Minimal Broker Overlay Design for Content-Based Publish/Subscribe Systems

Naweed Tajuddin, Balasubramaneyam Maniymaran and Hans-Arno Jacobsen
University of Toronto, Toronto, Ontario, Canada

Abstract

Mission-critical distributed applications, such as Internet advertising platforms, increasingly utilize distributed publish/subscribe systems as a messaging substrate for information dissemination. These applications require low latency performance from the substrate, as the timely delivery of messages can have a direct impact on revenue. The cost of managing and operating distributed publish/subscribe systems, however, can be prohibitive due to system size and scale. It is, therefore, critical to derive low latency message delivery from a minimal set of system resources.

To this end, this paper presents a solution for designing low latency, minimal-broker overlay networks for content-based publish/subscribe systems. The solution is developed in two parts. First, a framework is developed to quantify the similarity of entities in content-based publish/subscribe systems. Second, algorithms are presented for designing overlays that utilize a minimal number of brokers in order to provide low latency performance at reduced cost.

1 Introduction

Distributed publish/subscribe systems are increasingly the technology of choice for information dissemination in large-scale distributed applications [18, 17, 9]. A distributed publish/subscribe system consists of publishers that publish information, subscribers that consume published information, and a set of brokers connected in an overlay topology, that route publications from publishers to interested subscribers. When used as a messaging substrate for mission-critical applications, such as Internet advertising platforms, the timely delivery of messages by the underlying publish/subscribe system is of the utmost importance, as delivery delays can directly impact revenue [17, 20]. In practice, many publish/subscribe system designs over-provision resources to prevent processing delays from causing network bottlenecks, a technique that increases resource administration and utilization costs, but may not reduce delays if the broker overlay is poorly configured.

In this paper, we present techniques for designing a broker overlay that provides low latency message delivery at minimal cost. Delivery latencies are reduced in two ways: First, by placing clients (publishers and subscribers) with similar interests at the same broker, where possible; and second, by constructing an overlay topology that establishes direct overlay links between brokers with similar interests. System cost is reduced by utilizing only as many brokers as necessary so as to avoid latencies associated with broker congestion. We refer to this problem as the *Minimal Broker Overlay Design Problem* (MBODP). MBODP consists of multiple sub-problems, including the estimation of a minimal number of brokers that avoids processing delays caused by broker congestion; the quantification of the similarity among publishers, subscribers, and brokers; the efficient allocation of clients to the minimal broker set; and the construction of an overlay topology that provides low end-to-end message delivery latencies.

Previous works on overlay design [1, 11, 6, 14] have focused primarily on the overlay topology construction sub-problem for both topic-based and content-based publish/subscribe systems. In a topic-based publish/subscribe system, information is processed according to the group, or *topic*, to which the message belongs, whereas in the more

expressive content-based systems, information is processed according to message content. The solutions presented in the literature have assumed a fixed set of brokers, and therefore, no explicit consideration was given to the effects of processing delays caused by broker congestion, as well as the critical goal of reducing system cost. This paper, however, presents a comprehensive and unified solution for solving all sub-problems of the MBODP problem for content-based publish/subscribe systems. It includes a load modeling framework for determining a minimal number of brokers, a similarity framework for achieving low latency message deliveries, as well as client-broker allocation and overlay construction algorithms that produce low latency overlay designs.

In the content-based publish/subscribe space, much of the related work has focused on distributed, or run-time solutions, where brokers autonomically reorganize overlay connections while the system is operational [1, 11, 14]. This is opposed to a centralized, design-time solution, that assumes all pertinent information is available at a central location and is computed and implemented by a central service. A key benefit of the design-time approach is that it may avoid much of the messaging overhead and convergence delays associated with a run-time solution (see Section 2.2). In addition, the design-time methodology is the design principle of choice in practice [17, 7]. We, therefore, target to develop a design-time solution for content-based publish/subscribe systems, given that publish/subscribe implementations in industry tend to be managed using design-time schemes. Two such systems are *Google Publish/Subscribe* [17] and *Yahoo Message Broker* [7], both of which provide critical messaging support for large-scale external and internal applications. However, both systems are topic-based. Given the recent prevalence of applications of content-based systems, such as [12, 19, 13], there is a need for a design-time solution in the content-based space. More precisely, we target a quasi-design-time solution, where a new overlay design can be periodically re-computed based on more current information becoming available over time, and applied to a running system. In order to consider the possibility of converting an existing run-time solution into a design-time solution, we convert the popular run-time overlay construction algorithm in [1] into a design-time algorithm and use it for comparison in our experimentation. Our evaluation shows that our original design-time algorithm provides lower latency performance than the converted run-time algorithm.

In presenting a complete solution to MBODP we make the following contributions:

1. We present a general framework for quantifying the similarity of interests between publishers, subscribers, and brokers. The framework is used throughout the algorithms presented in this paper.

2. We present an analytical model for measuring load on content-based publish/subscribe systems. This load model predicts the load impact of publishers and subscribers on brokers, and is used for designing broker overlays.

3. We prove that the problem of determining a minimum number of brokers to satisfy a given workload without exceeding broker capacity is an NP-hard problem. We subsequently present a heuristic client-broker allocation algorithm that simultaneously determines a minimal number of brokers and allocates clients to the broker set.

4. We present a design-time overlay construction algorithm that establishes direct overlay links between brokers based on the degree of aggregated similarity of connected clients.

5. We provide an extensive simulation study to evaluate the performance of the publish/subscribe overlays created using different algorithms.

2 Related Work

2.1 Topic-based Publish/Subscribe

The overlay design problem for topic-based publish/subscribe systems has been primarily investigated for large-scale peer-to-peer (P2P) systems [6, 16, 4], with the various contributions exploiting the fact that a given pair of topics are either identical or different. Thus, the overlay design algorithms favour overlay links that connect nodes with a large number of common topic subscriptions. Initially, a dedicated overlay was maintained for each

topic, with publications forwarded to the appropriate overlay and flooded to all members. A limiting factor of dedicated topic overlays is the excessive number of links needing to be managed by nodes interested in many topics.

To reduce the number of links, Chockler et al. [6] introduced the notion of a *topic-connected* overlay, which requires the sub-graph consisting of nodes with at least one common topic to be connected. A design-time heuristic algorithm, called GM, is presented to construct topic-connected overlays by iteratively connecting the pair of nodes that maximally reduces the number of topics that are not topic-connected. The total number of overlay links is reduced since nodes with multiple topics in common are connected through a single overlay link. An additional work by Chen et al. uses divide and conquer techniques to connect multiple topic-connected overlays [4].

