Community Clustering for Distributed Publish/Subscribe Systems

Wei Li[1,2] Songlin Hu[1] Jintao Li[1] Hans-Arno Jacobsen [3]
[1] *Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China*
[2] *Graduate University of Chinese Academy of Sciences, Beijing, China*
[3] *University of Toronto, Toronto, Canada*
{*liwei01, husonglin, jtli*}*@ict.ac.cn, jacobsen@eecg.toronto.edu*

***Abstract*—Optimized placement of clients in a distributed publish/subscribe system is an important technique to improve overall system efficiency. Current methods, like interest clustering or publisher placement, treat a client as, either a pure publisher, or subscriber, but not as both. Also, the cost of client movement is usually ignored. However, many applications based on publish/subscribe systems model clients as publisher and subscriber at the same time, which breaks the assumptions made by current approaches. Considering the complex dependency among clients, we propose a new *community-oriented* clustering approach, based on the forming of client clusters that exhibit intense communication relationships, while keeping client movement cost low. The evaluation based on a public data set shows that our method is efficient, adapts to different settings of experimental conditions, and wins over the popular interest clustering approach with respect to number of messages sent, propagation hop count and end-to-end latency.**

***Keywords*-community clustering, publish/subscribe, interest**

I. Introduction

A distributed publish/subscribe system (DPSS) [1], [19] offers a loosely coupled communication abstraction for networked applications. DPSS has recently witnessed a renewed rise in adoption, especially in many large-scale commercial systems, such as GooPS [31], Google's internal publish/subscribe system connecting applications across disparate data centers worldwide. Similarly, Yahoo's PNUTS distributed data store employs a publish/subscribe system to propagate updates across data centers worldwide [11].

In state-of-the-art DPSS, clients join the system by connecting to the closest edge broker [8], [19] or to any broker without restrictions [7], [17]. The net result of these policies is the potential for introducing an unpredictable number of overlay hops between publishing and subscribing end-points that may result in untolerable overloads and high message delivery delays. In order to implement optimized placement of publishers and subscribers on the broker overlay, recent approaches, such as interest clustering [30], [32] and publisher re-location [9], have been proposed.

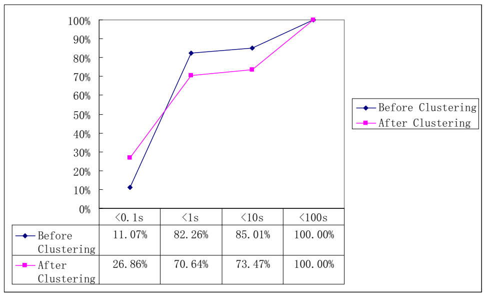
Interest clustering [30], [32] attempts to place subscribers with similar interest in close proximity on the publish/subscribe overlay, so as to reduce message processing and delivery delays by enabling path sharing among related subscribers. In contrast, publisher relocation aims to dynamically relocate publishers to shorten the message delivery distance [9]. Both approaches have proven effective in improving overall system performance.

However, these approaches neglect two important points. First, *complex communication relationships in interacting clients are not taken into account*. Existing approaches often treat clients as either publishing or subscribing end-points, but not as both. This simplifies analysis by modeling the communication relationship among clients as a bipartite graph. However, this assumption is in contrast to the needs of emerging applications employing publish/subscribe systems such as business process management [20], large-scale stream processing [36], and resource discovery [15], where clients play the role of a publisher and a subscriber at the same time. The communication relationship among clients in these scenarios is more complex and must be modeled with an acyclic graph rather than a bipartite graph. For example, in a service and resource discovery scenario employing publish/subscribe [15], a client acting as a Web service subscribes to service input and publishes output.

Current interest clustering (IC) [30], [32] and publisher relocation approaches [9] do not consider these more complex communication relationships in optimizing publish/subscribe interactions. In order to support this observation, we carry out a simple experiment to see how interest clustering effects a publish/subscribe-based application scenario dominated by complex communication relationships. For the experiment, we randomly deploy hundreds of Web services in a 20-node DPSS and trigger a number of service composition processes as described in [15]. We then record the latency of each message as the Web service composition is triggered. The message latency distribution is shown in Fig. 1. We observe that the latency of **82%** of the messages is within 1 s. We then use interest clustering based on k-means to group clients [32]. The results show that interest clustering cannot always increase the proportion of messages delivered within 1 s. On the contrary, the proportion of messages experiencing higher latency increases.

