Distributed Clustering with Limited Knowledge Sharing for Content Based Publish/Subscribe Systems

Emiliano Casalicchio and Federico Morabito

2006
Distributed Clustering with Limited Knowledge Sharing for Content Based Publish/Subscribe Systems

Emiliano Casalicchio∗, Federico Morabito†
emiliano.casalicchio@uniroma2.it, federico.morabito@telecomitalia.it

DELIS-TR-0010

Abstract

One of the main problems in content-based publish subscribe systems is to determine a subscription grouping such that the expressiveness and the efficiency of event routing and event matching mechanism trade-off. The idea is to create a set of multicast groups that better fit the subscription preferences, that allows fast and scalable event matching algorithms, and that allows a dynamic system reconfiguration. In this paper we propose a distributed clustering algorithm that allow to group subscription and that satisfy the main system requirement: expressiveness, efficiency, scalability and adaptability (dynamic system reconfiguration).

The novelty of the proposed solution is that the fully distributed clustering algorithm use only partial information on the system state (the event space and subscription space). This property drastically reduces the information stored on peers, reduces the cost of information exchange among peer and avoid the cost to maintain a global state.

Through a set of simulation we demonstrate the efficiency of the distributed solution.

1 Introduction

The publish/subscribe interaction paradigm provides subscribers with the ability to express their interest in an event or a pattern of events, in order to be notified subsequently of any event, generated by publishers, that matches their registered interest.

The different ways of specifying the events of interest have led to several subscription schemes: the channel-based [1], the topic-based (or subject-based) [19], and the content-based [2][21][24].

Content-based systems are relatively new and give users the ability to express their interest by specifying predicates. Publications are matched to submissions on the basis of their content. The content-based publish/subscribe paradigm is more powerful than topic based and is able to support rich subscription languages, by introducing a subscription scheme based on the actual content of the considered events. In other terms, events are not classified according to some pre-defined external criterion (e.g., topic name), but according to the properties of the events themselves.

The main issues to be considered in the design of content-based publish/subscribe systems are the following:

∗Universita’ di Roma “Tor Vergata”, Dipartimento di Informatica Sistemi e Produzione
†Telecomitalia
• maximize the expressiveness [2], that is the ability of the event notification service to provide powerful data model capable to capture information about the events, capable of expressing filter and pattern on the notifications of interest.

• Guarantee the scalability with respect to the number of subscriber and the published events. The main limitation to the scalability is given by the multicast techniques. The group-based multicast techniques, the most suitable in this context, are not readily applicable to resolve the scalability problem, as stated in [13].

• Adapt the network topology and the groups of common interest subscriptions to the dynamics of the system thus to maintain the desired level of performances. New subscriptions, changes in users preferences, node failures or unsubscriptions must be considered in the management of the system.

A trade off between expressiveness and scalability can be obtained providing an algorithm to properly groups subscription. Our idea is to create a set of multicast groups that better fit the subscription preferences, that allows fast and scalable event matching. In our solution we try also to address the dynamic system reconfiguration problem, allowing the use of a scalable and self-adaptable overlay network. Inspired to the solution proposed in [18] we decide also to cluster subscription in groups of similar interests, but the novelty of our approach is that we propose a fully distributed clustering algorithm that use only partial information on the system state (the event space and subscription space). This property drastically reduces the information stored on peers, reduces the cost of information exchange among peer and avoid the cost to maintain a global state. Our solution differs from those presented in [18] because the capability to adapt dynamically the groups of subscriber, as function of the system dynamic.

Our solution is based on the concept of application domain data model [CASA06], a model for the event space that have three main characteristics: flexibility, because any application domain can be represented; expressiveness, because complex preferences can be expressed through predicates over the set of attributes; representability, because the application domain data model can be represented by an N-dimensional cartesian space. The use of an N-dimensional cartesian space allow to use clustering algorithm to group subscriptions with similar preferences.

We define the algorithm to dynamically assign a subscription to the right cluster, and to reconfigure the actual clusters if needed. We propose an event matching algorithm.

The paper is organized as follow. In section 2 we report related works about content-based publish subscribe systems, event matching algorithms and clustering algorithm (both centralized and distributed). In section 4 we describe the problem, the subscription schema and the proposed solution for event matching and subscription management. In section 5 we describe the centralized solution adopted in the previous work. In section 6 we propose the distributed solution. In section 7 we demonstrate the efficiency of our solution.

2 Related works

2.1 Publish subscribe systems

Content-based publish/subscribe systems allow more complex and expressive subscriptions specifying pattern on the event content. A subscriber can express its preferences specifying multiple predicates as subscription and only those events whose content match all the predicates are notified to the subscriber. Examples of distributed content-based systems include Elvin [21], Siena [2], and Gryphon [6]. Elvin uses a central server which stores all the subscriptions and evaluates the subscriptions affected by the events.