Onus et al. [16] found that GM may produce overlays with high maximum node degree (relative to the average node degree of the network), which may create bottlenecks in the system. They present a design-time heuristic algorithm to construct an overlay with minimal maximum node degree.

Although limiting maximum node degree may reduce the incidence of network bottlenecks, nodes with low degree may experience message traffic in excess of their processing capacities. Accordingly, the contributions of this paper culminate in a design-time heuristic algorithm for content-based overlays, where overlay structure is determined with explicit consideration for expected traffic demand at each node.

2.2 Content-based Publish/Subscribe

For content-based publish/subscribe systems, overlay design has primarily been investigated for managed systems that implement a tree-based overlay network, as is typical of most publish/subscribe research implementations [3, 8]. In addition, overlay design is considered as a run-time problem, where brokers execute distributed protocols to autonomically restructure the overlay network [1, 11, 14, 2]. The various run-time techniques execute some or all of the following steps to reconfigure the overlay.

First, an evaluation phase is executed where a given broker considers new overlay connections. If a link is found that may improve system performance, a consensus phase is initiated where affected brokers verify that the proposed link will not decrease overall system performance. Since these phases are distributed, messages are exchanged among affected brokers to measure similarity and determine consensus. If a new link is established, to maintain a tree-based overlay, a link removal phase is executed where an existing link is selected for removal. From the cycle that results with the addition of the proposed link, the link that contributes the least similarity is the one that is removed. Lastly, a reconfiguration phase is executed where traffic is halted as routing table updates are propagated among the affected brokers.

A key difference among the various run-time approaches is the similarity metric used in the evaluation and consensus phases to determine if a new link should be established. In [1], the author's define a "zone of interest" for a broker as the union of all its local subscriptions. For a given broker pair, the overlapping region of the zone of interests is divided by the zone of interest of the broker evaluating the new overlay connection. Whereas this metric is asymmetric and therefore evaluates new connections from the perspective of a single broker, our commonality metric is a symmetric metric, as we aim to quantify the equivalence of a given pair of subscriptions, which in other words corresponds to the likelihood that a publication will match both subscriptions or neither.

The similarity metrics used in [11, 14, 2] measure similarity in terms of the number of recent publications that matched at a given pair of brokers. Thus, two brokers have high similarity if, of the recent N publications matched by one broker, many of them were also matched by the other broker. The similarity metrics in [11, 14] are additionally augmented with abstract node and link cost functions, intended to model the cost of processing a message on a given node and the cost of transmitting a message across a link.

Yoon et al. [21] and Di Nitto et al. [15] examine practical issues associated with run-time reconfiguration. They define primitives for dynamic reconfiguration of overlay networks which provide message delivery guarantees during the reconfiguration process.

The approaches discussed thus far for content-based systems are run-time approaches and have

assumed a fixed set of brokers. Our contribution, however, is a design-time overlay design solution. Run-time solutions may not be appropriate for certain applications that require low latency performance due to the time required for the distributed protocols to execute, as well as for multiple algorithm iterations necessary for the overlay to converge to a near-optimal solution. Moreover, a design-time approach may avoid much of the message overhead required for evaluation, consensus, and rewiring by run-time algorithms. In addition, current practice in industry are centrally managed publish/subscribe systems [17, 7]. To consider the possibility of converting an existing run-time algorithm into a design-time algorithm, in our evaluation, we implement a centralized version of the distributed algorithm presented in [1]. The results show that our original design-time algorithm presented in this paper offers improved performance than the converted run-time algorithm.

As an alternative to reducing overlay distance through rewiring the overlay, Cheung and Jacobsen present algorithms for relocating publishers closer to interested subscribers [5]. However, they do not consider clustering similar subscribers, which provides a significant performance improvement as shown in Section 6.3.

3 Problem Formulation

The goal of the *Minimal Broker Overlay Design Problem* (MBODP) is to design a broker overlay for a content-based publish/subscribe system that deploys a minimal number of brokers while producing a low average message delivery latency. Here, we assume that the system cost is measured by the number of brokers deployed. An overlay with more-than-an-optimal number of brokers incurs unnecessary initialization and maintanence costs, and additionally may result in higher delivery latencies due to longer routing paths. On the other hand, an overlay network with too few brokers may increase latencies due to processing delays caused by broker congestion.

The MBODP problem is formally defined as follows:

Inputs: The input to MBODP describes the available resources and expected workload. This includes the set of brokers available for use, $\mathcal{B} = \{b_i\}$, their associated processing capacities, $\mathcal{P}_i$ (measured in message rate), the set of publishers, $\mathbf{P} = \{p_j\}$, and subscribers, $\mathbf{S} = \{s_k\}$, the set of advertisements, $\mathbb{A} = \{\mathfrak{a}_x\}$, and subscriptions, $\mathbb{S} = \{\mathfrak{s}_y\}$, and the publication rate r_x for each advertisement $\mathfrak{a}_x$. Also given are the partitions of advertisements produced by each publisher p_j, $\mathbb{A}_j$, and of subscriptions submitted by each subscriber s_k, $\mathbb{S}_k$.

Outputs: A solution to MBODP produces the set of brokers chosen for deployment, $\mathbf{B} = \{b_i\}$, where $\mathbf{B} \subseteq \mathcal{B}$; the *client-broker allocation* describing the set of publishers and subscribers connected to each broker b_i, $\mathbf{P}_i$ and $\mathbf{S}_i$, respectively; and the broker overlay topology, T, defined as an adjacency matrix of size $|\mathbf{B}| \times |\mathbf{B}|$.

Objectives: MBODP has a dual objectives: First, to produce an overlay design that minimizes average message delivery latencies, and second, to minimize the number of brokers utilized.

Constraints: The processing load ω_i on each broker b_i should not exceed its processing capacity $\mathcal{P}_i$, i.e. $\omega_i \leq \mathcal{P}_i$. The processing capacity specifies the number of messages a broker can process in a unit of time.

4 Similarity and Load Modelling Framework

Our work extensively uses the concept of "similarity" between publish/subscribe components, which is a measure of how similar two subscriptions or an advertisement and a subscription are. This similarity model is used not only for clustering similar interested components to reduce the length of the path a message takes, but also for estimating load experienced by the brokers in the system. This section first introduces the *content space* model we used to quantify similarity and then provides the metrics for this quantification. It also presents the load model used in our work.

4.1 Content Space

This paper considers publish/subscribe systems that employ an advertisement-based routing protocol, where publishers advertise the information schema they intend to publish in form of advertisement messages. *Advertisements* are flooded in the system to all the brokers. Incoming subscrip-

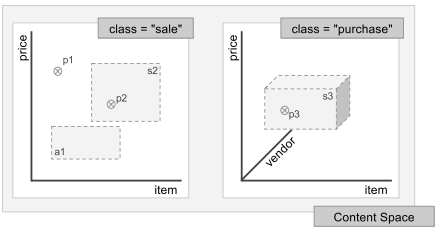


Figure 1: A content space with two attribute spaces.