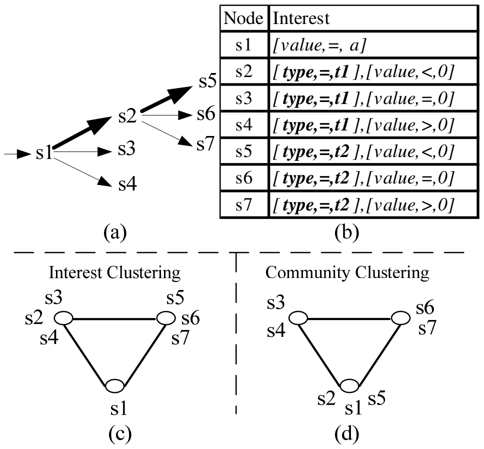
The reason is that although interest clustering can group clients with similar interest, it may inadvertently divide clients with intensive communication relationships into d-

This paper is accepted by IEEE International Conference on Cluster Computing 2012



	<0.1s	<1s	<10s	<100s
Before Clustering	11.07%	82.26%	85.01%	100.00%
After Clustering	26.86%	70.64%	73.47%	100.00%

Figure 1: Latency distribution before & after IC[32]



Node	Interest
s1	*[value,=, a]*
s2	*[**type,=,t1**],[value,<,0]*
s3	*[**type,=,t1**],[value,=,0]*
s4	*[**type,=,t1**],[value,>,0]*
s5	*[**type,=,t2**],[value,<,0]*
s6	*[**type,=,t2**],[value,=,0]*
s7	*[**type,=,t2**],[value,>,0]*

Figure 2: Interest clustering vs. community clustering

ifferent interest groups, thus increasing the average message delivery distance. Fig. 2-(a) shows a service composition for services $s1$ to $s7$ and Fig. 2-(b) shows the subscription issued by each service. At runtime, we find that there exists an intense communication relationship between $s1$, $s2$ and $s5$ for the given composition. That is, the number of messages exchanged among these three services is much larger than among the other services (marked in the figure with bold arrows.)

If we consider the interests of services $s2$, $s3$ and $s4$, which subscribe to messages matching $[type, =, t1]$, and $s5$, $s6$ and $s7$, which subscribe to messages matching $[type, =, t2]$, then an interest clustering algorithm may group these services into three clusters: $\{s1\}$, $\{s2, s3, s4\}$, and $\{s5, s6, s7\}$ (cf. Fig. 2-(c)). This grouping increases the delivery distance of messages among services $s1$, $s2$ and $s5$, thus negatively impacting the average message latency. If instead, we consider the complex communication relationship among $s1$, $s2$ and $s5$, and place these more intensely communicating clients together, the message delivery distance is shortened reducing average message delivery latency (cf. Fig. 2-(d)).

The second point existing approaches do not consider is *the cost of client relocation in optimizing the interactions in the publish/subscribe system*. Interest clustering and relocation algorithms change the location of clients [30]. For example, when a subscriber is relocated, the routing tables along the path from the source broker to the target broker have to be reconfigured, to guarantee correct delivery of notifications throughout the relocation process [16]. New system-internal messages need to be generated to update the routing tables of brokers, which brings about additional maintenance cost for client relocation [16]. This relocation cost must be considered in the overall optimization, as it is pointless to achieve better performance at too large a cost to bare for the system. Current approaches assess movement cost, but do not take the cost into account for optimizing client placement. For instance, Baldoni *et al.* proposed an algorithm for reorganizing the overlay and evaluates the notification cost [3]. Chockler *et al.* designed the SpiderCast protocol to effectively trade off the balance between average overlay degree and communication cost of event dissemination [10]. Besides experimentally assessing the cost, we suggest to carefully balance the trade-off between performance improvement and cost of client relocation in the algorithm.

To address this problem, we propose the *community-oriented clustering* approach to relocate clients. We define a community to be a set of clients with an intense communication relationship. Our approach addresses both of the issues outlined above. First, we explicitly take complex communication relationships among clients into account, especially those resulting from scenarios where clients act as publishers as well as subscribers. Second, we model the client relocation problem as a function optimization problem and treat the system maintenance cost as an additional constraint. We aim to improve overall system performance while keeping system maintenance cost low. In the end, clients within the same community, which may initially be scatter all over the overlay, are grouped together to shorten the message delivery path.

Our algorithm works in three phases. In Phase 1, we build the communication relationships on the overlay among clients and extract communities. In Phase 2, we place the communities to geographically close brokers. In this phase, we use a heuristic to balance the trade-off between reducing the average message hop while minimizing the maintenance overhead. In Phase 3, we apply load balancing in each geographic overlay cluster, introduced by [34].

We adopt a service composition scenario for the evaluation of our approach by experimenting with different types of network models based on public Web service composition data, generated by WSBen [27]. The relationships of Web service compositions can be classified into three types, including the scale-free network [5], the small-world network [33], [21], and the random network. We compare the performance of our approach against the interest clustering method and evaluate whether our approach can adapt to different physical overlay structures.

The main contributions of this paper are:

1) We propose the novel community-oriented clustering algorithm to relocate clients in a distributed publish/subscribe system, aiming to improve the performance of the system.
2) We propose a heuristic to trade off between performance improvement and system maintenance cost when relocating clients. Unlike existing approaches that measures the cost of relocating clients, our approach explicitly considers system maintenance cost as part of the optimization objective, to improve the system performance at a lower cost.
3) We conduct an extensive experimental evaluation and comparison to related approaches based on a public benchmark to demonstrate the effectiveness of our approach. The results show that our algorithm is effective under different experimental conditions.

This paper is organized as follows. Section 2 puts our work in the context of related approaches. Section 3 details our algorithms. An experimental evaluation and results are presented in Section 4.

