Maximizing the Number of Satisfied Subscribers in Pub/Sub Systems Under Capacity Constraints

Vinay Setty[1], Gunnar Kreitz[2,3], Guido Urdaneta[2], Roman Vitenberg[1], and Maarten van Steen[4]

[1]University of Oslo, Norway, {vinay,romanvi}@ifi.uio.no
[2]Spotify, Stockholm, Sweden, {gkreitz,guidou}@spotify.com
[3]KTH – Royal Institute of Technology, Stockholm, Sweden, gkreitz@kth.se
[4]VU University and The Network Institute, Amsterdam, The Netherlands, steen@cs.vu.nl

***Abstract*— Publish/subscribe (pub/sub) is a popular communication paradigm in the design of large-scale distributed systems. A provider of a pub/sub service (whether centralized, peer-assisted, or based on a federated organization of cooperatively managed servers) commonly faces a fundamental challenge: given limited resources, how to maximize the satisfaction of subscribers?**

We provide, to the best of our knowledge, the first formal treatment of this problem by introducing two metrics that capture subscriber satisfaction in the presence of limited resources. This allows us to formulate matters as two new flavors of maximum coverage optimization problems. Unfortunately, both variants of the problem prove to be NP-hard. By subsequently providing formal approximation bounds and heuristics, we show, however, that efficient approximations can be attained. We validate our approach using real-world traces from *Spotify* and show that our solutions can be executed periodically in real-time in order to adapt to workload variations.

I. Introduction

We are witnessing an increasingly widespread use of the publish/subscribe (pub/sub) communication paradigm in the design of large-scale distributed systems. Pub/sub is regarded as a technology enabler for 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 online delivery of notifications due to social interaction [1], application integration [2], financial data dissemination [3], RSS feed distribution and filtering [4], [5], and business process management [6]. As a result, many industry standards have adopted pub/sub as part of their interfaces. Examples of such standards include WS Notifications, WS Eventing, and the Active Message Queuing Protocol.

In this paper, we focus on the topic-based pub/sub model. In a topic-based system, publication events are associated with topics, and subscribers register their interest in receiving all events published to certain topics.

While traditional pub/sub implementations are either centralized or based on a federated organization of cooperatively managed servers, an increasingly higher number of pub/sub applications are being deployed in P2P environments [7]. In particular, the pub/sub service at *Spotify* [1] is suitable for a peer-assisted implementation, in line with the reported peer-assisted implementation of other *Spotify* services such as music streaming [8]. In a peer-assisted implementation, a limited number of servers provide a guaranteed high-quality service to a subset of pub/sub subscribers while the rest of subscribers receive notifications through peers, thereby getting a best-effort service that works convincingly well in practice. The part of the workload assigned to a server is dictated by maximizing server utilization as well as the overall quality of service given to the subscribers.

In this paper, we provide the first formal treatment of this subject to the best of our knowledge. Specifically, we introduce a measure of subscriber satisfaction that lends itself to a large class of pub/sub notification services where (a) publication-event message delivery is best effort: reliable delivery is desirable but it is not mandatory to deliver all notifications, and (b) every notification is intended to be read by a human user, so having a cumulative delivery rate to a particular subscriber above a certain threshold might not bring significant benefit to the user experience. For example, many applications where notifications are generated due to social interaction fall into this class of pub/sub services: following the tweets of selected users in Twitter, monitoring updates to the profiles of user's friends in Facebook, or receiving instant notifications related to favorite artists and albums in *Spotify*. According to our satisfaction metric, we consider a subscriber satisfied in such applications if and only if the user receives all notifications of interest at a configurable minimum threshold delivery rate. We also provide a fractional satisfaction metric: If a subscriber receives fewer notifications than desired, the satisfaction of the subscriber is defined as a fraction of the actual and desired number of notifications.

Then, we introduce a principal optimization problem: given a server with a limited capacity, and a workload consisting of (a) a set of topics each with its own publication event rate, and (b) a set of subscribers with their interests; the goal is to *maximize* the number of subscribers with their cumulative delivery rate of publications to match a certain threshold (satisfaction metric), while

978-1-4799-3360-0/14/$31.00

respecting the *budget* constraint imposed by the limited resources of the back-end servers. We define two distinct flavors of the problem: a **"Budgeted Maximum Multiset Multicover"** (*B3M*) and **"Fractional Budgeted Maximum Multiset Multicover"** (*F-B3M*) using the binary and fractional satisfaction metrics, respectively. We prove that both flavors are NP-Hard. We reduce *B3M* from the Densest-k-Subgraph (*DkS*) problem [9], a new way to reduce max-cover problems. We also show that, while *B3M* does not admit a Polynomial-Time Approximation Scheme (PTAS) unless NP has randomized algorithms that run in sub-exponential time, *F-B3M* has a polynomial-time approximation algorithm with a guaranteed constant ratio of $\frac{1}{2}\left(1-\frac{1}{e}\right)$. Furthermore, we derive an upper bound for the optimal solution of each problem.

We evaluated the proposed heuristics for *B3M* and *F-B3M* using a large-scale real data set from the pub/sub system of *Spotify*. We show that the heuristics provide an approximation of at least 0.7 for both problems, for the given dataset, using the derived upper bound on the optimal solution as the baseline. Finally, we propose various optimizations to make the heuristics more efficient. We show that the heuristics run in less than 30 seconds for workloads with over a million topics, and in less than one second in most realistic scenarios.

II. Motivating Application Scenario and Proposed Pub/Sub Architecture

A pub/sub system typically consists of publishers that publish messages at one end and subscribers that receive publication messages asynchronously at the other end. A pool of servers called *brokers* enable asynchronous communication between them. Use of pub/sub to notify events generated due to social interaction is becoming increasingly popular [1], [10]. Typically, in these systems notifications are intended to be read by human users and having a cumulative delivery rate to a particular user above a certain threshold will not always bring much benefit. For example, in some social-networking mechanisms such as tweet feed in Twitter or friend feed in Facebook or friend feed and artist updates in *Spotify*, users often ignore notifications beyond some threshold. In such cases, spending precious resources on delivering every single notification (Friend-feed or Tweet) beyond that threshold might be wasteful. In this case, the workload that does not increase the satisfaction of subscribers can be simply dropped or offloaded to a lower-cost external system (such as a peer-to-peer network). As we show in Section III-A and III-B, the problem of selecting a subset of the workload in such a way as to maximize subscriber satisfaction while respecting the back-end capacity is a challenging optimization problem.