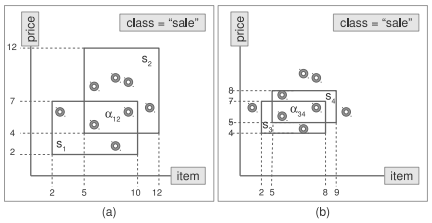


Figure 2: Calculating similarity using overlap

tions are then matched against advertisements and forwarded along the reverse path of the matching advertisements. Publications are then forwarded along the reverse path of the matching subscriptions.

We assume that advertisements, subscriptions, and publications can be mapped into a *content space*. The content space consists of multiple *attribute spaces*, each belonging to a type, or *class*, attribute. An attribute space is a multi-dimensional space, with each axis belonging to a particular attribute. A publication, produced as a set of attribute–value pairs, becomes a point in a particular attribute space of a content space. Advertisements and subscriptions are modeled as conjunctions of *range-based* predicates and, therefore, become hyper-rectangles in the content space. If the hyper-rectangles of an advertisement and a subscription overlap, there is a possibility that a publication (associated with said advertisement) will match the subscription. A publication matches a subscription if it is a point within the hyper-rectangle of the subscription. An example content space is shown in Figure 1.

4.2 Measuring Similarity

The term *commonality* describes the similarity between subscriptions. The commonality between two subscriptions denotes the likelihood that a publication that matches one subscription will also match the other subscription. When calculated for subscription sets maintained by brokers, the commonality between two brokers describes the likelihood of a publication matching a subscription at one broker to also match a subscription at the other broker.

Commonality-based similarity can be quantified by measuring the volume of intersection between a pair of subscriptions. For example, Figure 2(a) shows two subscriptions, s_1 and s_2, in a two-dimensional content space. The intersecting, or *overlapping*, region α_{12} is a region of the content space where a publication will match both subscriptions. Intuitively, the larger the intersecting region, the greater the similarity of the subscriptions. However, the absolute value of the area of overlap can be misleading. For example, Figure 2(a) shows an overlap with a larger absolute value than the one in Figure 2(b) (15 vs. 10). However, given an identical set of publications, only 40% that match the subscription s_1 will match subscription s_2, while 67% that match s_3 will match s_4. Therefore, the relative size of the overlap area is better suited for measuring similarity than the absolute value. We define the value of overlap between two subscriptions $\mathfrak{s}_x$ and $\mathfrak{s}_y$ as:

$$o_{xy} = \frac{\alpha_{xy}^2}{\mathcal{A}_x \mathcal{A}_y}, \tag{1}$$

where $\mathcal{A}_x$ and $\mathcal{A}_y$ are the volumes of subscriptions $\mathfrak{s}_x$ and $\mathfrak{s}_y$, respectively. The value of overlap ranges from 0 (two disjoint subscriptions) to 1 (identical subscriptions).

In order to calculate the commonality between two brokers, the entire subscription sets maintained by the brokers have to be considered. It is calculated by summing o_{xy} calculated for all subscription pairs from the subscription sets in a given pair of brokers. Formally, the overlap-based commonality c_{ij} between two brokers i and j is defined as:

$$c_{ij} = \sum_{\mathfrak{s}_x \in |\mathbb{S}_i|} \sum_{\mathfrak{s}_y \in |\mathbb{S}_j|} \frac{\alpha_{xy}^2}{\mathcal{A}_x \mathcal{A}_y}, \tag{2}$$

where $\mathbb{S}_i$ and $\mathbb{S}_j$ are the subscription sets of brokers i and j, respectively. Note that the value of c_{ij} is the same irrespective of which broker measures it.

Advertisements, when used in conjunction with subscriptions, offer an additional tool for quantifying similarity. We use the term *interest* to denote the similarity between an advertisement and a subscription, which is estimated similarly from the advertisement–subscription overlap. When a publication is originated, the similarity between the relevant advertisement and a given subscription provides an estimate for the probability of the publication matching the subscription. Note that, unlike commonality, interest is an asymmetric, directional metric, because publication flow is always from advertiser to subscriber. Therefore, the interest, $\overrightarrow{v_{xy}}$, between an advertisement $\mathfrak{a}_x$ and subscription $\mathfrak{s}_y$ is calculated as follows:

$$\overrightarrow{v_{xy}} = \frac{\alpha_{xy}}{\mathcal{A}_x}, \tag{3}$$

where α_{xy} is the area of the overlap between $\mathfrak{a}_x$ and $\mathfrak{s}_y$, and $\mathcal{A}_x$ is the volume of $\mathfrak{a}_x$.

The interest between two brokers i and j, $\overrightarrow{q_{ij}}$, can be estimated by considering the advertisement set $\mathbb{A}_i$ in broker i and the subscription set $\mathbb{S}_j$ in broker j. This describes the likelihood of a publication emanating from broker i matching a subscription at broker j. It is calculated as follows:

$$\overrightarrow{q_{ij}} = \sum_{\mathfrak{a}_x \in |\mathbb{A}_i|} \sum_{\mathfrak{s}_y \in |\mathbb{S}_j|} \frac{\alpha_{xy}}{\mathcal{A}_x}, \tag{4}$$

4.3 Load and Capacity Model

We introduce a load model in this section to estimate (1) the load that a client will impose on a broker and (2) the set of clients that a broker can support without exceeding its capacity. The capacity of a broker is given in terms of *message throughput*, the number of messages the broker is capable of processing in a unit of time.

The term *load impact* is used to describe the load a client will impose on the broker to which it is connected. It is defined as the number of messages the broker must process in a unit of time due to the client. The load impact of a publisher is equal to its publication rate, because a broker must process all of the messages the publisher produces regardless of whether they are forwarded or dropped.

On the other hand, the load impact of the subscriber must be determined by estimating the message rate with which the subscriber is going to receive messages (which must be processed by the broker to which it is connected). Consider a subscriber s_k with a subscription $\mathfrak{s}_x$ and a publisher p_j with advertisement $\mathfrak{a}_y$; let the publisher produce publications matching the advertisement $\mathfrak{a}_y$ with the rate of r_y. From Equation 3, we can estimate that s_k will receive messages from p_j with the rate of $r_y v_{yx}$. Because a subscriber can send out many subscriptions, the total receiving message rate, or load impact, of the subscriber s_k can be estimated as:

$$l_{s_k} = \sum_{\mathfrak{s}_x \in \mathbb{S}_{s_k}} \sum_{\mathfrak{a}_y \in \mathbb{A}} r_y v_{yx},$$

where $\mathbb{A}$ is the set of advertisements in the system and $\mathbb{S}_{s_k}$ is the set of subscriptions sent out by subscriber s_k. Figure 3 shows an example where the subscriber s contains only one subscription and receives publications from three publishers p_1, p_2, and p_3 each with one advertisement. The load impact of s on broker b_1 can be calculated as 9 msg/sec.

