Self-Organizing Broker Topologies for Publish/Subscribe Systems

Michael A. Jaeger* Helge Parzyjegla† Gero Mühl‡
Communication and Operating Systems Group (KBS), Berlin University of Technology
Einsteinufer 17, 10587 Berlin, Germany
{michael.jaeger,parzyjegla,g_muehl}@acm.org

Klaus Herrmann
Institute of Parallel and Distributed Systems (IPVS), Universität Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
klaus.herrmann@acm.org

ABSTRACT

Distributed publish/subscribe systems are usually deployed on top of an overlay network that enables complex routing strategies implemented in the application layer. Up to now, only little effort has been spent on the design of the broker overlay network assuming that it is either static or manually administered. As publish/subscribe systems are increasingly targeted at dynamic environments where client behavior and network characteristics vary over time, static overlay networks lead to suboptimal performance. In this paper, we present a self-organizing broker overlay infrastructure that adapts dynamically to achieve a better efficiency on both, the application and the network layer. This is obtained by taking network metrics as well as notification traffic into account.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Management, Performance

Keywords

Publish/Subscribe, Adaptable Middleware, Self-Organization, Overlay Networks

*Funded by Deutsche Telekom Stiftung.

†Funded by Deutsche Forschungsgemeinschaft.

‡Funded by Deutsche Telekom.

SAC'07, March 11-15, 2007, Seoul, Korea.

1. INTRODUCTION

In the interconnected business world of today, it is of increasing importance to quickly sense and react to relevant events. *Event-driven architectures (EDA)* have been proposed to enable business in real-time by providing decoupled many-to-many messaging. Publish/subscribe (pub/sub) middleware is a basic building block of EDA and has been subject to research for many years now. A distributed pub/sub system consists of a set of *brokers* that are connected to establish an *overlay network*. These brokers cooperate to provide the functionality of the *notification service* to the clients. Clients connect to their particular local broker and can act as producers, consumers, or both. *Producers* publish notifications and *consumers* subscribe to notifications they are interested in. Notifications are usually delivered asynchronously to consumers that are not required to know the publisher of the notification. This way, the communication between publishers and subscribers can be decoupled in space, time (if additional histories are used), and flow [4]. In the following, we write that "a broker publishes (subscribes to) a notification", when we actually mean that a local client of the broker publishes or consumes the notification.

Although pub/sub is targeted at dynamic environments, the structure of the notification service has been assumed to be static or manually managed in many systems so far. However, in scenarios where client usage patterns vary and the network may be subject to change, static topologies cannot keep the system at the optimal working point. On the other hand, manually managing such a system is at least expensive if not impossible. In this paper, we analyze the problem of finding an optimal topology under a given distribution of subscriptions and notifications and prove that this problem is NP-hard. We then propose a distributed on-line heuristic that automatically adapts the structure of the broker overlay network to the communication costs of the links, the processing costs of brokers, and the patterns in the notification flows between brokers.

The paper is structured as follows: in Sect. 2, we discuss related work and point out existing shortcomings. Next, we formalize the basic optimization problem in Sect. 3 and show why using a heuristic is a sound way to solve it. Our algorithms are presented in Sect. 4. In Sect. 5, we evaluate the performance of the proposed solution and compare it to

other heuristics. The results are discussed in Sect. 6. We present our conclusions in Sect. 7.

2. RELATED WORK

The problem of adding adaptivity to the notification service of a distributed pub/sub system to reduce managing costs and increase system performance has been approached in different ways in the past. This mainly concerns the cost metrics considered. Here, we concentrate on acyclic broker topologies as they are commonly used for pub/sub systems.

Maximum Associativity Trees. The approach closest to ours has been published by Baldoni et al. [1, 2]. It explicitly concentrates on tuning the performance of the notification service by exploiting structural similarities of brokers and clustering them accordingly. The authors propose a distributed algorithm that considers the interest of each broker and builds an *associativity* metric from the intersection of these interests. Based on this metric, the algorithm tries to connect brokers with a high mutual associativity value to increase the overall associativity of the system. Thereby, the algorithm aims at decreasing the latency of notifications by reducing the average number of hops they travel.

The authors derive a broker's *zone of interest* from the size of the notification space covered by the broker's local subscriptions [2]. Assuming that notifications are distributed uniformly, larger overlapping zones of interest result in a greater number of identical notifications being consumed by the brokers. However, this implicit assumption limits the applicability of the algorithm. Therefore, the authors introduce a history of local events to calculate the associativity based on the intersection of the messages consumed by two brokers [1]. Thus, the actual notification flows are considered. Unfortunately, the authors do not provide any details on how to efficiently determine the new associativity metric. To avoid extremely degenerated topologies a very limited network awareness is added by simply introducing a manually chosen upper bound for the costs a new overlay link can have. On this level of abstraction, it is not taken into account, that even brokers exhibiting a low associativity value can be successfully deployed to decrease the network traffic and, thus, increase the system performance.

Peer-to-Peer Routing Substrates. For the pub/sub middleware Hermes [10], Pietzuch et al. use Pastry [11] as a peer-to-peer (P2P) routing substrate for the overlay network. This way, they benefit from the scalability, efficiency, and redundancy properties provided by Pastry. Following this approach is certainly a huge improvement to using a manually administrated overlay network. Terpstra et al. [13] follow a similar approach and employ Chord [12] for the management of the overlay network in the Rebeca pub/sub system. Building on a well-understood P2P substrate for the management of the overlay network of a pub/sub system is a sound approach to ease the administration of such systems. However, P2P routing substrates like Pastry and Chord are optimized for query routing and for the location of objects in a P2P network. Neither do they care about the content nor about the flow of notifications in the pub/sub middleware. Thus, the degree of optimization gained by these approaches is limited as the overlay network does not adapt to what is happening on the upper layers.

