose latency settings, resp.



(a) Bandwidth (b) Broker load

Fig. 9. Other comparison (multi-level network, workload set #1); *tight* vs. *loose*

event space ($\mathsf{Closest}_{\neg b}$, $\mathsf{Closest}$, and $\mathsf{Balance}$) incur high bandwidth, while the algorithm that ignores the network space ($\mathsf{Gr}_{\neg l}$) produces long delays. Again, $\mathsf{Gr}_{\neg l}$'s bandwidth is too unrealistic to serve as a meaningful yardstick for other solutions. Therefore, we omit these algorithms in subsequent comparisons.

Under the loose latency setting, Gr and $\mathsf{Gr}^\star$ are comparable to SLP, and $\mathsf{Gr}^\star$ actually achieves slightly lower bandwidth than SLP. Under the tight latency setting, however, both Gr and $\mathsf{Gr}^\star$ fail to produce a feasible solution that satisfies the load balance constraints (like what happened to Gr for the one-level network). Since the solution quality of $\mathsf{Gr}^\star$ dominates that of Gr, we also omit Gr in subsequent comparisons.

Bandwidth: Figures 9(a). Interestingly, for all but one of the eight workloads, SLP underperforms $\mathsf{Gr}^\star$. One explanation is that subscribers have too few choices of brokers under the tight latency setting, and too many choices under the loose setting; in either case, SLP has little advantage over $\mathsf{Gr}^\star$. However, note that the comparison under the tight latency setting is misleading, because $\mathsf{Gr}^\star$ is unable to satisfy the load balance constraints, while SLP does. Under the loose latency setting, two algorithms actually have more similar performance.

Broker Loads: Figures 9(b). These figures show the results on (IS:L, BI:H). Regardless of the latency setting, SLP satisfies all constraints. On the other hand, $\mathsf{Gr}^\star$, despite its best effort, cannot enforce all load balance constraints under the tight latency setting. A closer look at the broker load distribution (not shown here) would reveal that more than 10% of the brokers are overloaded.

Effect of Filter Complexity. Figure 10 shows the effect of the filter complexity (α) on the total bandwidth of solutions by SLP, Gr, and $\mathsf{Gr}^\star$. The workload is (IS:H, BI:H), with a one-level network. As discussed in Section II, a larger α may reduce bandwidth, because multiple rectangles can summarize a set of subscriptions more precisely than a single rectangle. This effect is clear and similar for all three algorithms. At the lowest α settings of 1 and 2, SLP_1 is more vulnerable than Gr and $\mathsf{Gr}^\star$: a filter may consist of multiple faraway rectangles after rounding of the fractional solution; covering them with just one or two MEB may increase the filter volume dramatically. Overall, $\alpha = 3$ is reasonable for all algorithms; a larger α increases storage and processing overhead at a broker and its parent, and has diminishing effect on bandwidth.

Running Time of SLP. We measure the wall-clock time of running SLP on a Dell OptiPlex 960 desktop with Intel Core2 Duo CPU E8500 at 3.16GHz, 6144KB of cache, and 8GB of memory. The LP solver is CPLEX Version 10. A run with one million subscribers and 100 brokers in a single-level network takes about 23 hours. A run with one million subscribers and 200 brokers in a multi-level network takes about 4 hours (faster because each call to SLP_1 here involves far fewer than 100 brokers). Figure 11 shows how the number of subscribers impacts the running time of SLP.

In sum, for realistic problem sizes, SLP has manageable running time on mid-range hardware. While SLP is by no means a fast algorithm, its solution quality makes it well worthwhile, especially as a yardstick to gauge other algorithms.

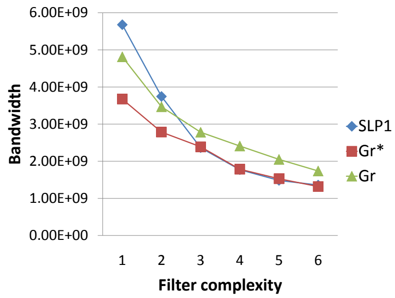


Fig. 10. Effect of filter complexity (one-level network).