II. Related Work

Clustering algorithms for DPSS: Current interest clustering algorithms are mainly focused on clustering subscriptions with similar interests. Riabov *et al.* defined a metric to measure the distance of subscriptions and compared several clustering methods such as K-means [32]. A subscription is represented as a vector, each element of which is 1 or 0. The non-zero elements correspond to the interest of subscribers in the whole event space. The distance between two subscriptions is calculated through the mean-square distance. Querzoni *et al.* [30] summarized the interest clustering algorithms for event routing in several systems.

Another algorithm from Milo *et al.* [22] based on topic-based publish/subscribe systems dynamically clusters subscriptions according to the benefit of adding a subscription to a cluster. This algorithm first defines the overall cost (OC) which is the sum of maintenance cost (MC) and dissemination cost (DC) in the system. If adding a subscription to a cluster or merging two clusters lowers the OC, meaning the system benefits, clustering takes place. Zhang *et al.* [35] proposed a hybrid approach based on grouping subscriptions to reduce matching cost. Cheung *et al.* [9] proposed a publisher placement algorithm, which optimizes system efficiency by publisher re-locating.

Community Structure of Complex Networks: It is being discovered that more and more systems tend to resemble complex networks, such as the network of interpersonal relations or the link relationships among Web pages [2]. In order to design a benchmark for service composition, Oh *et al.* [27] analyzed a large number of Web services and found that the relationships among services can be interpreted as a complex network. In this paper, we use Web service composition, as it is an emerging scenario for the use of publish/subscribe-style interactions [16], to evaluate our community-oriented clustering approach. The relationships of services in our evaluation are built based on Oh's insights.

The problem of how to determine communities in a network has been extensively studied. It has been found that there exists community structures in complex networks [12], [13], meaning that a network can be partitioned into several communities, and the intensity of links inside a community is stronger than to nodes outside the community. As the network of clients involved in a composition, based on Oh's analysis, resembles a complex network, a similar community structure exists among clients in a distributed publish/subscribe system.

Existing algorithms for extracting communities from networks can be classified into four types: Kernighan-Lin algorithms [18], information-theoretic algorithms [29], split algorithms [13] and assembly algorithms [6]. Newman *et al.* proposed an approach, which has been studied by many other researchers [25]. He defined a metric called modularity. Modularity indicates the difference between a good partitioning of a network and a random network, which exhibits no communities. Although the theory about modularity is still under development, the algorithm has proven effective in determining communities of interest. We adopt this algorithm in our work and construct communities of clients in the distributed publish/subscribe system based on the communication frequency of messages among clients

III. Community Clustering

The placement of clients on the broker federation affects the performance of the distributed publish/subscribe system. Current algorithms ignore the complex dependency among clients and the maintenance cost of relocation. Instead, we consider this complex dependency and model the client relocation problem as an optimization problem with maintenance cost as a constraint.

The objective function of our model is defined in Equation 1, where **m** denotes the ID of a message, **m** $\in$ **[1, M]**, **M** denotes the count of messages, and **c** denotes the ID of a client, **c** $\in$ **[1, C]**. We use the average travel distance of messages (the message ***m*** is routed hop by hop in the overlay, and the number of hops is calculated by ***Hop(m)***) to quantify the performance of the system. The average travel distance of messages is directly related to the amount of messages that need to be generated during delivery of notifications. It affects the average message latency, which determines the efficiency of the distributed publish/subscribe system.

$$\begin{aligned} &\min\left(\frac{\sum_{m=1}^{M} Hop(m)}{M}\right) \\ \text{s.t.}\quad &\min\left(\sum_{c=1}^{C} maintainCost(c)\right) \end{aligned} \tag{1}$$

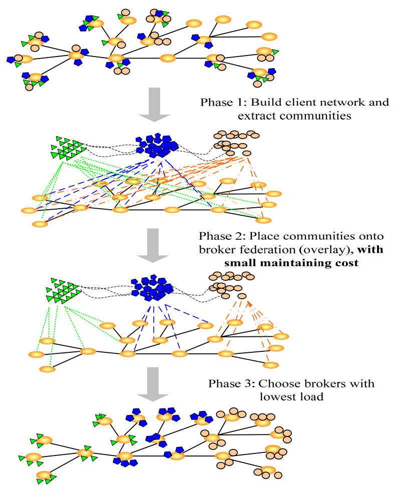

Phase 1: Build client network and extract communities
Phase 2: Place communities onto broker federation (overlay), with small maintaining cost
Phase 3: Choose brokers with lowest load

Figure 3: Overview of three phases of approach

In order to reduce the average travel distance, we determine the physical grouping of clients that exhibit an intense communication relationship. We call this approach the *community-oriented clustering* algorithm. The three phases of our algorithm are shown in Fig 3. First, the communication network of clients is built and communities of clients are extracted from the network. Second, the extracted communities are placed into geographically close brokers to shorten the message delivery distances. In addition, a heuristic method is introduced to balance the trade-off between reducing the average travel distance and minimizing the maintenance cost. Third, load balancing is performed in each geographical cluster. These steps are applied iteratively to obtain the optimized value given by Equation 1 to improve performance at small maintenance cost.