3. PROBLEM STATEMENT

Our goal is to improve the performance of a pub/sub system by adapting the structure of the broker overlay network. To measure the performance, we consider communication and processing costs which are counted per message as we assume that messages do not vary too much in size. Depending on the optimization goal, the performance metrics have to be chosen sensibly. Communication costs may correspond to the number of hops in the underlying raw network topology, to the message delay, to the bandwidth of links, or to any other desired metric. Of course, they can also be a combination of different metrics. The processing costs may correspond to the average CPU time or memory needed to process (relay or deliver) a message. It is important to note that a reasonable relation between link and processing costs is required to gain the desired performance improvement (e.g., put more weight on the communication costs if they have greater influence on the latency to achieve a low latency network). Besides the costs, we consider the notification flows because the forwarding of notifications determines the actual costs produced by the system. In static environments, where client behavior and network structure do not change over time, it is sufficient to construct the broker overlay network only once in order to optimize the performance of the whole system. In the following section, we consider this static pub/sub overlay optimization problem and show that it is NP-hard, even if the notification flows are known in advance. For large-scale settings, this means that heuristics must be used to find an acceptable solution for this problem. After proving the hardness of the problem, we present two basic heuristics that consider either network or application-level metrics, and give examples for which they lead to suboptimal results. In our evaluation, these heuristics will serve as benchmarks for our algorithm. We conclude, that it is important to consider communication costs, processing costs, and traffic patterns at the same time to improve the overall performance of a pub/sub system. In dynamic environments, the problem is even harder. To keep the system within good working conditions, we additionally have to *adapt* the system to *unpredictable changes*.

3.1 The Static Case

For the formalization of the problem, we start with a connected network graph $G = (\mathcal{V}, \mathcal{E})$. This graph consists of the brokers $\mathcal{V}$ and the potential overlay network links $\mathcal{E}$. Usually, every broker $v \in \mathcal{V}$ can potentially connect to every other broker in $\mathcal{V}$. However, some edges might be omitted, for example, due to administrative reasons. For every link $e \in \mathcal{E}$, we define the *communication cost* c_e. Accordingly, for every broker $v \in \mathcal{V}$, we define the *processing cost* p_v. The communication cost arises if a notification is sent over a link, while the processing cost arises if a notification is processed by a broker. We want to distribute a set of notifications N using a single *spanning tree* T of G where each notification $n \in N$ is published by a certain broker $P(n) \in \mathcal{V}$ and must be delivered to a certain set of brokers $S(n) \subseteq \mathcal{V}$. In the following, we call T the *broker overlay (network)*.

Figure 1 depicts an example of a broker overlay network in a graph consisting of 8 brokers. Instead of using the whole spanning tree for the distribution of n, only the minimal connected subtree of T that contains $P(n)$ and $S(n)$ is used. This subtree is called the *delivery tree* of n. For a given T and n, let $D_T^n = (\mathcal{V}_T^n, \mathcal{E}_T^n)$ be the respective delivery tree.

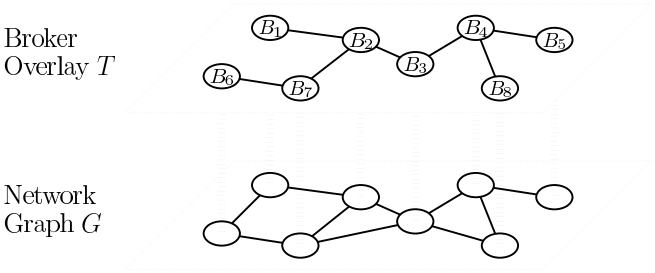

Figure 1: Example graph with an embedded broker overlay.

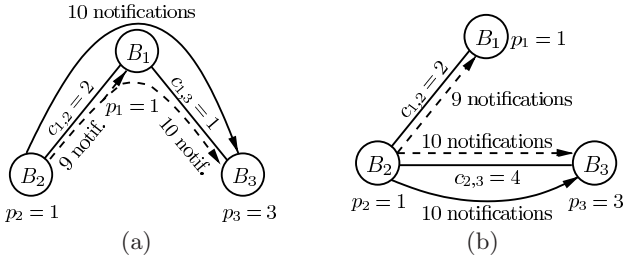

Figure 2: Minimum spanning tree example: B_1 is not interested in any of 10 notifications published by B_2 and consumed by B_3.

Then, the cost for distributing n is given by

$$\text{cost}(n, T) := \sum_{\forall v \in \mathcal{V}_T^n} p_v + \sum_{\forall e \in \mathcal{E}_T^n} c_e \quad (1)$$

since n is sent over all links in $\mathcal{E}_T^n$ and is processed by all brokers in $\mathcal{V}_T^n$. The delivery tree of $n \in N$ is defined by the set of brokers that consists of the publisher and the subscribers of n. We define this set as $V(n) := \{P(n)\} \cup S(n)$. $\mathcal{V}_T^n$ and $V(n)$ can differ significantly. Assume that in the example broker overlay depicted in Fig. 1, broker B_2 publishes a notification n in which only B_5 is interested. In this case, $\mathcal{V}_T^n = \{B_2, B_3, B_4, B_5\}$ includes B_3 and B_4 even though they are not interested in n as $V(n) = \{B_2, B_5\}$.