In this paper, we propose a methodology to select a fraction of the pub/sub workload such that this fraction is within the capacity of a back-end service with limited resources, while user satisfaction is maximized. This approach can help system managers to deal with the trade-off between deploying additional hardware and satisfying more users. It can also be used as a mechanism to drop or divert part of the pub/sub workload to an external lower-cost system with lower quality of service, such as a pool of lower-reliability servers, or a set of computers belonging to end users (peers) forming a peer-to-peer network.

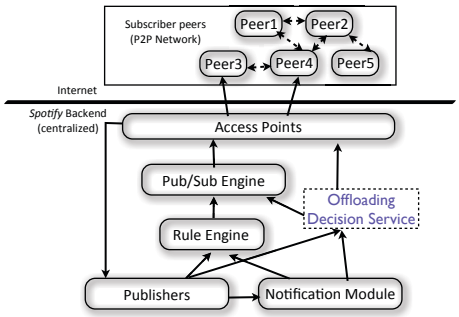

Fig. 1. Proposed Peer-assisted architecture for *Spotify* pub/sub

To facilitate the offloading of the workload we propose a service called *Offloading Decision Service* (ODS). In order to perform its work, the ODS divides the total pub/sub load on a per-topic basis and then decides for each topic whether the topic can be managed by the back-end service without exceeding the capacity. In this context, managing a topic means taking care of delivering the corresponding topic events for all subscribers of that topic.

The rationale for this design decision is that we believe that organizing the pub/sub load at this granularity level greatly simplifies system design compared to an approach based on dealing with each (topic, subscriber) subscription pair individually. While offloading at (topic, subscriber) granularity may be beneficial, it poses additional overhead to the pub/sub system and the ODS, making the offloading more complicated and expensive.

We now show the benefits of the ODS in the context of practical pub/sub systems designed for social interaction.

A. Social interaction among Spotify users

Spotify uses a pub/sub system to facilitate social interaction among its users. A *Spotify* user can follow friends (from Facebook or native to *Spotify*), artists and playlists[1]. The pub/sub system delivers the friend-feed, artist updates, and playlist updates to the appropriate *Spotify* users. The *Spotify* pub/sub system is implemented as a back-end service running in *Spotify*'s data centers. Details about *Spotify*'s pub/sub system have been presented in [1]. With the ever-growing user base of *Spotify* it is crucial for its pub/sub to scale accordingly. Typically, such services are scaled horizontally by deploying new hardware. In this paper, we provide a tool that can help system managers to estimate the amount of user satisfaction that can be achieved with existing resources, and estimate how it can be improved with additional hardware. We also show how the existing *Spotify* pub/sub architecture can be extended to divert part of the pub/sub workload to a P2P network by solving the proposed optimization problems, in line with the existing peer-assisted streaming solution already used by *Spotify*.

[1] user-created collections of music tracks

Fig. 1 shows our proposed peer-assisted pub/sub architecture. In a peer-assisted architecture, part of the load that is normally managed by a server in a classical client/server architecture, is managed by clients themselves, which act as servers towards other clients, and are referred to as peers. This approach has the advantage that it can reduce implementation costs, and can potentially scale easily with respect to the number of users, since peers bring with them an amount of resources that is proportional to the load the system has to handle.

As shown in Fig. 1, the ODS has access to Publishers and the Notification Module of the pub/sub architecture of *Spotify* to collect statistics about publication event rates and topic popularity. Depending on the satisfaction metric used, ODS will solve *B3M* or *F-B3M*, using collected statistics and the heuristics presented in Sections IV and V. The ODS then instructs the pub/sub engine to consider the list of topics it has found to maximize the subscriber satisfaction for real-time delivery of publications using its pool of brokers, while the remaining topics are offloaded to the P2P network. The ODS constantly monitors changes to the publication event rates as well as subscriptions and unsubscriptions and uses these updated statistics to periodically recompute the solutions for *B3M* or *F-B3M* to maximize the subscriber satisfaction. Therefore, an additional requirement for the ODS is that it should employ light-weight algorithms that can be executed relatively quickly. In this regard, we propose efficient algorithms to solve *B3M* or *F-B3M* in Section IV and V and validate them in Section VI to show that they can be executed in real-time for real traces from *Spotify*.

B. Cloud-based peer-assisted microblogging service

In [10], Cuckoo, a new Twitter-like microblogging system that offloads the workload from the cloud to a P2P network is proposed. However, the offloading technique is arbitrary and hence it may result in under utilization of the cloud resources. In addition, Cuckoo could benefit from our definition of satisfaction metrics to deal with overwhelming event rates of the topics related to news media. The Cuckoo design relies on offloading the topics with low publication rate and few subscribers to the P2P network. While this is proven to reduce the load on the cloud, we believe more can be achieved with the same cloud resources by using a more sophisticated strategy to select what to offload. In this paper we formalize this problem and provide approximation algorithms that could be applied in Cuckoo.

It is worth noting that application of the ODS is not limited to the two scenarios described above. It is not hard to see the applicability of the ODS in any pub/sub system with limited resources. In the future, we plan to design and implement a generic pub/sub framework built around the ODS, especially to facilitate peer-assisted pub/sub. However, in this paper we focus on designing and evaluating efficient algorithms to solve *B3M* and *F-B3M*.

III. Problem Definitions

The two QoS metrics mentioned in Section I prompt problems that are similar in nature but very different in hardness, as we show in Sections IV and V. In the first QoS metric, we are interested in maximizing the number of subscribers receiving at least τ (satisfaction threshold) events related to them from the back-end service. A subscriber is considered satisfied if and only if at least τ relevant events are received. This definition of user satisfaction is suitable for applications with events that are relatively infrequent but important for the user. *Spotify* updates about favorite albums and artists fall in this category. In this regard, we define a problem coined *Budgeted Maximum Multiset Multicover* (*B3M*) in this section. In Section IV we analyze the hardness of *B3M* and propose a feasible heuristic.

In the second QoS metric we quantify the amount of benefit towards the satisfaction of a subscriber with a fraction of cumulative events delivered to a subscriber relative to the given satisfaction threshold of τ. The goal is to maximize the sum of fractional benefits of individual subscribers of the topics set to be served by the back-end servers. This definition is appropriate for applications where events are frequent but of relatively low importance. An example would be *Spotify*'s updates about the activities of the friends of each given user. In this regard we define the *Fractional Budgeted Maximum Multiset Multicover* (*F-B3M*) problem. In Section V we analyze the hardness of *F-B3M* and propose a feasible heuristic that also gives a guarantee on the quality of the output.