Gryphon is a distributed system, in which a network of broker nodes is created and the events are distributed within the network. SIENA is designed to use both message brokering infrastructure and peer-to-peer overlay networks.
Peer-to-peer communication paradigm is widely used to exchange information among set of distributed peers. Advanced P2P systems provide more efficient lookups using structure in the logical overlay network formed by the peers. They implement distributed hash table functionalities and are referred to as Distributed Hash Tables (DHT’s). CAN [17], Chord [22], Pastry [19] are main example of such overlays. Such systems are designed for locating information based on exact id (e.g. the resource name). There are a variety of systems that extend this simple querying functionality to more complex queries [7, 20, 16]. Many subject-based publish/subscribe systems are implemented using the application layer multicast realized through DHT overlays. The main example is Scribe [4] that is built on top of Pastry [19]. In [15] the authors propose a type-based publish/subscribe system using similar rendezvous mechanism as in Scribe. In this system, content-based functionalities are added installing content-based filters close to the publishers.

Terpstra et al. [25] propose a content-based publish-subscribe system that partitions the event space among the peers in the system. Chord [22] is used to broadcast events and subscriptions to all nodes in the system.

Tam et al. [24] present a simple approach to build a distributed content-based publish subscribe system on top of a topic based pub/sub system, relying on DHT infrastructure. This approach is based on the concept of schema. Each schema is provided off line by the application designer on the basis of statistical measurement, users preferences, etc.

Riabov et al. [18] use a completely different approach, determining multicast groups from clustering of subscribers and solving a matching problem to find a proper multicast group for the generated events. In this solution, the multicast groups are built as clustering groups composed of subscriptions preferences. The published events are disseminated within the system to the proper subscribers using a matching mechanisms among the events and the interest subscribers. Riabov considers a static scenario (i.e. suppose that the subscriber are well known) in which the subscribers join the system at the same time and the pattern of subscriptions are well known.

Perng et al. [14] propose a solution where a particular content is described by a set of attributes and the publisher and subscriber must describe the published content and the subscription on the basis of these attributes. This solution provides a not flexible mechanism for content-based pub/sub. This solution is based on a DHT overlay network routing mechanism.

Meghdoot [8] implements a scalable architecture for content-based publish-subscribe system over a DHT based on CAN [17]. In this paper the authors propose a new method to balance the load among peers.

2.2 Large data-set clustering algorithms

In [23] the authors introduces the problem of combining multiple partitionings of a set of objects into a single consolidated clustering without accessing the features or algorithms that determined these partitioning. The authors propose three effective and efficient techniques for obtaining high-quality combiners (consensus functions). The First combiner induces a similarity measure from the partitionings and then reclusters the objects. The second combiner is based on hypergraph partitioning. The third one collapses groups of clusters into meta-clusters which then compete for each object to determine the combined clustering.

In [26] the authors present a method for k-means clustering when different sites contain different attributes for a common set of entities. Each site learns the cluster of each entity, but learns nothing about the attributes at other sites thus preserving privacy and security concerns.

In [23] the authors present Opossum, a novel similarity-based clustering approach based on constrained, weighted graph-partitioning. Opossum is particularly attuned to real-life market baskets, characterized by very high-dimensional, highly sparse customer-product matrices with positive ordinal attribute values and significant amount of outliers.
Since it is built on top of Metis, a well-known and highly efficient graph partitioning algorithm, it inherits the scalable and easily parallelizeable attributes of the latter algorithm.

In [12] authors present the Collective Hierarchical Clustering (CHC) algorithm for analyzing distributed, heterogeneous data. This algorithm first generates local cluster models and then combines them to generate the global cluster model of the data. The proposed algorithm runs in $O(|S|n^2)$ time, with a $O(|S|n)$ space requirement and $O(n)$ communication requirement, where $n$ is the number of elements in the data set and $|S|$ is the number of data sites. This approach shows significant improvement over naive methods with $O(n^2)$ communication costs in the case that the entire distance matrix is transmitted and $O(nm)$ communication costs to centralize the data, where $m$ is the total number of features. In [5] the authors propose a parallel implementation of the k-means clustering algorithm based on the message passing model. The proposed algorithm exploits the inherent data-parallelism in the k-means algorithm. The authors analytically show that the speedup and the scaleup of our algorithm approach the optimal as the number of data points increases.

### 2.3 Event matching algorithms

The event matching problem can be expressed as follows: given an event $e$ and a set of subscriptions $S$, determine all subscriptions in $S$ that are matched by $e$.

In the context of publish-subscribe, the ability to efficiently solve the event matching problem above introduced is a fundamental issue to be solved. Depending on which kind of subscription schema is adopted, different algorithms can be implemented to find the subscriptions that are satisfied by an event.

The major categories of state of the art matching algorithms are the predicate-indexing based algorithms or counting-based algorithms [?], [?], [?] and testing network-based algorithms [?], [?], [?].

An example of application of counting-based algorithm for matching the event with the subscriptions in a content-based system is the one described in [?]. In this work, the authors proposed an improved version of the counting algorithm, the Extended Counting Algorithm, permitting the system to perform a content-based forwarding mechanism. The Extended Counting Algorithm makes use of a content-based forwarding table, that represents a map between interfaces and predicates.