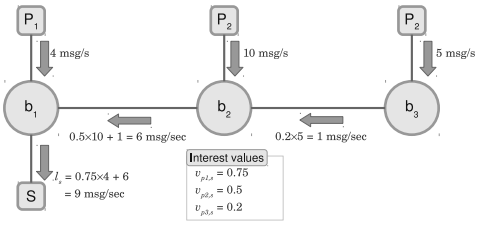


Figure 3: Measuring the load impact of subscribers

The load on a broker can be calculated by summing all the load impacts of the clients attached to it. However, if a local subscriber receives publications from a local publisher, the load impact caused by these publications are included twice in calculating the broker load: once in the load impact of the publisher and again in the load impact of the subscriber. Therefore, the load impacts of subscribers are corrected accordingly when calculating broker loads. The estimated load on a broker i, L_i, is calculated as follows:

$$L_i = \sum_{p_j \in \mathbf{P}_i} r_{p_j} + \sum_{\mathfrak{s}_x \in \mathbb{S}'_i} \sum_{\mathfrak{a}_y \in (\mathbb{A} \setminus \mathbb{A}'_i)} r_y v_{yx} \tag{5}$$

where $\mathbf{P}_i$ is the set of locally connected publishers of broker i; $\mathbb{S}'_i$ and $\mathbb{A}'_i$ are the sets of subscriptions and publications in broker i sent from the local subscribers and publishers, respectively.

5 Broker Allocation and Overlay Design

Our algorithm to solve MBODP is composed of two phases as depicted in Figure 4. The first phase, *client-broker allocation*, determines a minimal number of brokers needed to satisfy a given workload and simultaneously assigns clients to them. The set of brokers is deemed to be *minimal*, if all the selected brokers operate to their full capacity. The second phase, *overlay topology construction*, creates a tree-based overlay connecting all the chosen brokers.

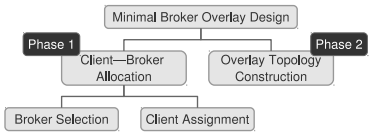


Figure 4: Phases of solving minimal broker overlay design problem

While determining a minimal set of brokers is handled by the first phase, both phases are designed to improve end-to-end message performance through the clustering of clients (in the first phase) and brokers (in the second phase) that have a high degree of similarity. The clustering of clients enables a larger number of messages to be contained within a broker, thereby avoiding routing delays. The clustering of brokers improves performance through a reduction in the number of pure forwarding brokers.

5.1 Phase 1: Client–Broker Allocation

The general flow of the client–broker allocation algorithm is to sequentially assign a client to an existing broker in the overlay that has not reached its capacity. When no existing brokers are able to handle the load imposed by the next unallocated client, a new broker is deployed. Existing brokers in the overlay are referred to as *deployed brokers*. The clients are allocated in a particular order, determined by a *client ranking function* (CRF). Four ranking functions are described shortly.

The pseudo code for the client-broker allocation algorithm is presented in Algorithm 1. The inputs to the algorithm are the set of available brokers, the set of clients (publishers, **P**, and subscribers, **S**), and a client ranking function. The output will be the set of deployed brokers and a client–broker assignments.

Algorithm 1 Client-Broker Allocation

```
1:  Input Brokers, Clients, CRF
2:  DeployedBrokers ← ∅
3:  DeployedClients ← ∅
4:  I ← computeInterestMatrix(Clients)
5:  J ← computeCommonalityMatrix(Subscribers)
6:  BrokerStack ← sort(Brokers)
7:  ClientStack ← sort(Clients, I, CRF)
8:  while ClientStack ≠ ∅ do
9:      c_x ← ClientStack.pop()
10:     for all c_i ∈ c_x do
11:         if c_i ∉ DeployedClients then
12:             b_x ← findBrokerForClient(c_i, I, J, DeployedBrokers)
13:             if b_x == null then
14:                 B ← BrokerStack.pop()
15:                 DeployedBrokers.add(B)
16:                 b_x ← B
17:             connectClientToBroker(c_i, b_x, ClientStack, CRF)
18:             DeployedClients.add(c_i)
```

Lines 2–7 describe the algorithm initialization process. The algorithm starts with an overlay with no brokers and no client allocations. The algorithm next computes the $|\mathbf{P}| \times |\mathbf{S}|$ interest matrix, **I**, and $|\mathbf{S}| \times |\mathbf{S}|$ commonality matrix, **J**. These matrices contain the pair-wise interest and commonality measurements among the set of clients, respectively. Next, in Line 6, the set of brokers are sorted in descending order of capacity, which determines the order in which brokers are deployed. They are sorted in descending order, which enables a greater number of clients to be allocated to a smaller set of brokers, thus reducing the overall number of brokers in the overlay and enabling stronger clustering per broker. Finally, the clients are sorted according to the specified CRF. The client rankings determine the order in which clients are allocated, and thus affect the set of clients allocated to a particular broker. The four ranking functions are described below:

Greatest Load Impact (GLI): This CRF sorts clients in descending order of load impact (Section 4.3), independent of whether the client is a publisher or subscriber. This sorting is motivated by a greedy heuristic for solving the knapsack problem [10], which aims to maximize the overall value of the items in a knapsack. Similarly, the technique assigns the clients with the greatest load impact first, under the assumption that clients with large load impacts have strong similarity, thus indirectly maximizing similarity at the most powerful brokers. In our evaluation study, we found that

places a batch of high load impact subscribers at the top of the list, followed by an interspersing of publishers and subscribers, followed by a batch of low load impact subscribers.

Greatest Interest (GI): This CRF does not sort individual clients, but sorts publisher–subscriber pairs in descending order of the interest value between them. It therefore places the publishers and subscribers that have the greatest interest, and therefore are projected to be heavy communication partners, at the top of the list. Note that a client can appear multiple times in the sorted list, but is assigned only once (the first time they appear). Since this function clusters subscribers with their most interested publishers, it favours low latency communication. During the client allocation process, the publisher is allocated first, followed by the client.