In both flavors of the problem, we want to ensure that the computational and communication costs to serve the events needed to maximize the number of satisfied subscribers does not exceed a given limit on the capacity of the resources at the back-end service.

Before we define the problem more formally, we introduce the following notations:

T **:** A collection of l *topics* $\{t_1, t_2, ..., t_l\}$ in the system.
V **:** A collection of n *subscribers* $\{v_1, v_2, ..., v_n\}$ participating in the pub/sub system. A subscriber can subscribe to one or more topics from T. Subscribers in a typical pub/sub system are generally end-user applications (e.g. *Spotify* client software).
T_v **:** The *interest* of subscriber v, that is, the set of topics subscribed by v.
Int **:** The collection of interests $\{T_{v_1}, T_{v_2}, ..., T_{v_n}\}$ for all subscribers in V.
ev_t **:** *Event rate* of the publications generated for a topic t, that is, mean of events published to topic t during a given period (e.g., per minute or per hour). Without loss of generality, we assume that $ev_t > 0$. When we say 'event' in the rest of the paper we mean a publication-event message generated by the back-end service for a topic intended for all subscribers of the topic.
τ **:** A system parameter that represents the *satisfaction threshold* for a subscriber. It is defined as a constant specifying the number of events to be delivered to a subscriber by the back-end service in order for the subscriber to be considered satisfied. The period over which the events are to be delivered is the same as the time unit of ev_t.
τ_v **:** Subscriber-specific satisfaction threshold. In practice, the total event rate of the topics subscribed to by a subscriber is sometimes less than τ. In such cases we

need to serve all the events the subscriber is interested in to meet the satisfaction threshold. It is mathematically expressed as follows: $\tau_v = min(\tau, \sum_{t \in T_v} ev_t)$.
V_t **:** $V_t \subseteq V$ is a non-empty set of subscribers to topic t. Given *Int*, V_t can be derived trivially.
$cost(t)$ **:** Represents the non-zero cost of serving a topic t by the back-end service. We say that the cost of a topic is *normalized* if it costs 1 per event sent by the server to each subscriber of the topic and hence, *normalized* cost is defined as $cost(t) = ev_t \cdot |V_t|$.
$\mathcal{C}$ **:** Capacity of the back-end service. A constant to quantitatively represent the amount of resources available to the back-end service. $\mathcal{C}$ has same unit as *cost*.
$\mathcal{S}$ **:** Solution ($\mathcal{S} \subseteq T$). It is a set of topics that can be served by the back-end service with a cost that does not exceed a given resource constraint expressed by the constant $\mathcal{C}$.
$\sigma(\mathcal{S})$ **:** Represents the sum of the satisfaction for all subscribers, given a potential solution $\mathcal{S}$. We want to maximize this function.

A. The problem of *Budgeted Maximum Multiset Multicover (*B3M*)*:

Given an instance of T, V and their interests *Int*, the goal of the $B3M(T, V, ev, cost, Int, \tau, \mathcal{C})$ problem is to find $\mathcal{S} \subseteq T$ so as to maximize the objective function defined below:

$$\text{Maximize } \sigma(\mathcal{S}) = \sum_{v \in V} f(v), \text{ subject to } \sum_{t \in \mathcal{S}} cost(t) \leq \mathcal{C} \quad (1)$$

$f(v)$ is a function that indicates if subscriber v is receiving a number of events that meets the satisfaction threshold:

$$f(v) = \begin{cases} 1 & \text{if } \sum_{\{t \in \mathcal{S} \cap T_v\}} ev_t \geq \tau_v \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

The first condition in the Equation (2) is the case when a subscriber v is receiving publication events at a rate not lower than τ_v. In order for v to contribute to the objective function $f(v)$, the solution $\mathcal{S}$ must include enough topics subscribed by v with a total event rate of at least τ_v.

B. The problem of *Fractional Budgeted Maximum Multiset Multicover (*F-B3M*)*:

We now define a relaxed version of the *B3M* problem in which we quantify the satisfaction relative to the number of events covered for a subscriber v out of τ_v events. Given an instance of T, V and their interests *Int*, the goal of the *F-B3M*$(T, V, ev, cost, Int, \tau, \mathcal{C})$ problem is to find $\mathcal{S} \subseteq T$ so as to maximize the sum of the fractions for all the subscribers.

$$\text{Maximize } \sigma(\mathcal{S}) = \sum_{v \in V} g(v), \text{ subject to } \sum_{t \in \mathcal{S}} cost(t) \leq \mathcal{C} \quad (3)$$

$g(v)$ is the fraction of events subscriber v receives, and it is defined as:

$$g(v) = \begin{cases} 1 & \text{if } \sum_{\{t \in \mathcal{S} \cap T_v\}} ev_t \geq \tau_v \\ \frac{\sum_{\{t \in \mathcal{S} \cap T_v\}} ev_t}{\tau_v} & \text{Otherwise} \end{cases} \quad (4)$$

The difference between *B3M* and *F-B3M* lies in the definition of the satisfaction metrics in Equation (2) and Equation (4) respectively. In Equation (2) the satisfaction is defined in a binary fashion i.e. the satisfaction is 0 when less than τ_v events are received by the subscriber and 1 otherwise. On the other hand in Equation (4) a fraction of events received up to τ_v is considered instead of a binary 1 or 0. This subtle difference makes the two problems fundamentally different in terms of difficulty of solving. We explore this in detail in Sections IV and V.

IV. Hardness of *B3M* and its Solution Approach

In this section we prove that *B3M* is NP-Hard and we also show that *B3M* has no Polynomial-Time Approximation Scheme (PTAS). We further propose an algorithm to give an upper bound on *B3M* instances. We use this bound to evaluate a greedy heuristic we propose in Section VI-B.

A. Hardness of B3M problem

To establish the hardness of *B3M* we prove that the well-known hard problem of Densest-k-Subgraph (*DkS*) can be reduced to a special case of *B3M*. We now define the *DkS* problem and an auxiliary unit-cost version of *B3M*.

Definition IV.1 (Densest-k-Subgraph)**.** Given an undirected graph $G(U, E)$ the Densest-k-Subgraph ($DkS(U, E, k)$) problem on G is the problem of finding a subset $U' \in U$ of vertices of size $|U'| = k$ with the maximum induced average degree. The average degree of the optimal subgraph is $2|E(U')|/k$. Here $|E(U')|$ denotes the number of edges in the subgraph induced by U'.