Obviously, the total cost of forwarding one specific notification in the broker overlay strongly depends on its delivery tree and, thus, on the spanning tree. The overall cost for distributing all notifications in N using one spanning tree T is given by

$$\text{cost}(N, T) := \sum_{n \in N} \text{cost}(n, T). \quad (2)$$

Since we want to use *one* spanning tree for distributing *all* notifications, we choose the spanning tree that minimizes $\text{cost}(N, T)$. Now, we can formally define the static optimization problem:

DEFINITION 1. (Pub/Sub Overlay Optimization Problem (PSOOP)) *Given a graph $G = (\mathcal{V}, \mathcal{E})$, communication costs c_e for all $e \in \mathcal{E}$, processing costs p_v for all $v \in \mathcal{V}$, a set of notifications N, and a function $V : N \to 2^{\mathcal{V}}$, determine the spanning tree T of G that minimizes cost(N, T).*

3.1.1 Complexity

In the following, we show that the decision problem that corresponds to the PSOOP is NP-complete. Thus, we can conclude that PSOOP is NP-hard and it is sensible to apply heuristics to solve the problem.

DEFINITION 2. (Pub/Sub Overlay Decision Problem (PSODP)) *Given a graph $G = (\mathcal{V}, \mathcal{E})$, communication costs c_e for all $e \in \mathcal{E}$, processing costs p_v for all $v \in \mathcal{V}$, a set of notifications N, a function $V : N \to 2^{\mathcal{V}}$, and a bound $C_{\max} \in \mathbb{N}^+$, is there a spanning tree T of G with cost$(N, T) \leq C_{\max}$?*

PSODP is a generalization of the *optimum communication spanning tree* (OCST) problem [5] which deals with the efficient construction of telecommunication networks interconnecting cities. In the OCST problem, each link $e \in \mathcal{E}$ of the graph $G = (\mathcal{V}, \mathcal{E})$ is labeled with communication costs w_e (e.g., with the length of the connecting wire), and the communication requirements $r(\{u, v\})$ between any two vertices $u, v \in \mathcal{V}$ are known. The problem consists of finding a spanning tree T for G with costs that lie below an upper cost bound $C_{\max}$.

The OCST problem has been proven to be NP-complete in 1978 [7]. We can easily show that an instance of the OCST decision problem can be transformed into an instance of the PSODP problem using restriction: we set the processing costs of the nodes to 0 and only consider the communication costs. Thus, PSODP is NP-hard. PSODP is also in NP because a deterministic Turing machine can verify a guessed solution in polynomial time, i.e., compute the actual overall costs of an overlay. Thus, PSODP is NP-complete.

Hence, in order to solve this problem efficiently, we need to apply heuristics. Please note, that heuristics for approximating the OCST optimization problem in an efficient way already exist. However, they all rely on global knowledge.

Before proceeding with the dynamic case, where costs may change and notification traffic is not known in advance, we present two heuristics that rely on simple greedy algorithms and global knowledge. Both have their individual drawbacks and do not consider processing costs. Decentralized versions based on local knowledge will serve as a benchmark in the evaluation section.

3.1.2 Example Overlay Network Heuristics

The main characteristic of *Minimum Spanning Tree (MST)* broker topologies is that they prefer cheap links. It is easy to see that an MST is generally not the best solution since the costs do not only depend on the link costs, but also on processing costs and the notification traffic. It may be sensible, for example, to prefer a more expensive link instead of a cheaper one, if a significant amount of messages on other (even cheaper) links can be saved and, thus, amortize the extra costs for the expensive link. Figure 2 illustrates this: part (a) depicts the minimum spanning tree in which B_2 and B_3 share common traffic (10 notifications, depicted by the solid line), while B_1 is not interested in any notification. The cost for forwarding 10 messages from B_2 to B_3 is given by $10 \cdot (p_2 + c_{2,1} + p_1 + c_{1,3} + p_3) = 100$. By directly connecting B_2 to B_3, as depicted in part (b), the costs are decreased to $10 \cdot (p_2 + c_{2,3} + p_3) = 80$.

The basic idea of the second heuristic is to connect brokers with similar interests (measured by the number of identical notifications consumed or published). To achieve this, the edges are labeled with a value that measures how much interests the brokers share. We call the tree that maxi-

mizes the sum of the edge values *Maximum Interest Tree (MIT)*. The construction of an MIT is similar to an MST. As communication costs are ignored, an MIT may use very expensive links as long as they connect brokers with similar interests. In part (b) of Figure 2, an MIT is depicted. Broker B_2 is directly connected to B_1 and B_3 because B_2 and B_3 share 10 common notifications from which B_2 and B_1 share 9 (depicted by the dashed lines). The costs for the MIT in part (b) is $10 \cdot (p_2 + c_{2,3} + p_3) + 9 \cdot (c_{1,2} + p_1) = 107$. The cost of the tree in part (a) where B_1 forwards notifications coming from B_2 to B_3—and even one notification it is not interested in—is lower: $10 \cdot (p_2 + c_{1,2} + p_1 + c_{1,3} + p_3) = 80$.