In [?], the authors introduced the clusters of similar subscriptions to solve the matching problem. The solution proposed is motivated by the need for being able to capture the dynamic aspect of the web information by notifying users of interesting events. The authors apply a predicate-indexing based algorithm on a set of pre-calculated clusters of subscriptions. They uses a propagation algorithm with the aims to limit the number of subscription to be verified by the matching events algorithms. Main result of such approach is the reduction of the time of inspection for the subscriptions.

A comparison of the two approaches for solving the vent matching problem has been faced in [?]. The authors showed that counting-based algorithms are likely to be more computationally expensive than tree-based algorithms. They also propose a new matching algorithm (RAPIDMatch) based on a tree-based approach, that partitions subscriptions offline based on their predicate characteristics, so that for a given event, it can quickly identify a small subset of relevant subscriptions and confine its search space.

In the Gryphon, the authors adopted a tree-based matching algorithm, that initially preprocesses the subscriptions into a matching tree where each non-leaf node contains a test, edges from that node represent the results of that test and leaf nodes contain a subscription. When an event occurs, the algorithm walks the matching tree by performing the test prescribed by each node and by following the edge that represents the result of the test. The set of matching subscriptions contains all those leaf nodes reached.

In [?] a matching algorithm similar to the Gryphon algorithm has been proposed. It also uses a matching tree. However, it allows a subscription to appear in more than a single leaf node. When an event occurs, the Gough algorithm generally needs to follow several
paths in the matching tree while Gryphon’s algorithm follows a single path. Therefore, the
Gryphon algorithm is more efficient.

In [7], the authors present an efficient, scalable solution to the matching problem, based
on tree-based matching approach. They prove that the time complexity is sublinear with
respect of the number of subscriptions. For obtaining such property, they preprocess the
predicates in order to reduce them as conjunctions of elementary tests, where this operation
is possible.

3 Clustering Algorithms

We consider subscriptions as abstract objects that have to be grouped into collection of
similar objects, these collections are named clusters. The concept of similarity depends on
the distance metrics used, to evaluate how distant are two objects and groups of objects.

The distance between two objects could be evaluated using different metrics, some
examples are Euclidean distance, Squared Euclidean distance, Chebychev distance. A de-
tailed description of metrics and their properties is reported in annex 9.

The distance between two clusters, measured by means of linkage rule, could be com-
puted in different way. Some examples are: Complete linkage (furthest neighbor) (FN),
Complete linkage (nearest neighbor) (NN), Unweighted pair-group average (UPGA). A
detailed description of linkage rules and their properties is reported in annex 9.

The clustering algorithms can be classified as hierarchical and non-hierarchical. On one
hand, hierarchical clustering algorithms joins together objects (e.g., animals) into succes-
sively larger clusters, using some measure of similarity (i.e., distance). On the other hand,
in the non-hierarchical method a position in the measurement is taken as central place and
distance is measured from such central point (seed). Identifying a right central position
is a big challenge and hence non-hierarchical methods are less popular. Non-hierarchical
clustering algorithms produce disjoint clusters and thus work well when a given set is com-
posed of a number of distinct classes or when the data description is "flat.

The main example of hierarchical clustering algorithm is the Minimal Spanning Tree.
In the minimal spanning tree algorithm at the begin each component is considered as a
cluster. Then the two closest cluster are merged in a single cluster. This process continue
until or when all point are grouped in a single cluster or when the number of desired cluster
are reached. The basic step of the MST algorithm are:

1. Assign each item to a cluster. At the begin each cluster contain just one item;
2. Repeat the following step until the number of desired cluster is obtained;
   
   a. For each cluster calculate the value of the centroid. Centroid is the means of
      the value of all point in a cluster;
   b. Calculate the intercluster distance matrix, where each element represent the
      distance between the centroid of two cluster;
   c. Discover the closest pair of clusters finding the minimum not null element of
      the previous distance matrix;
   d. Merge the two cluster of the previous step in a single cluster and decrement the
      number of cluster.

The main example of non-hierarchical clustering algorithm is the k-means. The k-
means algorithm begin finding k points represent a starting evaluation of the centroid of
the clusters. Remaining points are associated to the nearest centroid. This procedure is iterated
until all point are assigned to a cluster or until the maximum number of iteration is done.
So, given an input set and a fixed integer k (number of cluster) k-means algorithm return a
partition of the input set into k subset. This algorithm has four main step:
1. Select \( k \) initial cluster centroids, \( c_1, c_2, c_3, \ldots, c_k \);
2. Assign each object of the input set to the cluster that has the closest centroid;
3. For each cluster, recompute its centroid based on which elements are contained in;
4. Repeat step 2 and 3 until convergence is achieved.

4 Problem description

Riabov et al. in [18] define the subscription clustering problem, as a static preprocessing of subscription to compute a set of high quality multicast groups having as much commonality as possible, based on the totality of subscribers’ interests.

In these paper we define the dynamic subscription clustering problem, that is to compute, at run time, a set of multicast groups having as much communality as possible, and adapting these groups to the dynamics of the systems, that is to adapt the multicast groups to subscription arrival and leave.