The larger the number of clients moved, the higher the maintenance cost. Therefore, we use the number of moving clients to estimate the maintenance cost in Equation 1. The cost of client movement within clusters is ignored, as it is much smaller than the cost of movement across clusters.

$$maintainCost(c) = \begin{cases} 1 & \text{if } c \text{ is moved across clusters} \\ 0 & \text{else} \end{cases} \quad (2)$$

A. Phase 1: Community Extraction

The communication relationship among publish/subscribe clients can be represented with a weighted directed graph $G = \{V, E, W\}$. The vertex set V represents the clients, and a directed link $l_{nk} \in E$ with the weight $w_{nk} \in W$ shows that client n sends w_{nk} publication messages to client k. During a certain period, w_{nk} can reflect the frequency of messages that n sends to k, which shows the communication relationship from n to k. Placing a set of intensely related clients in geographically close brokers shortens the distance of the majority of messages sent and thus reduces the propagation hops, the delivery latency, and the overall load in the system.

We extract the communities from the network of clients using Newman's algorithm [25]. Suppose that a communication network is divided into several communities and $e_{i,j}$ is the fraction of edges in the network that link vertices in community i to vertices in community j. In a specific community division, the overall fraction of edges within communities is $\sum_i e_{i,i}$. The sum $a_i = \sum_j e_{i,j}$ represents the probability that the vertices in community i is connected to by the other vertices. So in a network without regarding communities, the probability that the vertices in i have connections with the vertices in j is $a_i \times a_j$. The $modularity$, defined as $q = \sum_i (e_{i,i} - a_i^2)$, measures the variance that in a specific community division, the fraction of within-community edges minus the same quantity but with random connection edges among the vertices. The community structures are more evident in the network if q is bigger. Initially, we set every client as a community, and iteratively merge two communities into one and compute the new value of q. The iteration stops when q reaches its peak value, and the corresponding partitions are the extracted communities.

B. Phase 2: Community Clustering

In this phase, we group the communities onto the broker federation subject to the constraint of minimizing the maintenance cost. First, we define what geographically close brokers are. Usually the broker federation can be considered as a combination of several sets of physically close brokers. We call such set of brokers a cluster. For example, the brokers connecting to the same hub or located in the same intranet could be considered a cluster. The broker federation of a distributed publish/subscribe system can be viewed as a set of clusters connected via a backbone. We then adopt a heuristic-based method to place the communities onto clusters.

We introduce two terms: client-proportion and majority-place, used in our algorithm. We use two denotations here: $i \in [1, I]$ is the ID of a community and $s \in [1, S]$ is the ID of a cluster. The client-proportion $Pr(i, s)$ of community i in cluster s shows the ratio amount of the community in the specific cluster, specified in Equation 3. The majority-place $Mp(i)$ is the cluster that has the biggest Pr for community i, specified in Equation 4, or the cluster that has the most amount of clients of community i.

$$Pr(i, s) = \frac{\text{the count of clients of } i \text{ located in } s}{\text{the count of all clients of } s} \quad (3)$$

$$Mp(i) = s \qquad \text{while} \qquad s = \operatorname*{argmax}_{s'} Pr(i, s') \quad (4)$$

The maintenance cost is minimized if the clients of community i are clustered onto its majority-place $Mp(i)$.

Algorithm 1 Community Clustering

```
1: Initial a Queue $Q$;
2: Put all clusters $S$ into $Q$;
3: while ($!Q.empty()$) do
4:   $s = Q.removeHead()$;
5:   if ($s$ is not overloaded) then
6:     Sort the communities located in $s$ according to
       $Pr(i, s)$ in descending order;
7:     for (each community $i$ in $s$) do
8:       if ($i$ is placed) then
9:         $s'$ = getCluster(i);
           // get the cluster $i$ was previously placed to
10:        if Pr(i,s)>Pr(i,$s'$) then
11:          Place $i$ to $s$;
12:          if ($s'$ is not overloaded) then
13:            add $s'$ into $Q$;
14:          end if
15:        end if
16:      else
17:        Place $i$ to $s$;
18:      end if
19:    end for
20:  end if
21: end while
```

However, we can not cluster a community onto its majority-place regardless of the locations of other communities, otherwise the majority-place may be overloaded. Here, a cluster s is defined as overloaded when the amount of messages handled by it exceeds $V(s)$, specified in Equation 5, by a specific ratio. In the worst case, if a cluster is the majority-place of all communities, placing all communities in the same majority-place overloads the cluster, while other clusters remain empty. Alternately, the community can be clustered onto the cluster with the second biggest client-proportion, or the third, unless the community is overloaded. This means that we are able to guarantee better performance at the expense of slightly higher maintenance cost.

$$V(s) = \text{the amount of total messages} \times \frac{\text{the count of brokers in the cluster } s}{\text{the count of total brokers}} \quad (5)$$

Algorithm 1 specifies our majority-place-based community clustering method. In the initial state, each cluster has a ranked list recording the client-proportions of all communities in it (Line 6). In the loop, we place the communities in a cluster in descending order of client-proportions and stop the allocation when the cluster is overloaded (Line 17). A community i placed previously is re-placed to another cluster which has larger client-proportion (Line 10, 11).