The *DkS* problem can be proven to be NP-Hard by reduction from the Max-Clique problem [11]. In [9] it has been shown that *DkS* is also NP-Hard even when restricted to a maximum degree of 3. The best known approximation algorithm achieves a ratio of $O(n^{1/4+\epsilon})$ and runs in $2^{n^{O(1/\epsilon)}}$ time, for any $\epsilon > 0$ [12]. On the other hand, it is known that *DkS* does not admit a PTAS [13].

Definition IV.2 (UC-*B3M*)**.** We define an auxiliary problem coined Unit-Cost-*B3M*(*UC-B3M*) which is a restricted version of *B3M*. We define *UC-B3M* to be an instance of *B3M* with unit cost for all the topics $\forall t \in T : cost(t) = 1$ and unit event rate $ev_t = 1$, each subscriber subscribes to exactly two topics $\forall v \in V : |T_v| = 2$, no two subscribers subscribe to same set of topics $\forall v_1 \neq v_2 : T_{v_1} \neq T_{v_2}$ and the satisfaction threshold $\tau_v = 2$.

Lemma IV.3. UC-B3M *is NP-Hard*

Proof: Given an instance of $DkS(U, E, k)$ we construct an instance of *UC-B3M*$(T, V, ev, cost, Int, \tau, \mathcal{C})$ in the following way: we take T with topics that one-to-one correspond to the vertices in the set U. We take V to one-to-one correspond to the edges in the set E. We build *Int* from the edges incident on the vertices. For example, V_t corresponds to the edges incident on the corresponding vertex in U. We set $\mathcal{C} = k$. We now prove that there is an induced subgraph of $A(U', E')$ with average degree δ and exactly k vertices if and only if there is a solution $\mathcal{S}$ to *UC-B3M* with value at least $|E(U')|$ (i.e., the total number of edges in the induced subgraph).

To see this, we observe that a subscriber in our *UC-B3M* instance only contributes to the objective function if both of her topics are included in $\mathcal{S}$. This precisely corresponds to the condition if and only if that exact edge with the vertices corresponding to those two topics is in the induced subgraph of the *DkS* instance. We can, without loss of generality, assume that $\mathcal{S}$ contains precisely k topics as the cost of each topic is 1 and the objective function is non-decreasing in the number of selected topics.

As we know that *DkS* is NP-Hard [11], it follows that *UC-B3M* is NP-Hard too. ∎

Theorem IV.4. B3M *is NP-Hard.*

Proof: UC-B3M is a special case of *B3M*. From Theorem IV.3 we know that *UC-B3M* is NP-Hard and hence *B3M* is NP-Hard too. ∎

Corollary IV.5. *Assuming* NP $\not\subseteq \cap_{\epsilon>0}$BPTIME$(2^{n^\epsilon})$*, there is no Polynomial-Time Approximation Scheme (PTAS) for* B3M*.*

Proof: The statement follows directly for *UC-B3M* from the reduction given in Lemma IV.3 together with a result by Khot [13] saying that unless NP has randomized algorithms that run in sub-exponential time (more formally: NP $\subseteq \cap_{\epsilon>0}$BPTIME$(2^{n^\epsilon})$) there is no PTAS for *DkS*. As *UC-B3M* is a special case of *B3M*, the statement also holds for *B3M*. ∎

B. Greedy heuristic for B3M

In the greedy algorithm to solve *B3M*, in each iteration of the algorithm, a topic t is chosen so as to maximize the ratio between its benefit and its cost. The benefit of a topic is quantified by its total contribution towards the objective function relative to the already chosen topics $\mathcal{S}'$. This is done for each subscriber of a topic in a for loop (lines 2 to 5 of Algorithm 1). We define the contribution of a topic t by considering the following scenarios: Adding t to the solution $\mathcal{S}'$ (a) guarantees to deliver τ_v events to its subscriber v (b) contributes partially to the target τ_v events for its subscriber v. In the first case, the contribution is of value 1. In the second case, the contribution is the ratio between ev_t and the remaining events needed to reach the target τ_v (computed in line 3). The intuition behind this choice is to give higher priority to a topic that satisfies a subscriber and hence, directly contributes to the objective function. On the other hand, a topic contributing partially to the satisfaction of its subscriber is given relatively lower priority. This step is repeated for each subscriber of the topic t and the contribution is accumulated as a sum (line 5). Finally, in line 6 the total contribution is divided by the topic's cost to return the benefit-cost ratio.

The pseudocode of the greedy algorithm to solve *B3M* is sketched in Algorithm 2 and the greedy strategy is to choose a topic that maximizes the objective function. In lines 2 and 3 an array containing the benefit-cost ratio of the individual topics is initialized using Algorithm 1. In practice, this array can be a max-heap structure optimized for obtaining elements with maximum value.

Algorithm 1: Heuristic value of topic t given partial solution $\mathcal{S}'$

```
1  GetHeuristicB3M(t, ev, cost(t), Int, S', τ)
   Input: t, ev, cost(t), Int, S', τ
   Data: h ← 0 : Heuristic value
   rem_v ← 0: Events remaining to make user v happy
2  foreach {v ∈ V_t} do
3      rem_v ← τ_v − Σ_{t'∈S'∩T_v} ev_{t'}
4      if rem_v > 0 then
5          h ← h + min(1, ev_t / rem_v)
6  return h / cost(t)
```

Algorithm 2: Greedy solution for ***B3M***

```
1  GreedyB3M(T, V, ev, cost, Int, τ, C)
   Input: T, V, ev, cost, Int, τ, C
   Data: A : Array of size l
   Result: S' ← ∅ : Output set of topics
2  foreach t ∈ T do
3      A[t] ← GreedyHeuristicB3M(t, ev, cost(t), Int, S', τ)
4  while T ≠ ∅ do
5      t ← argmax_{t'∈T} A[t']
6      if A[t] = 0 then
7          break
8      if cost(t) + Σ_{t'∈S'} cost(t') ≤ C then
9          S' ← S' ∪ {t}
10         foreach {t' : V_t ∩ V_t' ≠ ∅ ∧ t' ∉ S'} do
11             A[t'] ← GreedyHeuristicB3M(t', ev, cost(t'), Int, S', τ)
12     T ← T \ {t}
13 return S'
```