As already stressed before, the groups of subscription created have to tradeoff the expressiveness and scalability. The solution have also to minimize the clusters management overhead.

The proposed solution is a distributed subscription clustering algorithm with the following properties:

1. guarantee the scalability with respect to the number of subscribers, publishers and with respect the size of the event space.
2. adapt the groups of common interest subscriptions to the dynamic of the system (arrival and leave of subscriptions).
3. provide a fully distributed solution avoiding the use of specialized servers
4. reduce the bandwidth consumption and nodes’ memory usage for clusters management and reorganization.

The right representation of the event space and then of the subscription space is vary important in the solution of the dynamic subscription clustering problem. The idea is to use a data model that provides an high level of expressiveness in any application domain, and that can be mapped into a \( N \)-dimensional cartesian space, manageable by clustering algorithms.

4.1 Application domain data model

The event space and subscription representation must be applicable to any application domain of interest. We chose to use a data model representation based on a set of application attributes, called application schema (or shortly schema)\(^1\). Attributes are characterized by their type, name, and constraints on possible values, specifying the general format of data and their possible values (within each application domain). Each application domain has its own schema, thus multiple domain schemas can be handled simultaneously by the same application (and also different applications may run on the same network). The proposed application domain data model allows subscribers to specify the subscription preferences, indicating the attributes and the related range values. The proposed data model, inspired by [18], permits to represent the application domain through an \( N \)-dimensional cartesian space \( \Omega \) in which each event can be uniquely described with a single multidimensional element \( \omega \) such that \( \omega \in \Omega \). Using the proposed abstraction, each subscription preference

\(^1\)We remark that our notion of schema is different from the concept introduced by Tam et al. in [24]
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Resource Capacity</td>
<td>percentage (%)</td>
<td>0..100</td>
</tr>
<tr>
<td>Cost (C)</td>
<td>dollars ($)</td>
<td>10..1000</td>
</tr>
<tr>
<td>Reputation (R)</td>
<td>a-dimensional</td>
<td>1..10</td>
</tr>
<tr>
<td>Factory (F)</td>
<td>string[15]</td>
<td>any</td>
</tr>
</tbody>
</table>

Table 1: The schema for resource discovery application domain

of the form \{[name attribute 1, range value 1], [name attribute 2, range value 2], [name N attribute, range value N]\} is represented as an hyper-rectangle into the event space. Table 1 shows an example of application schema, the possible preferences are defined expressing range values on four attributes: Available Resource Capacity and Cost may assume continuous values on their ranges, Reputation is a discrete numerical attribute and Factory is an alphanumerical string that may assume any possible value. An example of subscription is: \{[ARC, 50..85], [C, *], [R, 5], [F, ‘CISCO’]\}, that means the subscriber desires information on all CISCO devices, with an ARC between 50% and 85%, and reputation $R = 5$, without any constraint on the resource cost ($C = *$).

5 Centralized approach

In a previous work [3] we propose two centralized solutions to the problem of dynamically create multicast groups of subscriptions. The approach used were based on the regular grid partitioning of the Ω space and the clustering of the subscribers preferences. In the following we briefly resume these solutions.

5.1 Regular grid partitioning solution

This solution is based on the regular grid partitioning of the event space Ω. The event space Ω is partitioned in cells $c_i$ of granularity ($\alpha_1, \alpha_2, ..., \alpha_N$), where $\alpha_i$ is the step of the cell along the axes $i$. Each cell is represented by a centroid $cen(i)$ corresponding to the center of the cell. A multicast group $MG(i)$ is associated to each cell $c_i$. The subscribers in the multicast group $MG(i)$ are organized as leaf-nodes in the multicast tree $MTree(i)$, with root represented by the centroid $cen(i)$. The multicast tree $MTree(i)$ is responsible for disseminating the events falling into cell $c_i$ to the subscribers with preferences intersecting the cell $c_i$. The multicast groups are identified evaluating the intersections among the hyper-rectangle $b_{i,j}$ and the cells in the events space. The intercepted cells determine the multicast groups to which is associated the subscriber $v_i$. Different heuristics can be used to trade-off the efficiency and the infrastructure management cost when the number of intercepted cells is too high. For example, the cells with intercepting volume greater than a predefined threshold or the multicast groups with the greatest intercepting volume could be selected. When an event $ω$ is published, the Multicast Group Matching process searches for the cell containing the event $ω$ and delivers the event $ω$ through the associated multicast tree. To speed up the search process, the partitions of $Ω$ are organized in a R-Tree [9] data structure. R-Tree is a dynamic indexing structure for spatial search allowing logarithmic searching with the numbers of clusters. The Multicast Group Creation process is responsible to evaluate the inefficiency of the current configuration of the multicast groups. This process, periodically, reorganizes the multicast groups considering the dynamics of the subscriptions.