Phase 1 and Phase 2 is iterated to determine the optimized community extraction and relocation policy. When a community is redivided into smaller communities, the performance of the system decreases and the maintenance cost is reduced. The loop termination condition is: if between two iteration, the ratio of performance decrease R_p is larger than the ratio of maintenance cost reduction R_m, which means that the loss of performance after redivision is bigger than the reduction in cost. Through the iteration, we aim to balance the trade-off between the performance improvement and the maintenance cost.

C. Phase 3: Load Balancing

After grouping a community to a cluster, we perform load balancing among the brokers within a cluster. In publish/subscribe systems, the load of a broker consists of two parts: the messages from the clients connecting to the broker and the messages from adjacent brokers. We take the first part into account in load balancing for two reasons: (1) After community-oriented clustering, the majority of messages are delivered within a few hops. For a broker, this helps to reduce the amount of messages from adjacent brokers. (2) This simplifies the calculation in the algorithm. The load of a broker is measured through the sum of the amount of messages sent by the clients connecting to the broker. A client moved from another cluster is allocated to the broker with the lowest load. A more accurate and complex solution for load balancing of brokers is introduced in [34], which is based on a data structure recording the historical matching information of publications that is used to predict the change of load when moving clients among brokers. We assume that the communication dependency among clients does not change very frequently. We could simply re-run the algorithm, if that assumption does not hold.

IV. Experiments

A. Experiment Setup

Platform and Metrics: We use *Padres*, an open source distributed publish/subscribe system developed by the Middleware Systems Research Group at the University of Toronto, as our experiment platform. Our community-oriented clustering algorithm is implemented as an independent module, which analyzes the logs of the whole system and determines the relocation places for the clients. The performance of the system is quantified through message hop counts and message propagation latency, and the maintenance cost is measured in terms of the number of clients moved across clusters. In addition, we also measure the amount of messages processed by brokers, especially those on the backbone of the system, to show the effect on load reduction.

Setup: We compare the performance of our community-oriented clustering algorithm under different experimental conditions with that of another popular type of clustering algorithm, the interest clustering algorithm [32]. Interest

http://www.msrg.utoronto.ca/projects/padres/

clustering in our experiments is applied as follows. Assume that the subscription set of client i is denoted as S_i. The similarity between client i and client j, D_{ij}, is calculated through Equation 6. The clients are grouped through a clustering method such as K-means [14].

$$D_{ij} = \frac{S_i \bigcap S_j}{S_i \bigcup S_j} \tag{6}$$

We use distributed service composition, a typical application of DPSS [15], as our application use case. Each service is a client deployed in *Padres*, while the input parameters are converted to subscriptions and the output parameters are converted to publications. We then issue random service compositions to simulate the execution of service compositions, and thus generate the workload for our experiments. We use a central monitor to accumulate all the logs from each publish/subscriber broker.

The factors that influence the results of our algorithm include the following aspects:

1. Relationship of Clients The communication relationships among clients effect the result of extracting communities. We use *WS-BEN* [27], a tool that produces Web services according to the features analyzed from about one thousand real Web services, to generate Web services for our experiment. In addition, the relationships of clients produced by *WS-BEN* tends to be considered as a scale-free network [5]. So for a more comprehensive understanding, we also generate two other types of network relationships, called a *small-world network* [33], [26] and a *random network*, to evaluate whether our algorithm is sensitive to different kinds of client relationships.

The small-world network is built as follows [26]: First, we number the clients. Each client subscribes to the messages from its k neighbors. Then, we add m communication links into the network by randomly choosing two clients and adding a link between them with probability p. Parameters k, m and p are constants set for each experiment. *The random network* is built as follows: Each client randomly subscribes to messages from other clients.

2. Topology of Overlay We generate two kinds of overlays to evaluate whether our algorithm is sensitive to the choice of topology. One kind is comprised of several clusters with different broker counts. We generate an overlay, each cluster of which has 3-8 brokers assigned to it randomly, with a backbone connecting all clusters. The other kind is comprised of clusters with an equal broker count.

3. Initial Location of Clients The initial location of clients is related to the calculation of moving cost. In our experiment, we test the efficiency of the algorithm in two kinds of distributions of clients in the overlay. One is that the clients connect to the overlay randomly, to achieve an overall balanced distribution on the whole overlay. The other is that the clients connect to just some clusters of the overlay and the other clusters remain empty, to achieve an extreme distribution on the whole overlay.

B. Experiment Results

We change the communication relationships among clients, the topologies of the overlay, and the initial location of clients to evaluate whether our algorithm is efficient and sufficiently general, meaning that it can achieve good performance under different experiment conditions.

Client Network: In this experiment, we construct three kinds of communication relationships among clients, including the scale-free network, the random network, and the small-world network. We generate an overlay randomly, as shown in Figure 4, which has 8 clusters and 45 brokers excluding the backbone, and about 200 Web services, which connect to the brokers of the overlay randomly. We compare the result of our algorithm with that of the interest clustering method [32].