3.2 The Dynamic Case

In the previous section, we formalized the problem of finding an optimal broker overlay network in a static setting where all parameters are known (the notifications, their respective publishing/consuming brokers, as well as the processing and communication costs of brokers and links, respectively). However, in many scenarios this setting is not realistic due to a number of reasons. The set of notifications that brokers publish and subscribe to may vary over time. This affects the number of notifications to be transferred and the parts of the overlay network through which they flow. The network topology may also change including processing as well as communication costs.

The above changes can, to a great extent, affect not only the operating costs at any point in time, but also the accumulated overall costs. In these cases, better results, (i.e., lower costs) can be achieved by adapting the overlay network whenever a significant advantage may be gained. Since the changes are in general not known in advance, we are facing a typical on-line problem. Thus, in order to adapt, we have to rely on gathered data about the past for predicting the future. We constantly gather data about notifications published and consumed as well as about processing and communication costs to derive potential adaptations that lower the operating costs.

Since our solution is targeted at large-scale systems, an algorithm based on global knowledge is not feasible. Consequently, the next section presents a heuristic that adapts the overlay network based on local knowledge only.

4. COST AND INTEREST HEURISTIC

In this section, we present our heuristic for dynamically optimizing the broker overlay network of a publish/subscribe system. The basic idea is to reduce the distance between brokers that consume a lot of identical notifications, while respecting communication and processing costs. Therefore, a broker has to *learn* about its local environment.

To estimate the common traffic or *interest* of two brokers (i.e., the notifications both consume or publish), we introduce a cache in every broker. Using this cache, we are able to track the notifications a broker delivers to or receives from its local clients. By comparing their caches, brokers can hence reason about the amount of common interest. Therefore, each broker B periodically exchanges its cache content as well as current processing and communication costs with all brokers in a bounded *neighborhood* $\mathcal{N}_\eta(B)$, which consists of the brokers that are at most η hops away in the overlay network. Based on this information, a broker gathers and maintains up-to-date information about its local environment.

In order to reduce the amount of exchanged data, we use a Bloom filter [3] to represent the contents of a broker's cache (more precisely, the identifiers of cached notifications). A Bloom filter is a space-efficient probabilistic data structure that also supports time-efficient membership queries for stored items. Queries might return false positives, but the probability for this can be reduced by increasing the size of the Bloom filter. Thus, there is an inherent trade-off between space and accuracy. Using Bloom filters, it is not possible to deterministically specify the identities of notifications shared by different brokers. Only their number may be estimated. But this is sufficient for our purposes.

When a broker B_i receives the Bloom filter of another broker B_j, it starts the *evaluation phase*. In this phase, B_i tries to figure out wether it is beneficial for it to connect directly to B_j. This is the case, if B_i and B_j have a significant common interest, such that the total cost of forwarding and processing notifications is decreased after reconfiguration. If B_i comes to the conclusion that it is sensible to connect directly to B_j, it has to coordinate its request for reconfiguration with the other brokers affected by this reconfiguration. This so-called *consensus phase* is important due to the limited knowledge of the individual brokers. It might happen, for example, that B_i decides for a reconfiguration that is beneficial for itself but raises unacceptable costs for the other brokers. In this phase, B_i asks the directly affected brokers about their estimation of the upcoming costs after the reconfiguration and about which link is to be removed in favor of the new link between B_i and B_j to keep the topology acyclic. If the reconfiguration still seems sensible after the consensus phase and a link to remove has been identified, the *reconfiguration phase* starts. In this phase, the actual reconfiguration is executed by exchanging the two links in the broker topology, while avoiding notification losses and maintaining message ordering.

In the following, we explain the evaluation and consensus phase in detail. The reconfiguration phase is not covered, since the deployed algorithms have already been published in [9]. Finally, we show how our heuristic can be integrated into an arbitrary publish/subscribe system.

4.1 Evaluation Phase

Based on the information gathered about their local environments, brokers evaluate alternative overlay connections to nodes in their neighborhood. The brokers use a heuristic to determine, whether it is beneficial to establish a direct link instead of routing notifications indirectly via intermediate brokers. The heuristic builds upon a basic case involving three brokers, that we describe in the following.

The setup of the basic case is composed of the brokers B_i, B_j, and B_k as shown in Fig. 3. Let T_1 be the spanning tree (as indicated by the solid lines), that represents the current network topology, where B_i and B_j are connected indirectly via B_k. Let T_2 be a possible alternative tree (indicated by the dashed lines) containing a link that directly connects them. Furthermore, we assume that B_i receives a Bloom filter representing B_j's cache entries, whereon B_i starts the evaluation phase. We define $I(S)$ as the number of identical notifications all brokers in a set S consume. Using a Bloom filter, B_i can determine $I(\{B_i, B_j\})$ probabilistically. In the following, we denote B_i's approximation as $I_{i,j}$ and use the same notation for analogous estimations.