5.2 Clustering based solution

In this solution, we group the different subscriptions into clusters of different volume, by applying the Minimum Spanning Tree [11] algorithm. The set $I$ of the subscriptions is
partitioned into clusters composed of similar preferences. The hyper-rectangle with the minimum volume including all preferences in the cluster is associated to each cluster \(c_i\). These hyper-rectangles are organized into R-Tree data structures. In the clustering based solution, we will use without distinction the term hyper-rectangle to represent the corresponding cluster \(c_i\) and vice versa. Each cluster in the \(\Omega\) space is represented by the centroid \(\text{cen}(i) = (x_1, x_2, \ldots, x_N)\), where \(x_j, j = 1..N\) are the values of the coordinates. Note that in the clustering-based solution, the coordinates of the centroid are not static as in the regular partitioning solution, but they dynamically change depending on the the pattern of subscriptions. A multicast group \(MG(i)\) is associated to each cluster \(c_i\) and the multicast tree \(MTree(i)\) with the centroid \(\text{cen}(i)\) as root and with the subscribers in the multicast group \(MG(i)\) as leaf-nodes is built. \(MTree(i)\) is responsible for the events and the subscriptions within the range of the cluster \(c_i\). The Multicast Group Identification problem is solved by evaluating the intersections among the hyper-rectangle \(b_{ij}\), and the hyper-rectangles representing the clusters in the events space. The intercepted hyper-rectangles determine the multicast groups to which the subscriber \(v_i\) will be associated. The Multicast Group Matching process searches for the cluster containing the published event \(\omega\) and disseminates the event \(\omega\) through the associated multicast tree. The Multicast Group Creation process is responsible to evaluate the inefficiency due to the current configuration of the multicast groups and the dynamics of the subscribers and publishers, and to start, if necessary, the multicast group reorganization.

6 Distributed clustering with limited knowledge sharing

As in the centralized solution we suppose that the overlay network, used by the distributed clustering algorithm, provides multicast primitives, and in particular we suppose that for each multicast tree exists a node responsible for event delivery, typically called root node or rendezvous node. These hypothesis are true in the majority of the overlay network used in publish/subscribe systems.

The core idea of the distributed clustering is that each root node have enough, but limited information on the subscription space, thus to manage its own multicast group (that is a subscription’s cluster). To manage a cluster of subscription means to determine when a new subscription became member of the group and when the cluster change its composition (i.e. the compound elements).

The information owned by a node are classified as local knowledge and global knowledge. Each root node \(r_i\) manage a set \(S_i\) of subscriptions, that are the member of the multicast group. The root node knows also the coordinate of the centroid \(c_i\) of the cluster for which it is responsible. The tuple \((S_i, c_i)\) is the node \(r_i\)’s local knowledge.

We suppose also that each node knows: the coordinates of the centroid \(c_i\) of each cluster \(i\), where \(i \in [1, K]\) and \(K\) is the number of clusters; and the node-id of each root node. These information are represented with a tuple \((c_i, \text{node-id}(r_i))\) and are stored in a data structure named clustering table. The node-id is any information to identify a physical node in the network and depends on the overlay network used. The clustering table is the global knowledge of a node. Clustering table are dynamic and are updated and distributed by each root node. The updating process is distributed and a new version of clustering table is delivered to all nodes using a flooding mechanism. Then each node have the global knowledge of the systems (resumed by a very small set of information) and each root node have both global and local knowledge.

Maintaining the consistency of clustering table is an open problem. An approach is to apply the standard technique used in BGP routing [27] to maintain routing table consistency, or to apply consistency mechanism typical of p2p systems [10]. But also new techniques could be explored. In this paper we do not address the clustering table consistency problem.
6.1 Subscriptions clustering

When a node subscribes one or more preferences, it doesn’t know the node-ids which subscribe to, thus the subscriber node floods the message (flooded subscription) to the neighbors nodes, until a root node \( r_i \) is reached. The flooded subscription message differs from the subscription message because it doesn’t specify the node-id of a root node.

When a node \( r_i \) receive a flooded subscription \( I_j \):

1. compute the centroid \( b_j \) of the multidimensional rectangle \( I_j \).
2. compute the distance \( d(c_i, b_j) \), between \( c_i \) and the centroid \( b_j \), for all the entries \((c_i, \text{node-id}(r_i))\) in the clustering table.
3. Add the subscription \( I_j \) to the cluster \( C_k \) such that \( d(c_k, b_j) = \min_{i=1..N} d(c_i, b_j) \).
4. If \( i \neq k \), \( r_i \) communicates the node-id(\( r_k \)) to the subscriber, thus it can complete the subscription \( I_j \) sending a subscription message to \( r_k \). Otherwise the subscription \( I_j \) is for the node \( r_i \), and will be managed as described below.