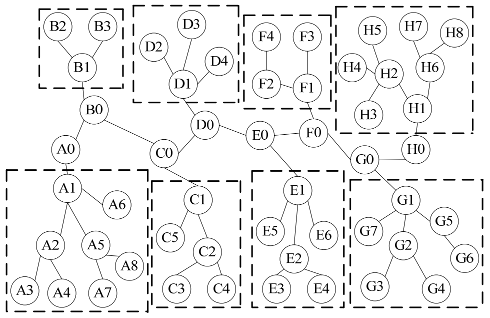
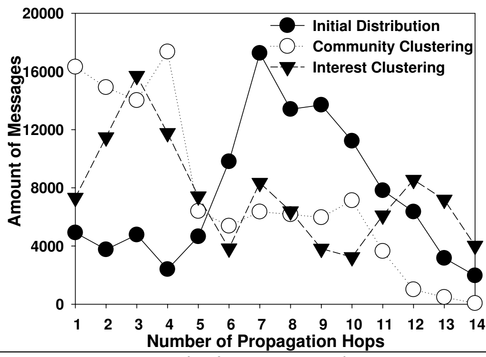


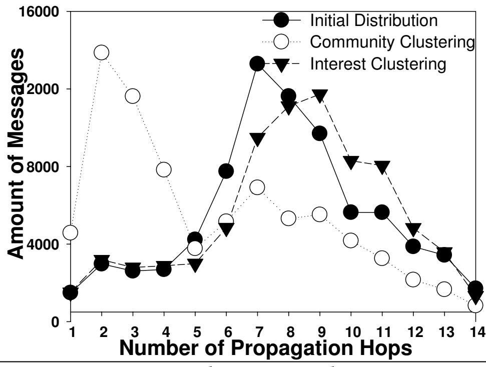
Figure 4: Randomly generated overlay

Figure 5 to Figure 7 show the distributions of propagation hops of messages in three kinds of communication networks. First, we can see that in the initial distribution, many messages need about 5 to 9 hops to be delivered (scale-free network: 56%, random network: 61%, small-world network: 54%, cf. tables in Figure 5 to Figure 7), and the peak values of Hops/Message amount lines are at 7 or 8 hops, as shown in Figure 5 to Figure 7. After the community clustering, the majority of messages are delivered within 4 hops (scale-free: 59%, random: 49%, small-world: 61%), and the values peak at 2 hops or 4 hops. Second, we can see that the interest clustering method is not efficient in these network models. It cannot significantly increase the proportion of messages that can be delivered within 4 hops in all models. It is interesting to note that in the scale-free network model, the proportion of messages delivered within 4 hops is increased to 44%. The reason is that in the scale-free network model, the community structure is not obvious and there exists some very popular clients that are linked by many other clients. Interest clustering may group clients together, which link to the same popular client, causing the distance among these clients to be shortened. Third, we find that the proportion of messages delivered within 4 hops is much bigger in the



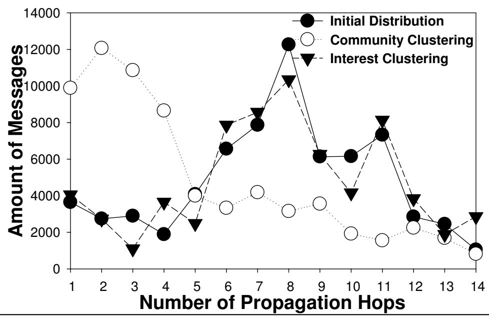
Scale-free Network			
	< 5	[5,9]	> 9
Initial distribution	**15%**	**56%**	29%
Interest clustering	**44%**	28%	28%
Community clustering	**59%**	29%	12%

Figure 5: *Scale-free network*: Message amount / hop number



Random network			
	< 5	[5,9]	> 9
Initial distribution	**13%**	**61%**	26%
Interest clustering	**14%**	52%	34%
Community clustering	**49%**	35%	16%

Figure 6: *Random network*: Message amount / hop number



Small-world network			
	< 5	[5,9]	> 9
Initial distribution	**16%**	**54%**	30%
Interest clustering	**17%**	52%	31%
Community clustering	**61%**	27%	12%

Figure 7: *Small-world network*: Message amount / hop number

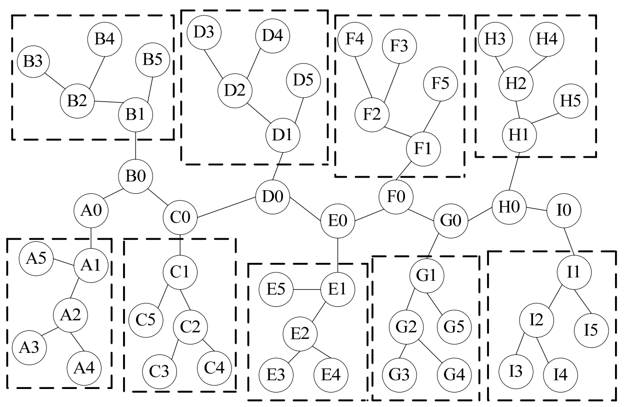


Figure 8: *Equal Cluster*: with equal number of brokers

small-world network than in the random network, as the community structures in the small-world network are more evident than in the random network. The modularity values q for the random network and the small-world network in our experiments are 0.35 and 0.65, respectively.