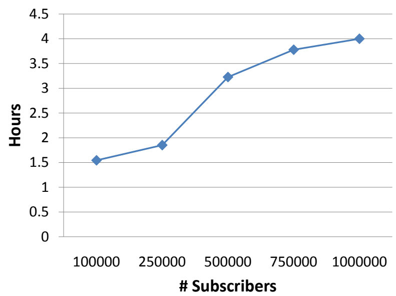


Fig. 11. Running time of SLP (multi-level network).

Other Results. We have experimented with other scenarios by varying constraints (e.g., bounding last-hop latency instead of path latency), workload characteristics (e.g., broker distribution in the network space), and choices of parameters for SLP (size of $|\mathcal{S}_b|$ (Section IV-A.2), number of super-subscriptions in candidate filter generation (Section IV-A.3), and threshold $\overline{\gamma}$ for the multi-level algorithm (Section V). These experiments reaffirm the relative robustness of SLP across constraints and workloads, and verify our settings of parameters. Because of limited space, see [15] for the results of these experiments.

Discussion. One take-way point from these experiments is that $\mathsf{Gr}^{\star}$ works well on many (though not all) workloads, including fairly realistic ones generated from statistics on Google Groups. What is more important, however, is what allows us to draw this conclusion. Solutions obtained by algorithms that ignore any performance criterion are not helpful—not only do they tend to fare terribly on criteria they ignore, but they also cannot offer meaningful bounds on what can be realistically achieved. On the other hand, our LP-based approach is a better yardstick for evaluating different algorithms. While we cannot guarantee the optimality of SLP_1, we have more assurance of its solution quality (Section IV-D) across problem instances. Furthermore, the fractional solution it produces gives us another indicator of optimality that is far more useful than, say, what $\mathsf{Gr}_{\neg\mathsf{l}}$ offers.

One might wonder if $\mathsf{Gr}^{\star}$ works well in general. It does not. We have already seen that it has trouble with load balance constraints under the tight latency setting. Furthermore, in [15], we show that there are several instances for which $\mathsf{Gr}^{\star}$ performs orders of magnitude worse than SLP. These examples further illustrate the importance of developing better yardsticks for evaluating algorithms for SA.

VII. Related Work

Dissemination network design for publish/subscribe has received much attention in the past few years. As discussed in Section I, some previous work considers either subscription similarity in the event space (e.g., [2]) or subscriber location in the network space [1] while ignoring the other aspect. Other performance objectives and constraints have also be considered in subscriber assignment. Shah et al. [22] maximize data fidelity. Tariq et al. [23] maximize the number of subscribers whose latency constraints are satisfied without violating bandwidth constraints.

Another line of research focuses on self-organizing, distributed algorithms that dynamically reconfigure the network topology to optimize specific measures. Baldoni et al. [24] minimize the number of hops and let subscribers be spread uniformly among brokers. Jaeger et al. [25] minimize total processing and communication costs (excluding last-hop latencies between brokers and subscribers). The distribution of subscribers to brokers is chosen probabilistically according to a random load value. Papaemmanouil et al. [5] present a general optimization framework that iteratively improves performance, starting by randomly attaching subscribers to a node. Understanding the robustness and global optimality of such algorithms has been challenging. We complement this line of research by providing a yardstick for evaluation that is computationally feasible over more realistic problem sizes.

Distributed stream processing is also related to our work. Stream processing systems process and aggregate data over a network of machines, and one key issue is how to optimally place query operators onto the set of machines (see [26] for overview and [27], [18] for more recent development). However, the number of queries involved in the operator placement problem is orders of magnitude smaller than the number of subscribers in the subscriber assignment problem.

There is a vast body of literature on network design in general. Problems that resemble ours to various extent include, for example, the minimum steiner tree problem, the weighted steiner tree packing problem, and content distribution network design. Additional discussion can be found in [15].