Greatest Interest with Group Allocation (GIg): This CRF sorts groups comprised of a single publisher and all the subscribers who has non-zero interest with it. The highest interest values within the groups are used to sort the groups in descending order. By clustering a publisher with a group of subscribers, this CRF incorporates a measure of commonality into the rankings, thus encouraging a client allocation where a single publication matches multiple subscriptions per hop. Per group, the allocation process assigns the publisher first, followed by the subscribers in descending order of interest with the publisher.

Baseline: This function sorts clients in a random order, with no regard for load impact or similarity. This technique models typical publish/subscribe deployments where clients are allocated in an ad-hoc manner. Note that when this scheme is used, the list of available brokers is also sorted randomly.

Once the initialization is complete, the algorithm begins assigning clients from the sorted client list to a chosen broker. The algorithm exits when all clients have been assigned. Each iteration of the algorithm (inside the **while** loop in Algorithm 1 at Lines 8–18) does the following: The client or client group is popped from the sorted client stack and a broker from the deployed brokers is chosen per client using the `findBrokerForClient()` function. If this function returns null, a new broker is popped from the broker stack (Lines 14–15). Once a broker has been selected, its available capacity is adjusted accordingly. When all the clients are deployed, the chosen broker set and client assignment is fed into the overlay construction algorithm (see Section 5.3.)

Algorithm 2 *findBrokerForClient()*

```
1: Input c_i, I, J, DeployedBrokers
2: totalSimilarity ← 0
3: SimilarityStack ← ∅
4: for all B ∈ DeployedBrokers do
5:    S ← B.subscribers
6:    for all s_x ∈ S do
7:       if c_i is publisher then
8:          totalSimilarity += I(c_i, s_x)
9:       else
10:         totalSimilarity += J(c_i, s_x)
11:   SimilarityStack.add(B, c_i)
12:   totalSimilarity ← 0
13: SimilarityStack.sort()
14: while SimilarityStack ≠ ∅ do
15:   b_x ← SimilarityStack.pop()
16:   newEstimatedLoad = b_x.currentLoad() + c_i.loadImpact(b_x)
17:   if (newEstimatedLoad < LCF × b_x.capacity()) then
18:      return b_x
19: return null
```

The `findBrokerForClient` function is described in Algorithm 2. It determines the best broker for the client assignment based on two criteria: First, the broker must have available capacity to accommodate the new client (see Section 4.3), and second, the client set already assigned to the broker must have the highest similarity value compared to the client sets of other deployed brokers. If the new client is a publisher, the similarity is equal to the sum of the interest values between it and all the subscribers at a broker (Line 8); if it is a subscriber, the commonality values between it and the subscribers are considered (Line 10). Once the similarity per broker has been determined, the set of deployed brokers are ranked according to greatest similarity (Line 13). The first broker in the sorted list that can handle the additional load imposed by the new client without exceeding its capacity is returned as the candidate (Line 14–18; the factor LCF in Line 17 is explained in Section 5.4.) If no such broker is found, the function returns *null*. The client–broker allocation algorithm therefore assigns each client to an existing broker with which it has the strongest similarity and deploys a new broker only when the capacity of the existing overlay becomes inadequate.

5.2 NP-Completeness of Client–Broker Allocation Problem

In this section, we prove that the client–broker problem (Phase 1 of MBODP) is NP-complete, thereby validating the need for a heuristic solu-

tion, as presented in the previous section. Because the complexity of the overlay construction problem (Phase 2 of MBODP) has already been proven NP-complete [11], we can conclude that MBODP is NP-complete.

The decision version of the client–broker allocation problem is stated as follows:

Definition 1. Client–broker allocation decision problem: Given n clients (publishers and subscribers) that send or receive messages at a rate of $r_1, \ldots, r_n$ messages per unit of time and B brokers capable of processing $\mathcal{P}$ messages per unit of time, is it possible to allocate all clients to the set of B brokers without exceeding the processing capacity of any broker?

Theorem 1. *The client–broker allocation problem is NP-hard.*

Proof. First, without loss of generality, consider a content-based publish/subscribe system where half of the clients are publishers and the other half are subscribers. Let each publisher and subscriber offer one advertisement and subscription, respectively. Additionally, let each advertisement occupy a distinct area of the content space, such that no two advertisements overlap. Moreover, let there be a publisher–subscriber pairing, such that the advertisement of the publisher and subscription of the subscriber occupy same regions of the content space. Thus, each subscriber will receive all messages from its paired publisher and only those messages. Here, the rate at which messages are sent and received is equal to the publication rate of the publisher.

We now show that the client–broker allocation decision problem is NP-hard by reducing an instance of the classical bin-packing problem to an instance of the client–broker allocation problem. The bin-packing problem states that n items with sizes $\beta_1, \beta_2, \ldots, \beta_n \in (0,1]$ must be packed into unit-sized bins so as to minimize the number of bins used. This problem has been proven to be NP-complete [10].

Let the n items of the bin-packing problem correspond to the publisher–subscriber pairings. Let the sizes of the items correspond to the publication rates of the publishers ($\beta_i = r_i$.) Let a bin correspond to a broker and the capacity of the bin to the broker's processing capacity. Thus, the bin packing problem has been reduced to allocating publisher-subscriber pairings to a minimal set of brokers. Therefore, the client-broker allocation problem is NP-hard. □

5.3 Phase 2: Overlay Topology Construction

In order to construct an overlay topology, the broker network is modeled as a graph with brokers as vertices and communication links as edges. The overlay construction then follows a two-step process: First, similarity values are calculated for every pair of brokers, and second, a maximum spanning tree algorithm is executed using the similarity values as edge weights to determine the overlay configuration that maximizes overall similarity. This process creates direct overlay links between brokers that share the most similar subscription sets, thus reducing the length of the path a publication has to traverse to reach all the interested subscribers. In other words, it reduces average message latency.

Algorithm 3 Commonality Overlay Construction

```
1: Input DeployedBrokers
2: C ← ∅
3: for all b_x ∈ DeployedBrokers do
4:     for all b_y ∈ DeployedBrokers do
5:         if (b_x == b_y) then
6:             continue
7:         C(b_x, b_y) ← computeCommonality(b_x.subs, b_y.subs)
8: T = computeMaximumSpanningTree(C)
```

The pseudo code of the overlay construction algorithm is presented in Algorithm 3. The input to the algorithm is the list of deployed brokers in the overlay. The algorithm first computes the commonality values (edge weights) for every broker pair using the set of subscriptions in each broker and Equation 2 (Lines 3–7.) Then a maximum spanning tree algorithm is applied to the weights to determine the overlay topology (Line 8.) We used Prim's algorithm to construct the maximum spanning tree.