Table I shows the distributions of propagation latency of messages in the three network models. We can see that the percentage of messages delivered within 0.5 seconds has been greatly increased after community clustering (scale-free: from 7% to 33%, random: from 11% to 84%, small-world: from 28% to 87%); interest clustering cannot achieve this. In addition, we observe that the increase in the scale-free network is smaller than that in the other two networks. This is because the brokers hosting the very popular clients in the scale-free network may suffer from the relatively big amount of messages from and to the popular clients, which increases the processing time of delivering messages. Overall, community clustering is more efficient than interest clustering in all these three network models.

Table II shows the load reduction on the backbone of the overlay. We can see that community clustering can significantly reduce the load (message amount) of the brokers on the backbone and outperform interest clustering.

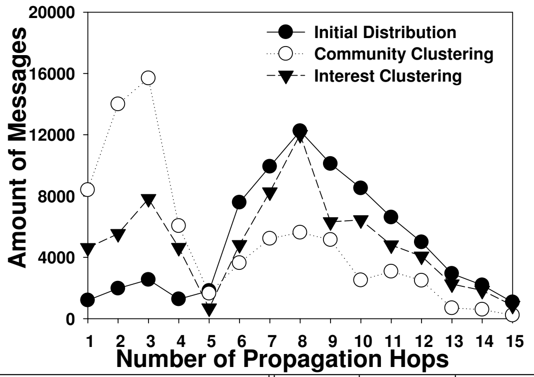
Sensitivity to Overlay Topology: In this experiment, we evaluate whether our algorithm is sensitive to the choice of overlay topology. Besides the overlay in the experiment above that contains clusters assigned to different number of brokers, we also generate another overlay comprised of clusters containing an equal number of brokers, shown in Figure 8, and the total number of brokers (excluding the backbone) equals the number of brokers of the previous overlay. Here, we just present the results carried out on *Equal Cluster*, as the experiments on the clusters assigned with different broker counts are already shown in the previous section.

Table I: Network model: latency distribution (s)

	Scale-free				*Random*				*Small-world*			
	< 0.1	[0.1,0.5)	[0.5,1]	> 1	< 0.1	[0.1,0.5)	[0.5,1]	> 1	< 0.1	[0.1,0.5)	[0.5,1]	> 1
Initial distribution	**2%**	**5%**	0.4%	**92.6%**	**7%**	**4%**	0.4%	**88.6%**	**7%**	**21%**	44%	28%
Interest clustering	**0.1%**	**3%**	5%	**91.9%**	**8%**	**12%**	3%	**77%**	**10%**	**25%**	35%	30%
Community clustering	**2%**	**31%**	16%	51%	**29%**	**55%**	11%	5%	**31%**	**56%**	10%	3%

Table II: Network model: average load on the backbone

	Scale-free		*Random*		*Small-world*	
	Average Amount	Reduction	Average Amount	Reduction	Average Amount	Reduction
Initial distribution	10230		14498		12305	
Interest clustering	7141	30%	14253	2%	10837	12%
Community clustering	3182	69%	10819	25%	4901	60%



	< 5	[6,10]	> 10
Initial Distribution	**12%**	**65%**	23%
Interest Clustering	**31%**	50%	19%
Community Clustering	**61%**	29%	10%

Figure 9: *Equal Cluster*: Message amount / hop number

Figure 9 shows the distributions of propagation hops of messages on *Equal Cluster*. We can see that the peak values of the Hops/Message amount lines are at 7 and 8, initially, and after community clustering the values peak at $2, 3$. The percentage of messages delivered within 5 hops has increased from 12% to 61%. In addition, the percentage of messages delivered within 0.1 seconds has increased from 8.6% to 39% (the right table in Figure 9). These results show that community clustering has significantly shortened the delivery distance and latency of messages. Although interest clustering can shorten the distance and latency, the change is not as big as for community clustering. Table IV shows the load reduction on the backbone of *Equal Cluster*. Community clustering has reduced 44% of the load, while interest clustering reduced 28%.

Table III: *Equal Cluster*: Latency distribution (s)

	< 0.1	[0.1,0.5)	[0.5,1]	> 1
Initial Distribution	**8.6%**	**75%**	11%	5.4%
Interest Clustering	**14%**	73%	8%	5%
Community Clustering	**39%**	52%	6%	3%

Table IV: *Equal Cluster*: Load reduction on backbone

	Average Amount	Reduction
Initial Distribution	14636	
Interest Clustering	10504	28%
Community Clustering	8157	44%

These sets of experiments show that our algorithm is adaptive to different communication network models of clients and different topologies of overlays, and the community clustering method outperforms the interest clustering method. These results show that our community clustering method is efficient and general.