When \( r_k \) receives the subscription \( I_j \) it have to:

4. recompute the new centroid of the cluster on the basis of the new added point \( b_j \);
5. If \( c_{k'} \) is the new centroid then, \( r_k \) compute the set \( G' \) of the clusters that get near: \( G' = \{ c_k \text{ s.t. } h \in [1,N] \text{ and } d(c_{k'}, c_h) < d(c_k, c_h) \} \). If \( G' \neq \emptyset \), then:
   a. for each cluster \( h \) that get near the cluster \( k \), \( r_k \) and \( r_h \) share their local knowledge, exchanging respectively the tuples \((S_k, c_k)\) and \((S_h, c_h)\), \( \forall h \in G' \);
   b. \( \forall h \in G', r_k \) and \( r_h \) apply the clustering algorithm to create \(|G'| + 1 \) clusters.
   c. \( \forall h \in G', r_k \) and \( r_h \) update their clustering table and flood it to the other nodes;
   d. \( \forall h \in G', r_k \) and all \( r_h \) apply recursively step 5.
6. Else, if \( G' = \emptyset \), the new subscription do not modify the system state and change only the local knowledge of node \( r_k \).

Steps (1-3) allow to match the new subscription with a multicast groups, that is, in term of clustering algorithm, to find the proper cluster to join. Steps (4-6) allow to reorganize the clusters on the basis of the new system state, determined by arrival and leave of subscription.

Step 5 is recursively repeated and can enter a loop. A possible limitation role is:

1. introduce a threshold \( \delta \), in the creation of the set \( G' \): \( G' = \{ c_j \text{ s.t. } j \in [1,K] \text{ and } d(c_{k'}, c_j) < \delta d(c_k, c_j) \} \), where \( \delta < 1 \). \( \delta \) can be a constant as it can be a function of the system efficiency.
2. introduce a threshold \( \rho \) to limit the number of recursions.

6.2 Ambiguous subscription management

There are some particular subscription that are ambiguous for the clustering algorithm, that is the chose operated by the clustering algorithm could lead to an undesired result. Considering the scheme in table 1, examples of ambiguous subscriptions are: \{[ARC,10..90],[C,*],[R,5],[F,'CISCO']\} and \{[ARC,70],[C,30..200],[R,*],[F,*]\}. The first subscription is ambiguous respect the attribute ARC and C, because the ARC and C coordinate of the centroid are not very representative, indeed the subscription should belong to all the clusters respect the attribute C an to more than one cluster respect the attribute ARC. On the other hand, the second subscription is ambiguous respect the attribute C,R and F.
Then in this work we propose a disambiguation technique for ambiguous subscriptions. The idea is to fragment a subscription into many parts, managed as independent subscriptions issued by the same subscriber (obviously these fragmentation is transparent to the subscriber). To decide how and when to fragment a subscription first of all we normalize the attributes’ ranges to [0,1], both for numerical and string type attributes. Then we introduce two parameters, \( \alpha \) and \( \beta \): \( \alpha \) is a threshold introduced to decide the dimensions (the attributes) respect which fragment the subscription; \( \beta \) is a threshold introduced to decide in how many fragment divide the subscription.

The threshold \( \alpha \) range from 0 to 1: smaller is \( \alpha \) higher is the probability is considered ambiguous, viceversa higher is \( \alpha \) lower is the probability that a subscription is considered ambiguous. The subscription is considered ambiguous if it exceed the threshold \( \alpha \) for at least one attribute. For example the subscription \{[ARC,10..90],[C,*],[R,5],[F,’CISCO’]\} could be normalized as \{[ARC,0.8],[C,1],[R,0.1],[F,10E-20]\}. If we chose \( \alpha = 0.7 \) the subscription must be partitioned respect to the attributes C and ARC. The parameter \( \beta \) range from 0 to 1 and allow us to determinate in how many part segment the subscription. Greater is \( \beta \) lower is the number of segment, lower is \( \beta \) greater is the number of segment. When the system find an ambiguous subscription, for each ambiguous attribute \( i \) we calculate the number of segment:

\[
\sum_{i=1}^{a} (n_i + 1) \text{ or } \sum_{i=1}^{a} n_i
\]

The new range of the \( i^{th} \) attribute \( range_{Attribute_i} \cdot \beta \), plus an additional subscription to cover all the original range if needed. At the end, from one initial ambiguous subscription, we obtain \( \sum_{i=1}^{a} n_i \) sub-subscription where \( a \) is the number of ambiguous attributes.

### 6.3 Event matching

A new generated event \( e_i \) must be matched to one or more multicast tree, before the delivering. To deliver \( e_i \), we chose the multicast tree \( MT_k \) such that \( d(e_i, r_k) = \min_{j=1..K} d(e_i, r_j) \).

Other combined metrics could be investigated.

Two solutions can be used to solve the event matching problem. The first solution require that each publisher know the clustering table, and then resolve the event matching problem by itself. The alternative is that the event \( e_i \) is flooded until a root node is encountered and then the event matching is determined.

### 7 Experimental results

In this section we evaluate the performance of the proposed solution respect to the performance metric of interest in a content-based publish subscribe system. These performance metrics are:

- **False Negative (\( fn \))**: the fraction of the events that a subscriber was expected to receive and that it does not receive.
- **False Positive (\( fp \))**: the fraction of the undesired events that a subscriber receive
- **Duplicated events (\( de \))**: the fraction of duplicated events received by a subscriber.