5.4 Load Compensation Factor

Note that when determining the available capacity of a broker in the client–broker allocation algorithm, only the load impacts of the locally connected clients are considered. However, broker load can also be affected by other clients. For example, in Figure 3, the load on the broker b_2 is contributed not only by p_2, but also by s, because b_2

has to forward messages from b_3 to s. This additional load imposed by the forwarding traffic is decided by the topology of the overlay. For example, if the topology in Figure 3 is such that b_3 is connected to b_1, then b_2 would not be subjected to any forwarding load.

When the load imposed by the forwarding traffic is taken into account, it can invalidate some of the assignments made in the client–broker allocation phase. This implies that phases 1 and 2 of our algorithm (Figure 4) have to be executed iteratively until convergence to a stable solution is reached. However, this type of iterative algorithm can be very costly, because each of the two phases themselves are compute-intensive. Therefore, in order to simplify the algorithm, a safety margin is introduced at each broker, called *load compensation factor* (LCF). This factor reserves a fixed portion of the broker capacity for handling forwarding traffic, where only the remaining capacity can be used in the client–broker allocation phase. The LCF is a system parameter, defined in the range of 0 to 1. The LCF allows the two phases of the algorithm to be disjoint, thereby permitting non-iterative execution of the algorithm. As we will see in our evaluation (see Section 6.4), the LCF technique is sufficient to build a low-latency overlay through a single execution of the algorithm.

5.5 Algorithm Assumptions

The models we have outlined thus far make several assumptions. First, we assume that publications are uniformly distributed across the content space. This has an effect on the interest metric, as well as the load impact calculation. We note, however, that the algorithm presented here is independent of the specific formula for interest calculation. The algorithm is also independent of a specific load impact computation method, as long as a load measurement function is provided to update broker load.

Second, we assume all brokers are identical. This implies that all brokers have a standard processing delay (excluding congestion-effects), and that the cost of deployment of all brokers is identical. In practice, however, the cost can vary dependent on a number of factors such as the capacity and monetary costs associated with broker utilization. In that case, there should be a specialized cost function that incorporates all those parameters. Although creating such a cost function is left for future work, the core of the algorithms presented here will remain the same. The broker ranking function may need modification, however.

Third, we assume unlimited network capacity. It is left for future work to extend our models and algorithms to account for link bandwidth constraints.

6 Evaluation

To evaluate our overlay design techniques, overlays are constructed using each of the four client ranking functions, and compared against one another, as well as a design-time version of the run-time algorithm presented in [1]. The overlay designs are examined in terms of cost (number of brokers deployed) and performance (message count and message latencies). In addition, the effectiveness of the load modeling framework at reducing congestion effects is evaluated. The section ends with an analysis of the effect of the load compensation factor on performance and load.

6.1 Evaluation Overview

The algorithms presented in this paper were implemented in Java and evaluated via simulations using the JiST discrete event simulator[1]. First, the client–broker allocation algorithm was run (Section 5.1) on a generated workload to assign clients to a minimal set of brokers. Next, the overlay construction algorithm (Section 5.3) was executed to produce the broker overlay. The produced overlay was programmed into the simulator and the results were collected as the simulations were executed. The values presented in each graph are averages over five independent runs.

A *time unit* was considered to be 1000 ticks of simulation time[2]. Thus, a publisher with a rate of five messages per time unit would publish a message every 200 ticks. The message processing model was implemented to produce the effects of congestion: When a broker is overloaded, the messages processed by the broker are subjected to an additional delay that is proportional to the overload value. We assumed overlay link latencies are fixed at 100 ticks.

[1] http://jist.ece.cornell.edu/

[2] An individual tick models a sub-unit of time, such as a millisecond.

Table 1: Workload Summary

System Parameter	Value
Number of Advertisements	200 - 1000
Number of Subscriptions	600 - 3000
Publication Duration (time units)	10
Broker Capacity (messages per time unit)	1000
Max Publication Rate	10
Distribution of Publication Rate	Zipf
Number of Attributes	2
Attribute Value Range	0 - 1000
LCF	0.75

The workload used in the experiments is summarized in Table 1. It follows the general advertisement-based publish/subscribe model where the number of publications is much greater than the number of subscriptions that is much greater than the number of advertisements. In our workload, the number of subscriptions created was three times the number advertisements. Each advertisement was assigned with a publication rate of up to 10 messages per time unit, based on a Zipf distribution. Publications were uniformly distributed across the content space. Publications were produced for the first ten time units of the simulation and the simulation was allowed to continue until all the publications have reached their destinations.

The followings are the performance metrics used to evaluate each overlay design:

- *Average message count*: The average number of messages generated per publication. A lower value indicates better client and broker clustering.

- *Average message latency*: The average latency (in simulation ticks) required for a message to be delivered.

- *Pure forwarding message ratio*: The ratio of the number of messages that merely passed through a broker (not matched by a local subscription) to the total number of messages processed by it. The lower the value, the better the performance, because the pure forwarding traffic imposes unnecessary load on brokers and increases message latency. The values presented are averages over all the brokers.

- *Average peak load*: Peak load is the maximum number of messages processed by a broker in any 1000-tick interval. Average peak load is the average over all brokers.

- *Average processing delay*: The average delay experienced by a message due to broker congestions.

Due to a lack of related work matching our design space, we were forced to create a design-time version of the distributed algorithm presented in [1] for comparison. This converted algorithm, referred to as DT-Baldoni is implemented as follows. First, the set of deployed brokers are connected in a random acyclic overlay. Then, each broker is compared with every other non-neighbor broker to identify a new link that can provide a greater similarity value. If such a link is found, the topology is restructured by establishing this new link and removing the link with the least similarity value in the resulting cycle. The algorithm loops through the brokers until an iteration produces no additional rewirings or a maximum number of iterations is reached.

6.2 Overlay Design Cost

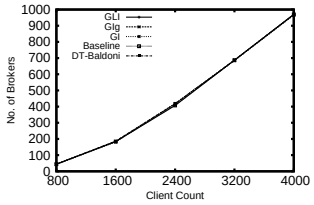
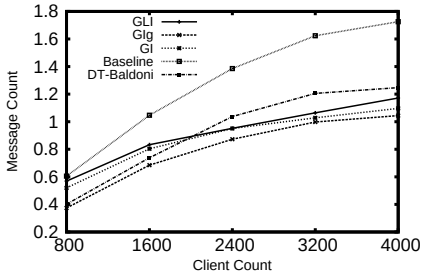


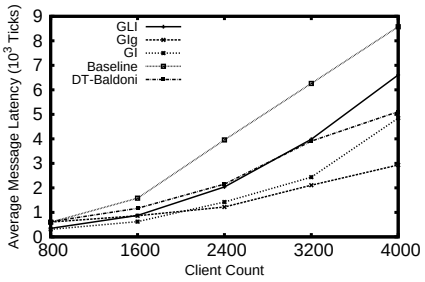
Figure 5: Overlay cost measured in terms of broker count

Figure 5 shows the number of brokers deployed by the algorithm for each client ranking function (CRF) as the number of clients is increased from 800 (200 publishers, 600 subscribers) to 4000 (1000 publishers, 3000 subscribers.) The figure shows that all ranking functions utilize approximately the same number of brokers. This is a consequence of the algorithm scanning all the deployed brokers for available capacity prior to deploying a new one. Accordingly, a client is allocated to a broker even if there is low similarity with local clients, so long as the broker can support the additional load. Thus, while the ranking function affects the client clustering per broker, and therefore has an effect on performance and broker load, the overall load impact of the entire client set remains the same and all clients can be accommodated by a similar number of brokers.