VIII. Conclusion and Future Work

In this paper we have presented SLP, a LP-based algorithm for SA, the subscriber assignment problem for wide-area content-based publish/subscribe. SLP considers the subscriber distribution in both event and network spaces to minimize bandwidth while satisfying latency and load balance constraints. To ensure its scalability to realistic problem sizes, SLP employs a suite of techniques, including LP relaxation, randomized rounding, coreset, sampling, and max-flow, to carefully reduce its complexity.

As a solution to the offline SA problem, SLP can be used for initial subscriber assignment and periodical re-optimization. More importantly, because of its better theoretical properties and robustness to workload variations, SLP serves as a reasonable yardstick for evaluating simpler heuristic algorithms across realistic workloads in both online and offline settings. Using this yardstick, we have shown that a simple and efficient greedy algorithm, $\mathsf{Gr}^\star$, works well for a number of workloads. Compared with previous work, we have pushed the sophistication and scale of evaluation workloads to new heights. While future work on improving the theoretical guarantees of such yardsticks is still needed, we hope researchers will find SLP and/or its ideas useful in evaluating algorithms for SA and similar problems, where the optimal solutions remain computationally elusive.

There are two immediate directions for future work. First, a principled approach is still much needed for the dynamic version of the subscriber assignment problem, where subscriptions come and go. Second, it would be good to drop the assumption that a broker tree is given in advance, and jointly optimize subscriber assignment, broker placement, as well as the dissemination network topology.

ACKNOWLEDGMENT

This work is supported by NSF grants CNS-05-40347, CCF-06-35000, IIS-07-13498, and CCF-09-40671, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, by an NIH grant 1P50-GM-08183-01, and by a grant from the U.S.–Israel Binational Science Foundation.

APPENDIX

Because of space constraints, see [15] for omitted proofs.

Lemma 1 (Size of coreset for filter assignment): There exists an ϵ-certificate $\mathcal{Q} \subseteq \mathcal{S}$ of size $O((n\ln(\Delta/\epsilon))^{2dn})$, where Δ is proportional to the ratio of the volume of $\mathrm{MEB}(\mathcal{S})$ to the volume of the smallest subscription.

Lemma 2 (Number of iterations): If no certificate is found after $4g\ln(|\mathcal{S}|/g)$ iterations, the size of a certificate must be greater than g.

Lemma 3 (Probability of valid round): Let $\mathcal{Q}$ be a random sample of size $cg\ln g$, and Φ be the set of filters assigned to $\mathcal{B}$ to cover $\mathcal{Q}$. Let $\mathcal{S}'$ be a set of subscriptions not covered by Φ. The probability that $W(\mathcal{S}') > \epsilon W(\mathcal{S})$ is at most 1/2 by choosing a sufficently large constant c.

Lemma 4 (Goodness of candidate filters): Let $\mathcal{R}^*$ be the set of $O(k^{2d})$ rectangles, where each rectangle is the minimum enclosing box of a subset of the k subscriptions. For each rectangle $R \in \mathcal{R}^* \setminus \mathcal{R}$, there exists a rectangle $R' \in \mathcal{R}$, such that $R \subset R'$ and $\mathrm{Vol}(R') \leq 4^d \,\mathrm{Vol}(R)$.

Theorem 1 (Solution quality of SLP_1): With probability at least $1/2$, the algorithm, without the optional step of merging subscriptions into super-subscriptions, returns a subscriber assignment with the following properites: 1) The expected bandwidth is at most $2\ln|\mathcal{S}_a|OPT$, 2) latency and nesting constraints properties are strictly enforced, and 3) the expected filter complexity constraints are violated by a factor of at most $2\ln|\mathcal{S}_a|$.