Given $I_{i,j}$, B_i has to evaluate whether it is sensible to di-

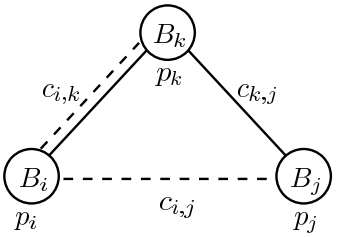

Figure 3: Basic case with three brokers.

rectly connect to B_j. This is the case when the estimated $\text{cost}_{B_i}(T_2)$ of the alternative tree T_2 is lower than $\text{cost}_{B_i}(T_1)$ caused by the current topology. From B_i's point of view, the cost of a tree is calculated as the sum of the communication and processing costs that are caused by routing the notifications it is interested in. For the current topology it is

$$\text{cost}_{B_i}(T_1) = I_{i,k} \cdot (p_i + c_{i,k} + p_k) \ + \ I_{i,j} \cdot (p_i + c_{i,k} + p_k + \ c_{k,j} + p_j) - I_{i,k,j} \cdot (p_i + c_{i,k} + p_k). \quad (3)$$

In Eq. 3, the first term on the right side describes the costs that are caused by the notifications B_i and B_k are both interested in. This includes the communication costs of their link and the processing costs at both brokers. The second term represents the cost of the common traffic between B_i and B_j. As the notifications are routed via B_k, the processing costs at B_k as well as the costs of the links connecting B_i with B_k and B_k with B_j are added. With the last term we subtract the costs for forwarding notifications from B_i to B_j via B_k in which all brokers are interested in (otherwise, they would be counted twice). Similarly, we calculate the cost of the alternative topology in Eq. 4:

$$\text{cost}_{B_i}(T_2) = I_{i,k} \cdot (p_i + c_{i,k} + p_k) + I_{i,j} \cdot (p_i + c_{i,j} + p_j) - I_{i,k,j} \cdot p_i. \quad (4)$$

Since a direct connection is only beneficial if it reduces B_i's costs, we compare the costs caused by the trees of the alternative and current topology (T_2 and T_1, resp.). Given that B_i and B_j share common traffic (i.e., $I_{i,j} > 0$), we come to a decision criteria B_i can use to evaluate the link:

$$c_{i,j} < \frac{I_{i,j} - I_{i,k,j}}{I_{i,j}} \cdot (c_{i,k} + p_k) + c_{k,j}. \quad (5)$$

The link's communication costs are compared to the costs for routing a notification via an intermediate broker reduced by a fraction proportional to the amount of traffic the intermediate broker is also interested in. Hence, the right side of Eq. 5 can also be interpreted as the communication costs of an indirect connection. If B_i and B_j do not share any interest (i.e., $I_{i,j} = 0$), the evaluation phase is aborted.

The number of identical notifications $I_{i,j}$ and $I_{i,k}$ in Eq. 5 are approximated by B_i using Bloom filters. However, $I_{i,k,j}$ is estimated by $I_{i,k,j} = \min(\phi, I_{i,j}, I_{i,k})$, where ϕ is the average number of notifications, in which B_i and any other pair of brokers in $\mathcal{N}_\eta(B_i)$ are interested. ϕ is given by:

$$\phi = \frac{\left(\sum_{B_j \in \mathcal{N}_\eta(B_i)} I_{i,j}\right) - I_i}{|\mathcal{N}_\eta(B_i)| - 1}. \quad (6)$$

As it is impossible that all three brokers are interested in more notifications than any pair of them, the mean ϕ calculated for the whole neighborhood is bounded by $I_{i,j}$ and $I_{i,k}$ using the minimum of the three.

Up to now, we only considered the basic case, which is limited to one intermediate broker. However, in general, other brokers might be more than one hop away in the overlay topology. To evaluate, whether a direct link to such a broker is beneficial, we use the right side of Eq. 5 to define the costs $C_{i,j}$ of an indirect connection recursively, based on the path $B_i, \ldots, B_k, B_j$ as

$$C_{i,j} = \begin{cases} 0 & i = j, \\ c_{i,j} & B_i \in \mathcal{N}_1(B_j), \\ \frac{I_{i,j} - I_{i,k,j}}{I_{i,j}} \cdot (C_{i,k} + p_k) + c_{k,j} & \text{otherwise.} \end{cases} \quad (7)$$

Thereby, estimating the cost of an indirect connection of length n is reduced to calculating the cost of a path of length $n-1$ until we reach the basic case with one intermediate broker. Additionally, we also obtain a general decision criteria: a broker B_i prefers a direct link to another broker B_j (that is more than one hop away), if the direct communication costs are less than the calculated indirect costs, i.e., $c_{i,j} < C_{i,j}$.

4.2 Consensus Phase

In the previous section, we described how a single broker evaluates whether it is sensible to establish a direct link to another broker in its neighborhood. This decision is based on its own local cost-benefit analysis. However, this local decision may lead to an overall increase in costs. To avoid this, the broker seeks a consensus with the other brokers lying on the cycle that would be created by adding the proposed link. We call this *reconfiguration cycle* $R = (\mathcal{V}_R, \mathcal{E}_R)$. Since one edge of the cycle must be removed again to keep the topology acyclic, all brokers on R are directly affected by a subsequent reconfiguration. Choosing only this subset of brokers limits the overhead for finding a consensus.

After broker B decided to directly connect to another broker proposing the new link e_n, it asks every broker on the reconfiguration cycle to estimate the cost of the topology that would result from removing a single edge from R. We define the cost of the reconfiguration cycle R from the perspective of one broker B_i when removing edge e as follows

$$\text{cost}_R(e, B_i) = \sum_{B_j \in \mathcal{V}_R} I_{i,j} \cdot C_{i,j}, \quad (8)$$

where the cost $C_{i,j}$ is calculated on R without e. Accordingly, the aggregated cost of the topology is

$$\text{cost}_R(e) = \sum_{B_i \in \mathcal{V}_R} \text{cost}_R(e, B_i). \quad (9)$$

Having calculated $\text{cost}_R(e)$ for all edges on R, we determine the edge e_r that shall be removed such that

$$\text{cost}_R(e_r) = \min_{e \in \mathcal{E}_R} \{\text{cost}_R(e)\}. \quad (10)$$

Since e_r is chosen as the edge, whose removal leads to a topology causing the least cost, we gain the maximum benefit for the affected brokers when replacing e_r by the proposed new link e_n. However, if both links are identical (i.e., $e_n = e_r$), the consensus phase is aborted as it then seems to be unfavorable to add e_n to the topology at all.