Algorithm 3: Upper bound for ***B3M*** with normalized topic costs

```
1  GetUpperBound(V, T, ev, Int, C, τ)
   Input: V, T, ev, Int, C, τ
   Data: C : Array of size n
   csubs ← ∅ : Set of subscribers covered
2  foreach {v ∈ V} do
3      C[v] ← max(τ_v, min_{t∈T_v} ev_t)
4  while V ≠ ∅ do
5      v ← argmin_{v'∈V} C[v']
6      if C[v] + Σ_{v'∈csubs} C[v'] ≤ C then
7          csubs ← csubs ∪ {v}
8      V ← V \ {v}
9  return |csubs|
```

A topic that maximizes the benefit-cost ratio in each iteration is selected in Algorithm 5. The topic is added to the solution if its addition keeps the cost of the solution within the budget. Otherwise the topic is ignored. If the topic is added to the solution, the benefit-cost ratio of all the topics not selected so far are updated based on the current solution set $\mathcal{S}'$ (lines 10 and 11). $V_t \cap V_{t'}$ is the set of subscribers common to subscribers of t and subscribers of t'. The algorithm terminates when it has considered all the available topics, or when all subscribers have been covered in which case the benefit-cost ratio of all the topics would be 0 (line 7).

Theorem IV.6. *The run time complexity of Algorithm 2*

is $\mathrm{O}\big(|T|^2(|V| + \log|T|)\big)$.

Proof: Refer to Appendix A of [14]. ∎

Theorem IV.6 gives the worst-case run time complexity, the cost being dominated by updating the cost for all topics in lines 10 and 11 of Algorithm 2 when a topic is added to the solution. We remark that in practice, the code runs significantly faster than this bound would imply. One of the reasons being that the number of updates is bounded by $max_{t \neq t'}|V_t \cap V_{t'}|$, which is usually significantly lower than $|T|$.

We now turn to the subject of computing an upper bound on the optimal solution. For this analysis, we only consider the case when the cost function is normalized, i.e., $cost(t) = ev_t \cdot |V_t|$.

Theorem IV.7. *Given an instance* B3M$(T, V, ev, cost, Int, \tau, \mathcal{C})$ *where the costs are normalized, for any solution* $\mathcal{S}$ *it holds that:*

$$\sigma(\mathcal{S}) \leq \max\left(|V'| : \sum_{v \in V'} \max\left(\tau_v, \min_{t \in T_v} ev_t\right) \leq \mathcal{C}\right),$$

where $V' \subseteq V$.

Proof (Sketch): With normalized costs, one can see that the amortized cost to cover each subscriber v is at least τ_v. The cost is also bounded by the lowest event rate of any event in which the subscriber is interested. For detailed proof please refer to Appendix B in [14]. ∎

Theorem IV.7 presents a way to compute an upper bound on the optimal solution. Since Algorithm 2 gives an unbounded approximation ratio, we make use of Theorem IV.7 to evaluate how well our proposed heuristic performs on real-world inputs (see Section VI-B). This theorem can be readily turned into an algorithm as shown in Algorithm 3. In lines 2 and 3 the minimum cost to consider a subscriber satisfied is initialized in an array. Then, in each iteration the subscriber with the least cost is selected until there is no more budget left to cover more subscribers (lines 4 to 8). Finally, the number of selected subscribers is returned as the upper bound for the optimal solution (line 9).

V. Hardness of *F-B3M* and its Solution Approach

In this section we analyze the hardness of *F-B3M*. Comparing to the results we obtained for *B3M*, the direct reduction we did from Densest-k-Subgraph no longer works as in that case it is imperative that we are not "paid" for a partially satisfied subscriber. This also means that the approximation-resistance results obtained for *B3M* do not translate. For *F-B3M*, we are instead able to give a greedy approximation algorithm with an approximation ratio of $\frac{1}{2}\left(1 - \frac{1}{e}\right)$. *F-B3M* is still NP-Hard, which we first prove by a reduction from the (unweighted) *Maximum Coverage problem* [15].

Theorem V.1. F-B3M *problem is NP-Hard.*

Proof: By reduction from Maximum Coverage. Refer to Appendix C of [14]. ∎

Algorithm 4: Heuristic value of topic t given partial solution $\mathcal{S}'$

```
1  GetHeuristicFB3M(t, ev, Int, S', τ)
   Input: t, ev, Int, S', τ
   Data: h ← 0 : Heuristic value
   rem_v : Events remaining to make user v happy
2  foreach {v ∈ V_t} do
3      rem_v ← τ_v − Σ_{t'∈S'∩T_v} ev_{t'}
4      if rem_v > 0 then
5          h ← h + min(rem_v, ev_t) / τ_v
6  return h
```

Algorithm 5: Appropriate simple greedy algorithm for *F-B3M*, given a type

```
1  GreedyFB3M(T, V, ev, cost, Int, τ, C, type)
   Input: T, V, ev, cost, Int, τ, C, type
   Data: A : Array of size l
   Result: S' ← ∅ : Output set of topics
2  foreach t ∈ T do
3      A[t] ← ComputeHeuristic(t, ev, cost(t), Int, S', τ, type)
4  while T ≠ ∅ do
5      t ← argmax_{x∈T} A[x]
6      T ← T \ {t}
7      if cost(t) + Σ_{t'∈S'} cost(t') ≤ C then
8          S' ← S' ∪ {t}
9          repeat
10             t' ← t
11             t ← argmax_{x∈T} A[x]
12             A[t] ← ComputeHeuristic(t, ev, cost(t), Int, S', τ, type)
13         until A[t'] = A[t]
14 return S'
```

Algorithm 6: Appropriate heuristic, given a *type*

```
1  ComputeHeuristic(t, ev, cost(t), Int, S', τ, type)
   Input: t, ev, cost(t), Int, S', τ, type
2  if type = G then
3      return GetHeuristicFB3M(t, ev, Int, S', τ)
4  else if type = R then
5      return GetHeuristicFB3M(t, ev, Int, S', τ)/cost(t)
```

Algorithm 7: Greedy algorithm for ***F-B3M***

```
1  ModifiedGreedyFB3M(T, V, ev, cost, Int, τ, C)
   Input: T, V, ev, cost, Int, τ, C
2  S' ← GreedyFB3M(T, V, ev, cost, Int, τ, C, G)
3  S'' ← GreedyFB3M(T, V, ev, cost, Int, τ, C, R)
4  if σ(S') ≥ σ(S'') then return S'
5  else return S''
```

A. Greedy Heuristic

Theorem V.2. *The objective function in the* F-B3M *problem from Expression* (3) *is a* ***submodular*** *function.*

Proof: Refer to Appendix D of [14], where we also define submodularity. ∎

From Theorem V.2 we infer that the *F-B3M* problem is essentially the budgeted maximization of a submodular function. The generalized greedy heuristic for maximization of submodular functions is known to guarantee a con-

stant approximation factor as shown in [16]. Unfortunately, greedily selecting topics with best benefit-cost ratio for a budgeted maximization of a submodular function no longer gives a constant approximation guarantee. Greedily choosing the topics similarly to the solution for *B3M* performs arbitrarily poorly.