REFERENCES

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, "Matching events in a content-based subscription system," in *PODC* 1999, pp. 53–61.
[2] Y. Diao, S. Rizvi, and M. J. Franklin, "Towards an internet-scale XML dissemination service," in *VLDB* 2004, pp. 612–623.
[3] O. Papaemmanouil and U. Cetintemel, "SemCast: Semantic multicast for content-based data dissemination," in *ICDE* 2005, pp. 242–253.
[4] P. Rao, J. Cappos, V. Khare, B. Moon, and B. Zhang, "Net-χ: unified data-centric internet services," in *NETDB* 2007, pp. 1–6.
[5] O. Papaemmanouil, Y. Ahmad, U. Cetintemel, J. Jannotti, and Y. Yildirim, "Extensible optimization in an overlay dissemination trees," in *SIGMOD* 2006, pp. 611–622.
[6] A. Yu, P. K. Agarwal, and J. Yang, "Generating wide-area content-based publish/subscribe workloads," in *NETDB* 2009.
[7] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: a decentralized network coordinate system," in *SIGCOMM* 2004, pp. 15–26.
[8] J. Ledlie, P. Pietzuch, and M. Seltzer, "Stable and accurate network coordinates," in *ICDCS* 2002, p. 74.
[9] T. S. E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *IEEE INFOCOM* 2002, pp. 170–179.
[10] V. V. Vazirani, *Approximation Algorithms*. Berlin Heidelberg: Springer-Verlag, 2003.
[11] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, "Geometric approximation via coresets - survey," in *Combinatorial and Computational Geometry (MSRI publication)*, 2005, vol. 52, pp. 1–30.
[12] K. L. Clarkson, "Las vegas algorithms for linear and integer programming when the dimension is small," *J. ACM*, vol. 42, no. 2, pp. 488–499, 1995.
[13] H. Brönnimann and M. T. Goodrich, "Almost optimal set covers in finite vc-dimension," *Discrete and Computational Geometry*, vol. 14, pp. 463–479, 1995.
[14] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan, "Approximation algorithms for k-line center," in *ESA* 2002, pp. 54–63.
[15] A. Yu, P. K. Agarwal, and J. Yang, "Subscriber assignment for wide-area content-based publish/subscribe," Duke University, Tech. Rep., 2010. http://www.cs.duke.edu/dbgroup/papers/2010-YuAgarwalEtAl-sub_assign.pdf
[16] V. Bilò, I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos, "Geometric clustering to minimize the sum of cluster sizes," in *ESA* 2005, pp. 460–471.
[17] V. Ramasubramanian, R. Peterson, and E. G. Sirer, "Corona: A high performance publish-subscribe system for the World Wide Web," in *NSDI* 2006, pp. 15–28.
[18] O. Papaemmanouil, U. Çetintemel, and J. Jannotti, "Supporting generic cost models for wide-area stream processing," in *ICDE* 2009, pp. 1084–1095.
[19] S. Voulgaris, E. Rivire, A. M. Kermarrec, and M. van Steen, "Sub-2-sub: Self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks," in *IPTPS* 2005.
[20] A. Machanavajjhala, E. Vee, M. Garofalakis, and J. Shanmugasundaram, "Scalable ranked publish/subscribe," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 451–462, 2008.
[21] S. Bianchi, P. Felber, and M. Gradinariu, "Content-based publish/subscribe using distributed r-trees," *Euro-Par*, vol. 4641, pp. 537–548, 2007.
[22] S. Shah, K. Ramamritham, and C. V. Ravishankar, "Client assignment in content dissemination networks for dynamic data," in *VLDB* 2005, pp. 673–684.
[23] A. Tariq, B. Koldehofe, G. Koch, and K. Rothermel, "Providing probabilistic latency bounds for dynamic publish/subscribe systems," in *KiVS 2009*, pp. 155–166.
[24] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito, "Efficient publish/subscribe through a self-organizing broker overlay and its application to SIENA," *The Computer Journal*, vol. 50, no. 4, pages 444–459, 2007.
[25] M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann, "Self-organizing broker topologies for publish/subscribe systems," in *SAC* 2007, pp. 543–550.
[26] G. T. Lakshmanan, Y. Li, and R. Strom, "Placement strategies for internet-scale data stream systems," *IEEE Internet Computing*, vol. 12, no. 6, pp. 50–60, 2008.
[27] Y. Zhou, B. C. Ooi, and K.-L. Tan, "Disseminating streaming data in a dynamic environment: an adaptive and cost-based approach," *The VLDB Journal*, vol. 17, no. 6, pp. 1465–1483, 2008.