4.3 Integration

In this section, we describe the integration of the heuristic presented in the previous sections into the publish/subscribe model. This covers the dissemination of the Bloom filters as well as the protocol for the consensus phase.

4.3.1 Broadcast Messages

Every broker B regularly broadcasts the Bloom filter of its cache in the time interval Δt to the brokers in its neighborhood $\mathcal{N}_\eta(B)$. Therefore, B sends a *broadcast message* to all its direct neighbors, it shares a link with. The broadcast contains a time-to-live (TTL) counter initialized with η and the Bloom filter representing B's cache entries added in the last interval. Additionally, it stores the path including the brokers and links the message has already passed together with the processing and communication costs, respectively.

When a broker receives a broadcast message, it first determines its common interest with the sender by comparing its cache contents to the Bloom filter stored in the message. Then, it updates the message by appending itself, the link the broadcast was received on, and the corresponding processing and communication costs to the message's path. The broadcast's TTL is decreased by one, and if it does not equal 0 the broker forwards the message to its neighbors. Finally, the broker starts the evaluation phase (as described in Sect. 4.1) to determine whether a direct connection to the broadcast's origin is sensible.

With every broadcast message a broker receives, it learns about its environment (i.e., the processing and communication costs as well as the common interest). Information about brokers in the neighborhood of a broker B may become obsolete when reconfigurations changed the topology in a way such that these brokers do not belong to the neighborhood anymore. Therefore, stale information is removed by a second-chance garbage collection algorithm based on the broadcast interval Δt.

In the beginning or as a result of a reconfiguration, the information about the neighborhood may not be sufficient for B_i to calculate $C_{i,j}$ for every node B_j it receives a broadcast message from. In this case, B_i cannot decide whether a reconfiguration is sensible and, thus, has to wait until it has gathered enough information about its neighborhood. Meanwhile, broadcast messages are still forwarded regularly, but subsequent evaluation or consensus phases are aborted.

4.3.2 Request Messages

While broadcast messages serve to gain knowledge about the local environment and are, thus, a prerequisite for the evaluation phase, *request messages* coordinate the consensus phase to finally decide whether the introduction of a new link is beneficial and which link has to be removed in turn. When broker B_i decides to add a new link connecting it with broker B_j, its decision is based on the last broadcast message it has received from B_j. Thus, the reconfiguration cycle R consists of the path the broadcast message was forwarded along and the new link that directly connects B_i and B_j.

Broker B_i starts the consensus phase by sending a request message to B_j along the reverse path of the broadcast message. The request message consists of a cost vector containing an element for every link in R. After receiving the request message, each broker on R adds its own costs for every link on R it has calculated according to Eq. 8. Then, the request is forwarded to the next neighbor broker on R until it reaches B_j. After adding its calculated costs B_j examines the resulting cost vector to determine the most expensive edge, whose removal leads to the best solution for all brokers on R. If this is the proposed new link between B_i and B_j, B_j discards the reconfiguration. Otherwise, B_j starts the reconfiguration phase.

5. EVALUATION

We evaluated the proposed heuristic by conducting simulation experiments. For creating physical network topologies we used the transit-stub model which produces Internet-like topologies. We used BRITE [8] to create 25 different topologies with 100 domains and over 10,000 nodes for the experiments. In all simulations, we placed 100 brokers randomly on the network. There are 50 different types of notifications, each produced by one publisher and subscribed to by subscribers connected to 9 different brokers. The distribution of clients to brokers was chosen probabilistically according to a load-value that was chosen randomly for each broker. The cache size of each broker is set to 8,192 entries and broadcast messages are sent every 250 simulation ticks. The size of the Bloom filters is 100,000 bits, and 5 different hash functions are used to create them.

We implemented our heuristic, which we call the *Cost and Interest heuristic (CI heuristic*) in the following, and chose decentralized versions of the MST and the MIT heuristic (called *ℓMST* and *ℓMIT*) described in Sect. 3.1 for comparison. Like the CI heuristic, they both rely on broadcast and request messages. While ℓMST only considers network costs, ℓMIT concentrates solely on the notification traffic and is, thus, similar to the approach by Baldoni et al. [1]. We conducted several experiments to compare the three heuristics. The goal of the experiments is to evaluate the following properties of the CI heuristic: its ability to adapt to changes in the system; the performance of the CI heuristic compared to ℓMST and ℓMIT; the relation between the costs caused by the heuristic and the savings gained by reconfigurations; and the effect on the performance of the different heuristics when changing the weights of the link and processing costs.

In the experiments, we varied different parameters with the default values given in parenthesis. The costs for the overlay links were normalized to values between $c_{\min}$ (= 0.0) and $c_{\max}$ (= 10.0) according to the underlying physical topology. The processing costs were also randomly chosen between $p_{\min}$ (= 0.0) and $p_{\max}$ (= 10.0) as well as the probability that a new client connects to a certain broker. The size of the neighborhood (in hops) is defined by η (= 4). Random values were chosen using a uniform distribution. Please note that doing this marks a very conservative setting, where the CI heuristic has less opportunities to optimize the costs.