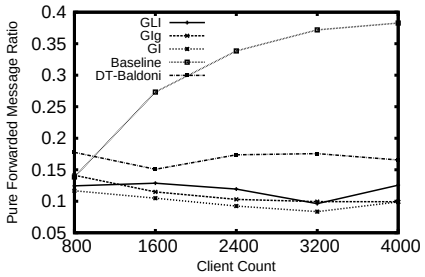
6.3 Overlay Design Performance



(a) Message count



(b) Message latency



(c) Pure forwarding traffic

Figure 6: Effect of clustering on overlay performance

Figures 6(a) and 6(b) show the average message count and average message latency in the overlays constructed using different CRF and with the DT-Baldoni algorithm. The metrics are evaluated as the number of clients is increased from 800 to 4000. Through the plots we can evaluate the impact of the frequency with which publishers are allocated, as follows.

The GLI technique performs better than the baseline approach, but worse than the GI and GIg techniques. Since GLI allocates clients with high load impact first, all of which are subscribers, the GLI technique creates an overlay where the majority of subscribers are clustered to the initial set of brokers. When it is turn for publishers to be allocated, there is less capacity at the deployed brokers for the clustering publishers and subscribers that share interests. This results in an increased overlay distance between publishers and subscribers, and therefore larger message counts and latencies.

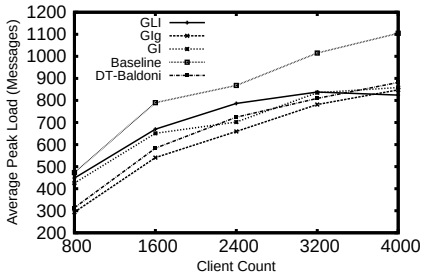
The GI technique performs second best. As it allocates publishers and subscribers in turn, the overlay distance is reduced for many interested publishers and subscribers. However, the frequency with which publishers are allocated favours interest at the expense of commonality. In other words, as publishers are allocated frequently, there is less capacity at the brokers for additional subscribers to connect. This hinders a clustering where a single publication matches many subscriptions at the same broker.

The GIg technique provides the lowest message counts and latencies, and therefore produces the most effective client clustering. Since it allocates a single publisher followed by all interested subscribers, the overlay contains a subset of brokers that connect very many subscribers. This is in fact similar to the GLI technique. However, since a publisher is allocated along with the group of subscribers, the initial set of brokers contains publishers in addition to many subscribers. Thus, there is greater occurrence of a single publication matching multiple subscriptions at the same hop, which GLI is not able to produce since it allocates publishers much later in the allocation process.

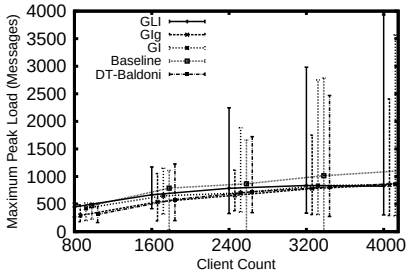
Figure 6(c) shows the fraction of all messages that are pure forwarding messages. Since the baseline technique does not explicitly cluster similar clients, it produces a large number of pure forwarding messages. The ranking routines, through the clustering of clients at brokers, are able to limit the number of pure forwarding messages to 10–15% of all messages processed. Moreover, the percentage of pure forwarding traffic remains constant as the workload increases, despite the increased number of brokers.

The DT-Baldoni algorithm performs better than the baseline algorithm, but worse than the GI and GIg versions of our overlay design algorithm. This is likely because the iterative DT-Baldoni algorithm greedily selects new links for a broker considering only itself, which might lead to an inferior overlay overall.

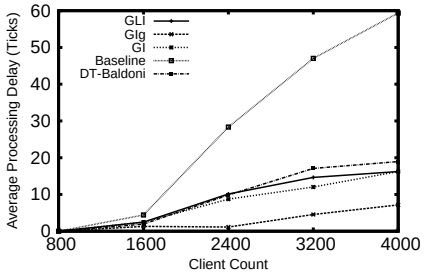
6.4 Load Modeling and Broker Congestion Effects



(a) Average peak load



(b) Maximum peak load



(c) Processing delays due to broker congestion

Figure 7: Effectiveness of load modeling framework at reducing broker congestion

The plots in Figures 7(a) – 7(c) show the effectiveness of our load modeling framework. Figure 7(a) shows the variation of average peak load with an increasing number of clients. Recall that the peak load measures the maximum load experienced by a broker. Whereas the baseline overlay produces an average peak load above the broker capacity (1000 messages per time unit) for a large workload, the overlays produced with our algorithms are able to contain the load, and thereby reduce processing delays. In addition, as the workload increases, the peak load values level-off, indicating that the solution is scalable.

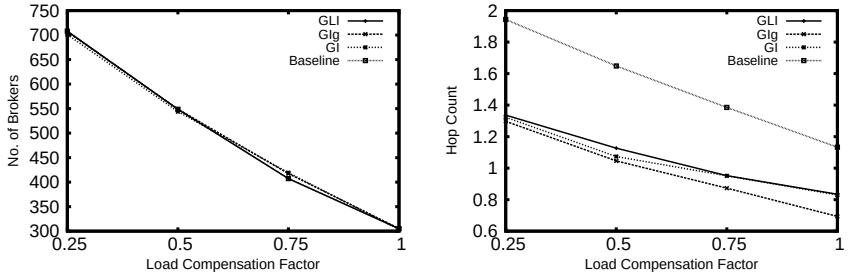
Figure 7(b) shows the maximum peak load experienced in the overlay (by a single broker). We see that the maximum peak load for the GLI ranking function is significantly larger than that of the other techniques. Since GLI initially assigns the clients with the largest load impacts to the first set of brokers, these brokers are a destination for a disproportionately large number of messages, and therefore experience large loads. The GIg overlay has the lowest maximum peak load, and therefore produces the overlay with both the lowest load and the most even load distribution.