V. Conclusions

In order to cope with complex client relationships in publish/subscribe application scenarios, where clients are simultaneously modeled as subscribers and as publishers and taking the system maintenance cost into account, we propose the community-oriented clustering approach to relocate clients in distributed publish/subscribe systems. The objective of our approach is to overcome shortcomings of current interest clustering approaches by achieving improved system performance at lower client movement cost. Compared with alternatives, our approach focuses on communication relationships among clients rather than entirely on client interests. Our approach extracts community structures from the communication relationships among clients and relocates clients according to this logical organization in the overlay rather than by forming physical subscriber groups. A heuristic algorithm based on the majority-place of a community is also put forward to balance the trade-off between the improvement in performance and the maintenance cost. Experiments under a variety of experimental conditions show that our community-oriented clustering approach is efficient and outperforms the popular interest clustering approach.

References

[1] M. K. Aguilera, et al. Matching events in a content-based subscription system. In *PODC '99*, pages 53–61, New York, NY, USA, 1999. ACM.

[2] D. Alderson, et al. Understanding internet topology: principles, models, and validation. *IEEE/ACM Transactions on Networking*, 13:1205–1218, 2005.

[3] R. Baldoni, et al. Efficient publish/subscribe through a self-organizing broker overlay and its application to siena. *Comput. J.*, 50:444–459, July 2007.

[4] G. Banavar, et al. An efficient multicast protocol for content-based publish-subscribe systems. In *ICDCS'99*, Washington, DC, USA, 1999. IEEE Computer Society.

[5] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. 286:509–512, October 1999.

[6] R. L. Breiger, S. A. Boorman, and P. Arabie. An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. *Journal of Mathematical Psychology*, 12(3), 1975.

[7] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19:332–383, August 2001.

[8] Y. Chen and K. Schwan. Opportunistic overlays: efficient content delivery in mobile ad hoc networks. In *Middleware '05*, pages 354–374, NY, USA, 2005.

[9] A. K. Y. Cheung and H.-A. Jacobsen. Publisher placement algorithms in content-based publish/subscribe. In *ICDCS '10*, pages 653–664, Washington, DC, USA, 2010.

[10] G. Chockler, et al. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *DEBS '07*, pages 14–25, New York, NY, USA, 2007. ACM.

[11] B. F. Cooper, et al. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1:1277–1288, August 2008.

[12] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *HYPERTEXT '98*, pages 225–234, New York, NY, USA, 1998. ACM.

[13] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–7826, June 2002.

[14] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1):100–108, 1979.

[15] S. Hu, et al. Distributed automatic service composition in large-scale systems. In *DEBS '08*, pages 233–244, New York, NY, USA, 2008. ACM.

[16] S. Hu, et al. Transactional mobility in distributed content-based publish/subscribe systems. In *ICDCS'09*, pages 101–110, 2009.

[17] R. S. Kazemzadeh and H.-A. Jacobsen. Reliable and highly available distributed publish/subscribe service. In *SRDS'09*, pages 41–50, Washington, DC, USA, 2009.

[18] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(1):291–307, 1970.

[19] G. Li, V. Muthusamy, and H.-A. Jacobsen. Adaptive content-based routing in general overlay topologies. In *Middleware '08*, pages 1–21, NY, USA, 2008.

[20] G. Li, V. Muthusamy, and H.-A. Jacobsen. A distributed service-oriented architecture for business process execution. *ACM Trans. Web*, 4(1):1–33, 2010.

[21] S. Milgram. The small world problem. *Psychology Today*, 1:61, 1967.

[22] T. Milo, T. Zur, and E. Verbin. Boosting topic-based publish-subscribe systems with dynamic clustering. In *SIGMOD '07*, pages 749–760, New York, NY, USA, 2007. ACM.

[23] G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.

[24] V. Muthusamy, et al. Sla-driven business process management in soa. In *CASCON '07*, pages 264–267, NY, USA, 2007.

[25] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 69(2 Pt 2), February 2004.

[26] M. E. J. Newman and D. J. Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263:341–346, 1999.

[27] S.-C. Oh. *Effective web-service composition in diverse and large-scale service networks*. PhD thesis, University Park, PA, USA, 2006. Adviser-Kumara, Soundar R.

[28] S. Pallickara, et al. Naradabrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids. In *Middleware '03*, pages 41–61, NY, USA, 2003.

[29] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[30] L. Querzoni. Interest clustering techniques for efficient event routing in large-scale settings. In *DEBS '08*, pages 13–22, New York, NY, USA, 2008. ACM.

[31] J. Reumann. Pub/sub at google. In *CANOE and EuroSys Summer School*, 2009.

[32] A. Riabov, et al. Clustering algorithms for content-based publication-subscription systems. In *ICDCS '02*, page 133, Washington, DC, USA, 2002. IEEE Computer Society.

[33] D. J. Watts and S. H. Strogatz. Collective dynamics of /'small-world/' networks. *Nature*, 393(6684):440–442, 1998.

[34] A. K. Yeung Cheung and H.-A. Jacobsen. Dynamic load balancing in distributed content-based publish/subscribe. In *Middleware '06*, pages 141–161, New York, NY, USA, 2006.

[35] R. Zhang and Y. C. Hu. Hyper: A hybrid approach to efficient content-based publish/subscribe. In *ICDCS '05*, pages 427–436, Washington, DC, USA, 2005. IEEE Computer Society.

[36] Y. Zhou, et al. Leveraging distributed publish/subscribe systems for scalable stream query processing, 2007.