In the following, we describe several experiments and discuss the results. Vertical bars in the plots show the 95% confidence interval. As a benchmark for the heuristics we used a static randomly generated overlay network.

5.1 Adaptivity

The algorithm's goal is to adapt the broker overlay topology according to dynamic changes in the network. We start with changes that affect the communication and processing costs as well as changes in the load values of the individual brokers. We want to measure, if the algorithm (i) is able to optimize the broker overlay to lower the total cost of forwarding and processing notifications and (ii) reacts to dynamic changes in the system.

To keep the cost for the heuristic reasonably low, we limit the neighborhood to $\eta = 4$ hops. The publication rate and the duration of the subscriptions is exponentially distributed with rate parameter 0.2 and 0.0004, respectively. This means that, on average, every 5 ticks a new notification is published by a publisher and every 2,500 ticks a subscriber

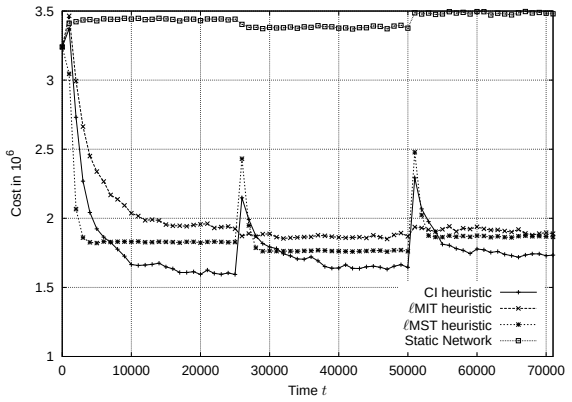

Figure 4: Accumulated costs with and without the different heuristics (95% confidence intervals were below 6% on average).

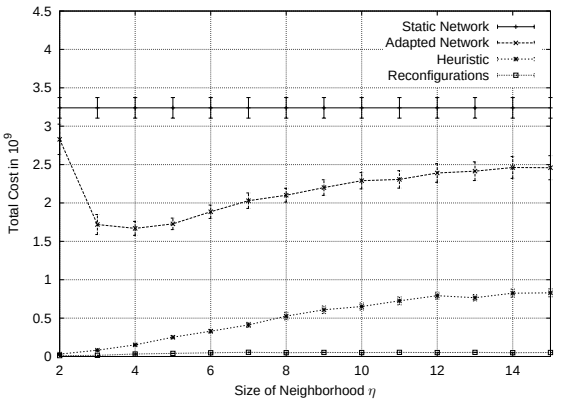

Figure 5: Costs of the static and the adapted network in comparison to the costs induced by the CI heuristic.

is removed from the system and replaced by a new subscriber at a randomly chosen broker. With 9 subscribers per job we get a relation of approximately 50 notifications per issued subscription. We start with the static broker network and apply the heuristics. At time $t_1 = 25,000$ and $t_2 = 50,000$ we completely change the cost of links and brokers, respectively. Every $1,000$ ticks we measure the communication and processing costs caused by forwarding notifications and by additional messages the heuristics produce.

In Fig. 4, one can see that all heuristics are able to quickly reduce the costs. When the topology is changed at t_1 and t_2, an increase in the costs is measured since the overlay network is optimized for the prior setting. After a short time, the heuristics manage to improve the overlay topology again. All heuristics manage to adapt the broker overlay and, thus, reduce the costs significantly compared to the static network. Changes in the system increase the costs, but the heuristics react very fast. The CI heuristic performs better than the other heuristics. However, the advantage gained depends on the distribution of cost and load values. In a following experiment, we analyze how the variation of cost value weights affects the effectiveness of the heuristic.

5.2 Cost of the Heuristic

In this experiment, we measure the cost when applying the CI heuristic and the cost of the static network. To learn about the extra load imposed by the heuristic, we also measure the costs induced by its application. We increase the size of the neighborhood by changing the value of η to see the trade-off between the improvement and the costs spent for gaining knowledge about a bigger neighborhood.

Figure 5 shows the accumulated costs after $25,000$ simulation ticks for the static network and the network that has been adapted by the CI heuristic. The biggest cost advantage is gained for $\eta = 4$, which is used in the following experiments. For bigger values of η the performance gain decreases. Apparently, although the brokers gain knowledge about a bigger neighborhood, the costs spent for the heuristic cannot be outweighed by potentially better reconfiguration decisions. Among other reasons, this relates to less meaningful estimations of common interest of brokers with a growing number of brokers on the reconfiguration cycle. Thus, knowledge about bigger neighborhoods does not necessarily lead to better results for the CI heuristic.

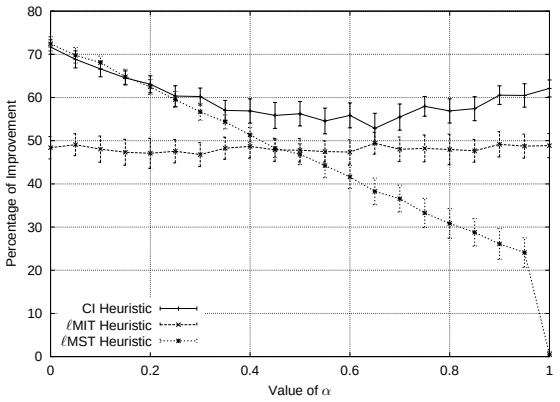

Figure 6: Improvement when increasing processing cost weight α.