Having a number of false negative greater then 0 is undesired. Thus the proposed solution must guarantee \( fn = 0 \) and minimize (or reduce) \( fp \) and \( de \).

If we deliver the event \( e \) to a multicast group in any case (without filtering) we don’t have false negative but we have only false positive.

If we deliver an event \( e \) to a designated multicast group (with centroid \( c \)) only if \( d(c,e) < \alpha \cdot \max d \) where \( \max d \) is the maximum distance between a subscription and the centroid, we experiment some false negative. If \( \alpha \) increase the number of false positive increase and the number of false negative decrease. The we expect to find a value of \( \alpha \) that tradeoffs the number of false positive and false negative.
First of all we test the random subscriptions generator and the events generator. We show how subscriptions are distributed in a 3D space. What emerge from the experiment (see figures 1 and 2) is that subscription are fairly distributed in the space, covering all the dimension over all the range.

![Random generation of 500 subscriptions. Subscription are represented by rectangles and their centroids are represented by points.](image)

Figure 1: Random generation of 500 subscriptions. Subscription are represented by rectangles and their centroids are represented by points.

![Random generation of 1000 subscriptions. Subscription are represented by rectangles and their centroids are represented by points.](image)

Figure 2: Random generation of 1000 subscriptions. Subscription are represented by rectangles and their centroids are represented by points.

### 7.1 Sensibility analysis

The second analysis that we perform is the evaluation of the sensibility of the clustering algorithms to the set of subscriptions and to the number of clusters. Experiments are organized as follows:

1. we fix the clustering algorithm (e.g. MST);
2. we chose the Euclidean distance metric;
3. we generate 1000 subscriptions and evaluate the number of false positives and false negatives;
4. we repeat steps 1-3 for 20 different value of the seeds used as input for the random number generator;
5. in case of k-means we repeat steps 1-3 for a different value of $k$ and in particular $k = 5, 50, 100$;
6. We repeat steps 1-5 for a different clustering algorithms (e.g. K-means).

The results are shown in figures 3 and 4 and show how our work is insensitive from the set of subscription.

Figure 3: Sensibility of the Minimum Spanning Tree clustering algorithm to the seed value

Figure 4: Sensibility of the \(k\)-means clustering algorithm to the number of clusters \(k\) and to the seed value

7.2 Subscription disambiguation algorithm

In these experiments we evaluate the sensitivity of the disambiguation algorithm to the coefficients \(\alpha\) and \(\beta\). \(\alpha\) range from 0 to 1 and is a threshold coefficient that allows to trigger when a subscription is ambiguous. Greater \(\alpha\) less is the probability that a subscription is ambiguous. \(\beta\) range from 0 to 1 and indicate in how many sub-subscription an ambiguous subscription is divided/fragmented. Experiments are organized as follows:

- We fix the clustering algorithm (e.g. MST);
- We fix the seed for the random number generator.
- In case of K-means we fix the number of clusters (50) or we limit the maximum number of clusters in case of MST;
- We generate 1000 subscriptions and evaluate the number of false positives, false negatives and duplicates;
we repeat the above 4 steps first fixing $\beta = 0.5$ and ranging $\alpha$ from 0.25 to 0.75, and then fixing $\alpha = 0.5$ and ranging $\beta$ from 0.25 to 0.75.

In picture 5 fixing $\alpha = 0.5$ we consider ambiguous all subscriptions which have some attribute greater than the half of the maximum attribute value. For grater $\beta$ the original subscription is segmented into less sub-subscription. So when $\beta$ go to 0 the value of the duplicates grow because the probability that two or more sub-subscription finish in the same cluster is greater. In fact there are many subscription and the sub-subscription are more close each to another.

The number of false negative decrease because with more probability a sub-subscription finish in a more appropriate cluster and because different parts of the same subscription finish in more than one cluster, receiving more event. In picture 6 we fixing $\beta = 0.5$. For greater $\alpha$ the number of duplicate decrease because with less probability a subscription is considered ambiguous and is subdivided. The number of false negative is greater because greater is the probability that a very wide subscription, approximated through centroid, finish in a non appropriate cluster. When $\alpha$ is near to 1, practically, the ambiguity is not controlled. So the value of duplicated is close to 0 and the number of false negative is great. With lower value of $\alpha$, for example 0.25, the number of duplicate grow while the number of false negative decrease considerably because the possibility of capture an event is higher.

Figure 5: Performance metric fixing $\alpha$ value to 0.5

Figure 6: Performance metric fixing $\beta$ value to 0.5
8 Conclusion

In this paper we have propose a distributed clustering algorithm that use a limited amount of information. Such solution is ideal to disseminate information in content-based publish subscribe systems limiting the number of false positive experimented by the users.

Our solution is designed to be independent on the clustering algorithm used, indeed we show the results for both hierarchical and non-hierarchical algorithm. As representative of this class of clustering algorithm we conduct our experiment with the MST and k-means.

The main results is that its possible to tune the parameters of the disambiguation algorithm thus to have a number of false positive and false negative less the the 5% and without duplicated messages.