To see why the simple greedy approach fails, consider an instance with two topics t_1 and t_2 with $\sigma(t_1) = 1$ and $cost(t_1) = 1$ and $\sigma(t_2) = x$ for some $x > 1$ and $cost(t_2) = x+1$ and with $\mathcal{C} = x+1$. The heuristic of benefit-cost ratio prefers t_1 over t_2. Having spent a budget of 1 the heuristic can no longer select t_2 and terminates with $\sigma(t_1) = 1$ while the optimal solution is choosing t_2 with the gain $\sigma(t_2) = x$ giving an approximation ratio of x.

Taking inspiration from [17], we address this problem by running two instances of a greedy algorithm, each using a different heuristic. The first algorithm, which we refer to as being of *type* $\mathcal{G}$, uses σ as shown in Algorithm 4. The second algorithm, of *type* $\mathcal{R}$, uses the benefit-cost ratio $(\sigma/cost(t))$. The final solution is the best of the two solutions provided by executing the algorithms of type $\mathcal{G}$ and $\mathcal{R}$, respectively. The pseudocode of the simple greedy algorithm is shown in Algorithm 5. Algorithm 7 is the pseudocode for the modified greedy algorithm to solve the *F-B3M* problem that executes the simple greedy algorithms of *type* $\mathcal{G}$ and $\mathcal{R}$ and selects the best solution.

Our simple greedy algorithm (Algorithm 5) includes an optimization that is important in practice, but does not affect the worst-case run time. After selecting a topic, the contribution of other topics needs to be updated. Here we observe that, due to submodularity, the contribution of those topics can only decrease. Thus, we loop over the sorted list of topics in descending order of value and stop updating as soon as the contribution of the topic with maximum contribution (top topic in max-heap) does not change. This is done between lines 9 and 12.

Theorem V.3. *Algorithm 7 has an approximation ratio of* $\frac{1}{2}\left(1 - \frac{1}{e}\right)$.

Proof: A general result for budgeted maximization of submodular functions was given by Krause and Guestrin [18][Theorem 1]. Our Algorithm 7 is a minor extension of theirs, the difference being that they only select a single element when $type = \mathcal{G}$. ∎

We remark that, following [18], one can also create a greedy heuristic with an approximation ratio of $1 - \frac{1}{e}$ at the cost of an additional factor of $|T|^3$ in the running time of the algorithm.

Theorem V.4. *Given an instance* $\text{B3M}(T, V, ev, cost, Int, \tau, \mathcal{C})$ *where costs are normalized, for any solution* $\mathcal{S}$ *it holds that:*

$$\sigma(\mathcal{S}) \leq \max\left(|V'| : \sum_{v \in V'} \max\left(\tau_v, \min_{t \in T_v} ev_t\right) \leq \mathcal{C}\right) + 1\,,$$

where $V' \subseteq V$.

Note that Theorem V.4 is an extension of Theorem IV.7 with a minor difference in that there may be a fractional contribution to the objective function. This fractional part is upper bounded by 1.

Theorem V.5. *Algorithm 7 has run time complexity of* $\mathrm{O}\big(|T|^2(|V| + \log|T|)\big)$.

Proof: Refer to Appendix A of [14]. ∎

VI. Evaluations

A. Experimental Setup

We implemented both GreedyB3M and Modified-GreedyFB3M using C++. To evaluate these heuristics we make use of real data from *Spotify*'s deployed pub/sub system. The data consists of about 1.1 million topics and 4.9 million subscribers. The traces were gathered for 10 days from *Spotify*'s data center at Stockholm. For more information about the data traces refer to [1]. We use the normalized cost function: for each topic $cost(t) = ev_t \cdot |V_t|$. To choose $\mathcal{C}$ we analyzed the full data traces and computed the total capacity needed to handle the full traces in terms of the total cost of all the topics $\sum_{t \in T} cost(t)$. Unless mentioned explicitly, for evaluations in this paper we set the capacity constraint $\mathcal{C}$ to be 10% of this sum. For τ we used 1%(27) to 100%(2763) of the mean event rate of all the topics. All experiments were executed single threaded on a server with 16 cores of Intel Xeon 2.13GHz processors and 32 GB of RAM.

B. Performance of GreedyB3M

First we analyze the performance of GreedyB3M (Algorithm 2) comparing it to the upper bound computed by GetUpperBound (Algorithm 3). To visualize the performance we observe that both algorithms iteratively construct solutions. Thus, in Figure 2 we show the progress of the GreedyB3M algorithm after selecting a topic in each iteration, by comparing the service capacity used so far (x-axis) against the number of satisfied subscribers (for a given τ) (y-axis) by the chosen topics. Note that this represents a single run of GreedyB3M until a budget $\mathcal{C}$ of 10% of the workload is reached. However, the intermediate results are equivalent to having stopped GreedyB3M at the corresponding values of $\mathcal{C}$. We can see that the gap between GreedyB3M and the upper bound increases as $\mathcal{C}$ also increases in most cases when $\mathcal{C}$ is restricted to 10%.

An interesting observation is that, with $\mathcal{C}$ equivalent to 10% of what is needed to handle the full workload, the gap between GreedyB3M and the upper bound increases as τ increases from 27 to 276 (the approximation ratio drops from 0.87 to 0.75, as shown in Figure 4). However, this changes when τ is increased to 2763, in which case the approximation ratio of GreedyB3M increases from 0.75 to 0.82. $\tau \approx 27$ is a reasonably realistic value. With this parameter we satisfy around 72% of all subscribers (3.5 million of the total 4.9 million). The upper bound gives that at most 86% (4.2 million) of subscribers can be satisfied, with an approximation ratio of around 0.83.