As expected, the cost produced by sending broadcast messages grows when increasing the neighborhood size as the number of hops for each broadcast message is increased with η. The cost for executing the reconfigurations is negligibly small with less than 3% of the cost of the adapted network—even though longer reconfiguration paths are possible for bigger values of η. This is due to the fact, that less reconfigurations seem to be beneficial because of poor estimations of common interests.

5.3 Effect of Cost Distribution

In contrast to the ℓMST and the ℓMIT heuristic the CI heuristic considers the common interest, communication costs, and additionally the processing costs. In this experiment, we test the effect of different cost weights for the communication and processing costs on the performance of the heuristics. Therefore, we set $c_{\max}(\alpha) = (1-\alpha) \cdot \Gamma$ and $p_{\max}(\alpha) = \alpha \cdot \Gamma$ for $\Gamma = 20.0$ instead of $c_{\max} = p_{\max} = 10.0$ as done in the previous experiments. By varying α from 0 to 1 and measuring the cost improvement, we can observe how the overall costs develop as the processing costs increase and the communication costs decrease. The results plotted in Fig. 6 show that the ℓMST heuristic performs worse with growing

α, which reflects the fact that it does only take communication costs into account. The ℓMIT heuristic only concentrates on common interest and performs reasonably well when the communication costs are outweighing the processing costs. This does not change when the processing costs become more relevant since the clients are distributed uniformly such that processing and communication costs keep balance. The CI heuristic performs equally or better than the other heuristics for all α as it considers both costs.

6. DISCUSSION AND FUTURE WORK

Our evaluation shows that self-organizing broker overlay networks for publish/subscribe systems can increase their applicability in dynamic environments. We simulated three different heuristics and analyzed their behavior in different scenarios. Our CI heuristic shows the best performance, exhibiting robustness against parameter variations. This advantage results from considering link and processing costs as well as notification flows between brokers. The experiment in Sect. 5.1 shows that all heuristics were able to cope with drastic changes in the system.

We are currently working on more appropriate dynamics models, in order to analyze the behavior of the CI heuristic in scenarios where changes are less abrupt. Another interesting issue is the relation between the frequency of change and the effectiveness of the heuristic. There may be a degree of dynamics, at which the cost of constant reconfigurations exceeds the savings.

We used Bloom filters to determine the common interest of brokers. This way, we kept the size of the broadcast messages small. It is still an open question, as to which size and how many hash functions should be used for the Bloom filter in which scenario. It is part of future work to find ways to dynamically adapt both parameters in the system.

The experiments regarding the costs of the CI heuristic show that an increase of the neighborhood does not necessarily lead to better results regarding the effectiveness of the algorithm. Among other reasons, this is caused by the inherently inaccurate estimation $I_{i,j,k}$ of the common interest of brokers discussed in Sect. 4.1. Finding a better way to approximate this value (e.g., by using the conjunction of Bloom filters), may lead to better results.

7. CONCLUSIONS

Adaptive broker overlay networks are an important prerequisite for the application of pub/sub middleware in large-scale dynamic environments. The self-organizing broker topology presented in this paper is an important contribution to better suit pub/sub to new application domains that exhibit dynamics in user behavior and network characteristics. It quickly adapts to changes in the environment and reduces the costs significantly. The additional costs imposed by the heuristic are negligibly small with a reasonable neighborhood size. Furthermore, it exhibits a large flexibility in supporting different optimization goals.

We proved that finding an optimal configuration for the broker overlay is NP-hard—even in a static setting. Building a heuristic to solve this problem is, thus, a sensible approach. The heuristic presented does rely neither on manual intervention nor on global knowledge. Additionally, it is independent of the applied routing algorithm. The simulations conducted show that our approach is superior to other heuristics—even though the setting is rather conservative as we did not make any further assumption regarding the distribution of clients (e.g., locality) which would enable our heuristic to lower the costs even more.

8. REFERENCES

[1] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. A self-organizing crash-resilient topology management system for content-based publish/subscribe. In A. Carzaniga and P. Fenkam, editors, *3rd International Workshop on Distributed Event-Based Systems (DEBS'04)*, Edinburgh, Scotland, UK, May 2004. IEEE.

[2] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Subscription-driven self-organization in content-based publish/subscribe. Technical report, DIS, Mar. 2004.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[5] T. Hu. Optimum communication spanning trees. *SIAM Journal on Computing*, 3(3):188–195, 1974.

[6] H.-A. Jacobsen, editor. *2nd Intl. Workshop on Distributed Event-Based Systems (DEBS'03)*, San Diego, CA, USA, June 2003. ACM Press.

[7] D. S. Johnson, J. K. Lenstra, and A. H. G. R. Kan. The complexity of the network design problem. *Networks*, 8:279–285, 1978.

[8] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS '01)*, Cincinnati, Ohio, Aug. 2001.

[9] H. Parzyjegla, G. Mühl, and M. A. Jaeger. Reconfiguring publish/subscribe overlay topologies. In *5th Intl. Workshop on Distributed Event-based Systems (DEBS'06)*, page 29, Lisbon, Portugal, July 2006. IEEE Press.

[10] P. R. Pietzuch and J. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In Jacobsen [6].

[11] A. Rowstron and P. Druschel. Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems. In R. Guerraoui, editor, *18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 2218 of *LNCS*, pages 329–350, Heidelberg, Germany, 2001. Springer-Verlag.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *2001 Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[13] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In Jacobsen [6].