Figure 7(c) shows the average processing delay that messages experience due to broker congestion. Since the graph shows that none of the overlays produce zero processing delays, all overlays do experience some congestion. This observation is also supported by Figure 7(b) where all maximum peak loads are above the broker capacity of 1000 messages per time unit. The congestion is due to pure forwarding traffic, which is not explicitly accounted for by our load modeling framework. However, by comparing our algorithms to the baseline approach in Figure 7(c), it is evident that the clustering aspects of our algorithm, along with the load modeling framework, are sufficient to significantly reduce congestion effects.

The performance of DT-Baldoni in terms of broker load is similar to its performance in terms of message latency: While it performs better than the baseline it is still inferior to our GIg algorithm.

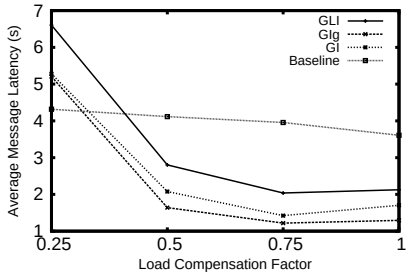
6.5 Load Compensation Factor

In this section, we examine the effect of the load compensation factor in terms of overlay design performance and broker congestion reduction. The results presented are for a workload consisting of 2400 clients, and otherwise as described in Table 1. Figure 8(a) shows that the LCF value has a significant impact on the number of brokers deployed. Since a lower LCF reduces the amount of capacity reserved for client allocation, more brokers are required. Figure 8(b) shows that the message count decreases with increasing LCF: when fewer brokers are deployed, (a) more clients are allocated per broker, thereby increasing the probability of a publication match in fewer hops; and (b) the network di-

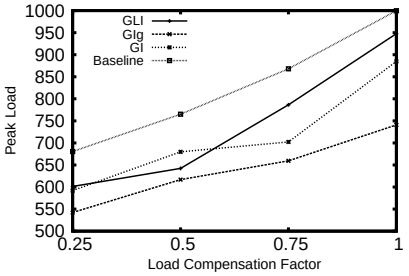


(a) Effect of LCF on overlay cost

(b) Effect of LCF on message count



(c) Effect of LCF on message latency



(d) Effect of LCF on broker load

Figure 8: Impact of LCF value on performance and load

ameter is reduced, thus reducing the average message path.

Figure 8(c) shows the effect of the LCF on average message latency. As with message count, the latency is affected by the number of brokers in the overlay: When fewer brokers are deployed, the average path length of the overlay is reduced, and messages reach their destinations with less delay. The slight increase in message latency as LCF is increased LCF beyond 0.75 is the effect of broker congestion (as shown in Figure 8(d), a higher LCF value results in an increases in peak load, thereby causing broker congestion delays).

Figures 8(a) and 8(c) show the tradeoff between overlay cost and average message latencies. The cost to acheive a marginal improvement in message latency (by reducing the LCF from 1.0 to 0.75) requires the deployment of an additional one-hundred brokers.

7 Conclusion

In this paper, we defined and presented a comprehensive solution for the *Minimal Broker Overlay Design Problem* (MBODP) for content-based publish/subscribe systems, where a broker overlay with a minimal number of brokers is to be found, that also provides low average message delivery latency. Our solution decomposes the MBODP into two sub-problems: (a) Client–broker allocation, where the number of brokers and client–broker assignments are determined, and (b) overlay construction, where the minimal set of brokers are placed in an overlay that provides low message delivery latency. To the best of our knowledge, this is the first work to look at these sub-problems as a whole; wherein our approach of devising algorithms to solve the sub-problems whilst exploring their complexity and interdependency consequently produces an effective solution to the whole.

Our algorithms are supported by a system model extended from previous works for measuring similarities between publish/subscribe components. In addition to considering subscriptions in content-

based publish/subscribe systems, our model also exploits the presence of advertisements, which provides new options in estimating content-based traffic. We have also introduced a model for estimating load at brokers due to content-based traffic. Again, we believe this to be among the first load estimation works found in the literature for content-based publish/subscribe systems.

References

[1] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Subscription-driven self-organization in content-based publish/subscribe. tehnical report, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 2004.

[2] R. Baldoni, R. Beraldi, L. Querzoni, A. Virgillito, and R. Italia. Efficient publish/subscribe through a self-organizing broker overlay and its application to siena. *The Computer Journal*, 2007.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 2001.

[4] C. Chen, H.-A. Jacobsen, and R. Vitenberg. Divide and conquer algorithms for publish/subscribe overlay design. In *ICDCS '10*, 2010.

[5] A. K. Y. Cheung and H.-A. Jacobsen. Publisher placement algorithms in content-based publish/subscribe. In *ICDCS '10*, 2010.

[6] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *PODC '07*, 2007.

[7] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 2008.

[8] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski. Distributed publish/subscribe for workflow management. In *ICFI'05*, 2005.

[9] K. Heires. Budgeting for latency: If I shave a microsecond, will I see a 10x profit?, 2010.

[10] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems.* PWS Publishing Co., 1997.

[11] M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann. Self-organizing broker topologies for publish/subscribe systems. In *SAC '07*, 2007.

[12] I. Koenig. Event processing as a core capability of your content distribution fabric. In *Gartner Event Processing Summit*, 2007.

[13] G. Li, V. Muthusamy, and H.-A. Jacobsen. A distributed service-oriented architecture for business process execution. *ACM Trans. Web*, 4(1), 2010.

[14] M. Migliavacca and G. Cugola. Adapting publish-subscribe routing to traffic demands. In *DEBS '07*, 2007.

[15] E. D. Nitto, D. J. Dubois, and A. Margara. Reconfiguration primitives for self-adapting overlays in distributed publish-subscribe systems. *SASO '12*, 2012.

[16] M. Onus and A. W. Richa. Minimum maximum-degree publish-subscribe overlay network design. *IEEE/ACM Trans. Netw.*, 2011.

[17] J. Reumann. "Pub/Sub at Google", lecture and personal communication at CANOE summer school, 2009.

[18] C. Schuler, H. Schuldt, and H.-J. Schek. Supporting reliable transactional business processes by publish/subscribe techniques. In *TES '01*, 2001.

[19] L. Vargas, L. I. W. Pesonen, E. Gudes, and J. Bacon. Transactions in content-based publish/subscribe middleware. In *DEPSA*, June 2007.

[20] R. Yerneni. "Internet Advertising", lecture and personal communication at CANOE Summer School, Toronto, Canada, 2010.

[21] Y. Yoon, V. Muthusamy, and H.-A. Jacobsen. Foundations for highly available content-based publish/subscribe overlays. *ICDCS '11*, 2011.