C. Performance of ModifiedGreedyFB3M

We now analyze the performance of Modified-GreedyFB3M. From the theoretic results, we know that

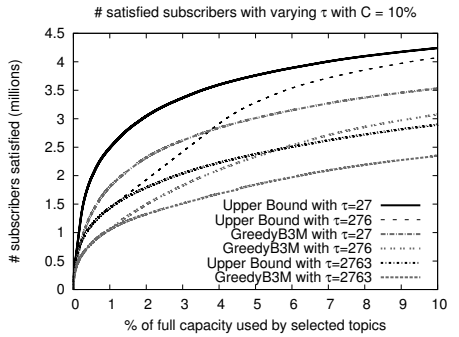

Fig. 2. Comparison of GreedyB3M with the Estimated Upper Bound

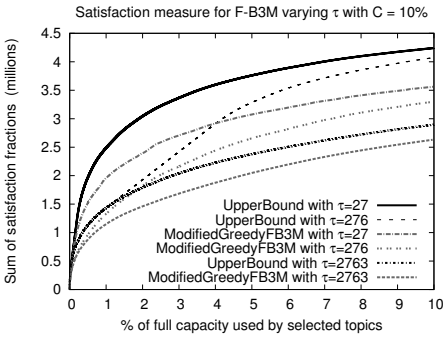

Fig. 3. Comparison of ModifiedGreedyFB3M with the Estimated Upper Bound

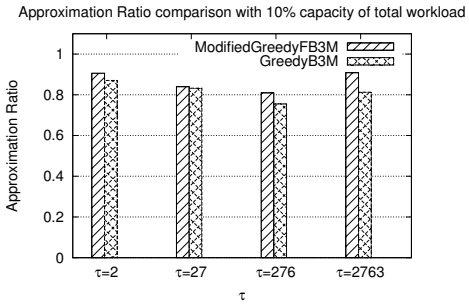

Fig. 4. Computed approximation ratios for *B3M* and *F-B3M* with varying τ

ModifiedGreedyFB3M guarantees an approximation ratio of $\frac{1}{2}\left(1-\frac{1}{e}\right)$. In our real-world data set, we achieve a significantly better ratio (up to 0.9). Analogously to our analysis of GreedyB3M, we use the upper bound given by Theorem V.4. This theorem can be easily turned into an algorithm identical to Algorithm 3 but with the change that in the last step (line 9) we return $|csubs|+1$ instead. Since the goal of *F-B3M* is to maximize the total satisfaction fraction among the subscribers of all the topics, the outcome is measured in terms of total fraction instead of number of subscribers. As shown in Figure 3, a similar pattern to GreedyB3M is observed in the approximation ratio when the τ changes from 27 to 2763. However, the gap between the ModifiedGreedyFB3M and the upper bound is much lower compared to the gap between GreedyB3M and its corresponding upper bound. For example for $\tau = 2763$ the approximation ratio between ModifiedGreedyFB3M and the upper bound is 0.9 compared to 0.82 for GreedyB3M, as shown in Figure 4.

GreedyB3M and ModifiedGreedyFB3M algorithms are intended to run on a regular basis, thus it is important that they are fast. In Figure 5 the running times of the greedy approaches proposed in this paper are shown in seconds (mean of 3 runs). We also introduce a naive version coined ModifiedGreedyFB3MSlow, to evaluate the gain of exploting submodularity structure to lazily updating costs in ModifiedGreedyFB3M as explained in Section V-A . ModifiedGreedyFB3MSlow is identical to ModifiedGreedyFB3M except from line 9 to 12 of Algorithm 5. Instead of lazily updating topic costs, all the topics that have a common subscriber with the chosen topic in the current iteration are updated (same as lines 6 and 7 of Algorithm 2). From Figure 5 it is clear that ModifiedGreedyFB3M outperforms ModifiedGreedyFB3MSlow and runs in less than 20 seconds for all values of τ, while without optimization it takes a maximum of 33 seconds to run for $\tau = 276$. It is clear that these algorithms are in general fast to run in large-scale settings and can be run on a regular basis.

D. Real-Time Performance

The solutions for *B3M* and *F-B3M* are expected to be run periodically to recompute the solution. In these periodic computations, the input sizes are smaller as they only need to provide a solution until the next computation, meaning that topics without publications can be ignored. To evaluate their performance in this scenario, we use the stream of publications from *Spotify* with a fixed $\tau = 20$ and $\mathcal{C}$ varying from 1% to 50%. We divide the stream in smaller time windows, where each window is an hour long. We then execute our algorithms for the topics active in 10 consecutive time windows. In Fig. 6 we show the execution time of the GreedyB3M and ModifiedGreedyFB3M algorithms. The algorithms execute in just a few hundred milliseconds, and ModifiedGreedyFB3M executes at least twice as fast as GreedyB3M due to the proposed optimization. The running times reflect the size of the workload and, for a typical workload in *Spotify*, the solutions are suitable for periodic execution in real-time. In Fig. 7 we show that both heuristics provide similar approximation ratios. However, ModifiedGreedyFB3M performs slightly better in all cases. An interesting observation is that, as $\mathcal{C}$ increases, the approximation ratios also increase.

VII. Related Work

There are many types of pub/sub systems proposed in the literature [19]. Proposals from the last 15 years come from both industry [1]–[3] and academia [6], [7], [19]. Publisher placement and subscriber relocation to minimize metrics like publication-notification delay and system load in content-based pub/sub systems have been proposed before [20]. In [10] a peer-assisted pub/sub service to offload workload from the cloud is proposed. We believe that cloud resource utilization under this approach can be improved by defining and then maximizing satisfaction metrics using our proposed algorithms. To the best of our knowledge, we are the first to formalize subscriber satisfaction metrics and formulate the problem of maximizing the number of satisfied subscribers under resource constraints.

The formal definition we arrive at bears a strong resemblance to (set) coverage problems; the problem of Budgeted Maximum Coverage (BMC) [17] being the closest match. However, a significant difference is that in our setting a subscriber may need to be "covered" more than once. The family of coverage problems are generally proven NP-Hard using reductions from the Max-Cover problem [15]. We instead reduce *DkS* to our *B3M* problem, which allows us to rule out the existence of a PTAS.