There are two open problem that should be addressed: the first is about the maintenance of the clustering table consistency, and the second is about the evaluation of the consumed bandwidth to run our distributed clustering algorithm with limited knowledge sharing. These are two correlated topic and will be investigate in a future work.

9 Acknowledgment

We would like to thanks Eng. Alessandro Vinciarelli of the University of Roma “Tor Vergata” that help us in the development of the simulator.

References


Annex 1: Distance Metrics

In this annex are reported the distance metrics used in our experiments.

Euclidean distance. This is probably the most commonly chosen type of distance. It simply is the geometric distance in the multidimensional space. It is computed as:

$$\text{distance}(x, y) = \left(\sum_i (x_i - y_i)^2\right)^{0.5}$$

Note that Euclidean (and squared Euclidean) distances are usually computed from raw data, and not from standardized data. This method has certain advantages (e.g., the distance between any two objects is not affected by the addition of new objects to the analysis, which may be outliers). However, the distances can be greatly affected by differences in scale among the dimensions from which the distances are computed. For example, if one of the dimensions denotes a measured length in centimeters, and you then convert it to millimeters (by multiplying the values by 10), the resulting Euclidean or squared Euclidean distances (computed from multiple dimensions) can be greatly affected (i.e., biased by those dimensions which have a larger scale), and consequently, the results of cluster analysis may be very different. Generally, it is good practice to transform the dimensions so they have similar scales.

Squared Euclidean distance. You may want to square the standard Euclidean distance in order to place progressively greater weight on objects that are further apart. This distance is computed as (see also the note in the previous paragraph):

$$\text{distance}(x, y) = \sum_i (x_i - y_i)^2$$

City-block (Manhattan) distance. This distance is simply the average difference across dimensions. In most cases, this distance measure yields results similar to the simple Euclidean distance. However, note that in this measure, the effect of single large differences (outliers) is dampened (since they are not squared). The city-block distance is computed as:

$$\text{distance}(x, y) = \sum_i |x_i - y_i|$$
Chebychev distance. This distance measure may be appropriate in cases when one wants to define two objects as "different" if they are different on any one of the dimensions. The Chebychev distance is computed as:

$$distance(x, y) = \max_i |x_i - y_i|$$

Power distance. Sometimes one may want to increase or decrease the progressive weight that is placed on dimensions on which the respective objects are very different. This can be accomplished via the power distance. The power distance is computed as:

$$distance(x, y) = \left(\sum_i |x_i - y_i|^p \right)^{1/r}$$

where $r$ and $p$ are user-defined parameters. A few example calculations may demonstrate how this measure "behaves." Parameter $p$ controls the progressive weight that is placed on differences on individual dimensions, parameter $r$ controls the progressive weight that is placed on larger differences between objects. If $r$ and $p$ are equal to 2, then this distance is equal to the Euclidean distance.

Percent disagreement. This measure is particularly useful if the data for the dimensions included in the analysis are categorical in nature. This distance is computed as:

$$distance(x, y) = \frac{\text{Number of } x_i \neq y_i}{\text{Number of dimensions}}$$

Annex 2: Linkage Rules

In this annex are reported the linkage rules used in our experiments.

Single linkage (nearest neighbor). As described above, in this method the distance between two clusters is determined by the distance of the two closest objects (nearest neighbors) in the different clusters. This rule will, in a sense, string objects together to form clusters, and the resulting clusters tend to represent long "chains."

Complete linkage (furthest neighbor). In this method, the distances between clusters are determined by the greatest distance between any two objects in the different clusters (i.e., by the "furthest neighbors"). This method usually performs quite well in cases when the objects actually form naturally distinct "clumps." If the clusters tend to be somehow elongated or of a "chain" type nature, then this method is inappropriate.

Unweighted pair-group average. In this method, the distance between two clusters is calculated as the average distance between all pairs of objects in the two different clusters. This method is also very efficient when the objects form natural distinct "clumps," however, it performs equally well with elongated, "chain" type clusters.

Weighted pair-group average. This method is identical to the unweighted pair-group average method, except that in the computations, the size of the respective clusters (i.e., the number of objects contained in them) is used as a weight. Thus, this method (rather than the previous method) should be used when the cluster sizes are suspected to be greatly uneven.

Unweighted pair-group centroid. The centroid of a cluster is the average point in the multidimensional space defined by the dimensions. In a sense, it is the center of gravity for the respective cluster. In this method, the distance between two clusters is determined as the difference between centroids.
Weighted pair-group centroid (median). This method is identical to the previous one, except that weighting is introduced into the computations to take into consideration differences in cluster sizes (i.e., the number of objects contained in them). Thus, when there are (or one suspects there to be) considerable differences in cluster sizes, this method is preferable to the previous one.

Ward’s method. This method is distinct from all other methods because it uses an analysis of variance approach to evaluate the distances between clusters. In short, this method attempts to minimize the Sum of Squares (SS) of any two (hypothetical) clusters that can be formed at each step. In general, this method is regarded as very efficient, however, it tends to create clusters of small size.