Seminal work on analysis of the maximization of submodular set functions was originally done in [16]. We ex-

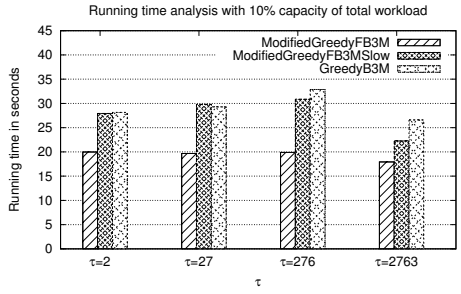

Fig. 5. Running time comparison for Greedy Heuristics with varying τ

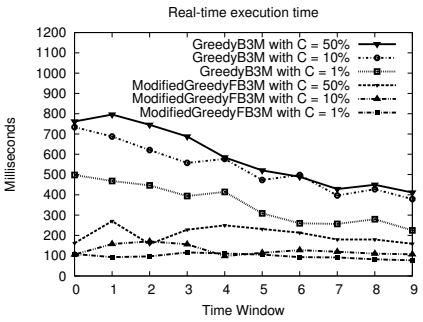

Fig. 6. Real-time execution time of heuristics

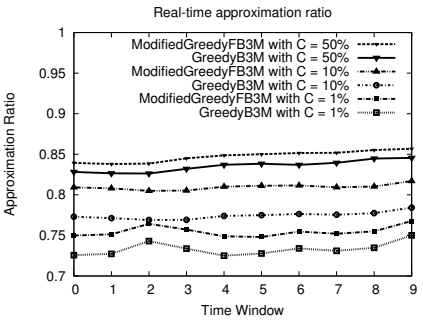

Fig. 7. Approx. ratio measured in real time

ploit the submodularity property of the objective function of *F-B3M* to derive a constant approximation ratio for its greedy heuristic and to speed up the corresponding algorithm.

VIII. Conclusions

In this paper, motivated by practical scenarios in a real deployed pub/sub system at *Spotify*, we proposed a new approach to maximize subscriber satisfaction. In the process, we introduced a new set of problems (*B3M* and *F-B3M*) to address the maximization of the number of satisfied subscribers in a pub/sub system and proposed greedy heuristics to solve both problems. We proved that *B3M* is NP-Hard by reduction from the *DkS* problem and, as a corollary, also proved that *B3M* has no PTAS under a standard assumption. *F-B3M* is a relaxed version of *B3M* that is relatively easy to solve. We proved that the objective function of *F-B3M* is submodular, derived a constant approximation bound for its greedy heuristic, and proposed a way to exploit the submodularity of the objective function to improve the running time of the heuristic for typical scenarios. We evaluated our heuristics for both problems using a large-scale real data set from *Spotify*'s pub/sub system and compared their performance with upper bounds we derived for the optimal solutions of both problems. We illustrated that, with a realistic pub/sub workload as input, our heuristics achieve an approximation ratio of at least 0.7 and they can be run in under a second in a realistic scenario to adapt to the workload variations. We conclude that we have demonstrated that there is theoretical and practical evidence that pub/sub systems (like *Spotify*'s pub/sub) can benefit from the algorithms presented in this paper.

Acknowledgments

The authors would like to thank Per Austrin for suggesting the reduction from *DkS* to *UC-B3M*.

References

[1] V. Setty, G. Kreitz, R. Vitenberg, M. van Steen, G. Urdaneta, and S. Gimåker, "The hidden pub/sub of spotify: (industry article)," in *Proc. ACM DEBS*, 2013, pp. 231–240.

[2] J. Reumann, "GooPS: Pub/Sub at Google," Oslo, Norway, Lecture & Personal Communications at EuroSys & CANOE Summer School, 2009.

[3] "Tibco rendezvous," http://www.tibco.com.

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

[5] M. Petrovic, H. Liu, and H.-A. Jacobsen, "G-ToPSS: Fast filtering of graph-based metadata," in *Proc. ACM WWW*, 2005, pp. 539–547.

[6] G. Li, V. Muthusamy, and H.-A. Jacobsen, "A distributed service-oriented architecture for business process execution," *ACM Trans. Web (TWEB)*, vol. 4, no. 1, pp. 2:1–2:33, Jan. 2010.

[7] P. Triantafillou and I. Aekaterinidis, "Peer-to-peer publish-subscribe systems," in *Encyclopedia of Database Systems*, L. Liu and M. Özsu, Eds. Springer US, 2009, pp. 2069–2075.

[8] G. Kreitz and F. Niemela, "Spotify – large scale, low latency, P2P music-on-demand streaming," in *Proc. IEEE P2P*, 2010, pp. 1–10.

[9] U. Feige, M. Seltser *et al.*, "On the densest k-subgraph problem," *The Weizmann Institute, Rehovot, Tech. Rep*, 1997.

[10] T. Xu, Y. Chen, L. Jiao, B. Y. Zhao, P. Hui, and X. Fu, "Scaling microblogging services with divergent traffic demands," in *Proc. ACM/IFIP/USENIX Middleware*, 2011, pp. 20–40.

[11] U. Feige, D. Peleg, and G. Kortsarz, "The dense k-subgraph problem," *Algorithmica*, vol. 29, no. 3, pp. 410–421, 2001.

[12] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan, "Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph," in *Proc. STOC*, 2010, pp. 201–210.

[13] S. Khot, "Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique," *SIAM Journal on Computing*, vol. 36, no. 4, pp. 1025–1071, 2006.

[14] V. Setty, G. Kreitz, G. Urdaneta, R. Vitenberg, and M. van Steen, "Maximizing the number of satisfied subscribers in pub/sub systems under capacity constraints," Insitute for Informatics, University of Oslo, Tech. Rep. 429, July 2013. [Online]. Available: http://tidal-news.org/techreports/Setty-TR-429-07-2013.pdf

[15] D. S. Hochbaum, "Approximation algorithms for NP-hard problems," D. S. Hochbaum, Ed. PWS Publishing Co., 1997.

[16] M. Fisher, G. Nemhauser, and L. Wolsey, "An analysis of approximations for maximizing submodular set functions-I," in *Polyhedral Combinatorics*. Springer Berlin Heidelberg, 1978.

[17] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Information Processing Letters*, vol. 70, no. 1, pp. 39–45, 1999.

[18] A. Krause and C. Guestrin, "A note on the budgeted maximization of submodular functions," Carnegie Mellon University, Tech. Rep. CMU-CALD-05-103, 2011.

[19] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv. (CSUR)*, vol. 35, no. 2, pp. 114–131, Jun. 2003.

[20] A. K. Y. Cheung and H.-A. Jacobsen, "Publisher placement algorithms in content-based publish/subscribe," in *Proc. IEEE ICDCS*, 2010, pp. 653–664.