Brief Announcement: Weighted Partial Message Matching for Implicit Multicast Systems ⋆

William Culhane[1], K. R. Jayaram[2], and Patrick Eugster[1]

[1] Purdue University.{wculhane,peugster}@purdue.edu
[2] HP Labs. jayaramkr@hp.com

In *implicit multicast*, receiving processes delineate the messages they wish to receive by specifying predicates, also called *filters*, on the message's content. In *weighted partial* message matching, distributed applications using implicit multicast protocols do not require that a message match all the elementary constraints constituting the filter. Typically, receiving processes in such applications associate a non-negative weight to each constraint and require that the *match score*, i.e., sum of the weights of matching constraints, exceeds a threshold value. In this paper, we consider top-k weighted partial matching, where a process is interested in multicasting a message only to $k < n$ other processes corresponding to the top-k match scores. This is a fundamental problem underlying online advertising platforms, mobile social networks, online dating, etc.

A message m is a set of attribute/interval pairs $\{a_1 : [v_1, v_1'], \ldots, a_k : [v_k, v_k']\}$. We consider filters to be *conjunctions* of interval constraints, each of which has an associated weight. A filter with n constraints on n attributes is represented as a predicate $\phi = a_1 \in [v_1, v_1'] : w_1 \wedge \ldots \wedge a_n \in [v_n, v_n'] : w_n$. Interval filters are as expressive as regular filters, e.g., `x>1000` where `x` is an integer attribute can be expressed as `x`$\in$ [`1001`, `MAX_INT`] and equalities like `x=1000` can be modeled as `x`$\in$ [`1000`, `1000`]. We use interval filters because they increase efficiency (as we demonstrate in this paper). Given a message m and a filter $\phi = \bigwedge_{i=1}^{n} \delta_i : w_i$, where $\delta_i = a_i \in [v_i, v_i']$, the match score $score(\phi, m) = \sum_{i=1}^{m} w_i \mid \delta_i(m) = \text{TRUE}$. Given a set of filters Θ, and a message m, the top-k weighted partial matching set $\Phi \subseteq \Theta$ is defined as $\Phi = \{\phi \mid score(\phi, m) > 0 \wedge score(\phi, m) > score(\phi', m)\ \forall\ \phi' \in \Theta \setminus \Phi\}$ and $|\Phi| = k$.

The key strategy adopted by our matching algorithm is to consider filters *per attribute*. A broker in an implicit multicast system receives filters from senders and other brokers connected to it. We assume that an incoming filter from client c_i is uniquely identifiable by an identifier denoted by fid. We consider the cases where weights are (1) specified by receivers and are associated with the elementary constraints of a filter, and (2) specified by the sender and attached to the message overriding any weights associated by receivers with filters (shown in Line 24 of Figure 1).

Our algorithm uses a two-level index for filters. Constraints on attributes are indexed at the attribute-level in interval trees, which are in turn indexed by attribute name in a "master index" hash map (Line 1). An incoming filter is split

⋆ Funded by US NSF grants 0644013 and 0834529, DARPA grant N11AP20014, and by Purdue Research Foundation grant 205434.

by attribute name. The constraint for each attribute is added to its respective interval tree (Line 7). If no interval tree exists for an attribute, a new interval tree with the new constraint is created for that attribute (Line 9) and inserted into the master index (Line 11). Removing filters requires removing constraints from their interval trees and removing empty trees from the master index.

Our algorithm tracks filters' match scores via a hash map called *smap* (Line 13). For matching, it retrieves the identifiers and weights for each attribute's filter matching constraints (Line 18). The filters' aggregate scores are updated in *smap* by adding the weight of the matched constraint (Lines 22 and 24). If senders specify weights on attributes, the weights retrieved from the interval tree are discarded (Line 24). After calculating the scores using all filters on the message, the entries in *smap* are inserted into the tree set *topscores* for pruning the top-k. Entries from *smap* are inserted into *topscores* ordered by their scores, and *topscores* is maintained so that its size never exceeds k (Lines 25–33).

Our companion technical report [3] presents our model, algorithm and related work in more detail. We prove that the time complexity of our algorithm for top-k weighted partial matching based on interval trees is $O(M \log N + S \log k)$ and space complexity is $O(M\ N + k)$ where N is the number of filters, M is the number of attributes in a message and S is the number of matching constraints.

```
 1: mIndex ← new hashmap
 2: upon RECEIVE(φ) do
 3:   fid ← new filter id
 4:   for (a_i ∈ [v_i, v'_i] : w_i) ∈ φ do
 5:     tree_i ← HMAP-GET(mIndex, a_i)
 6:     if tree_i ≠ ⊥ then
 7:       INSERT(tree_i, [v_i, v'_i], w_i, fid)
 8:     else
 9:       tree_i ←new interval tree
10:       INSERT(tree_i, [v_i, v'_i], w_i, fid)
11:       HMAP-PUT(mIndex, a_i, tree_i)

12: upon RECEIVE(MSG) do
13:   smap ← new hashmap
14:   topscores ← new treeset
15:   min ← 0
16:   for a_i : [v_i, v'_i] : w_i ∈ MSG do
17:     tree_i ← HMAP-GET(master, a_i)
18:     matches ← INTERSECT(tree_i, [v_i, v'_i])
19:     if w_j = 0 ∀ a_j ∈ MSG then
20:       for all ⟨fid, [v_r, v'_r], w_r⟩ ∈ matches do
21:         score ← HMAP-GET(smap, fid)
22:         HMAP-PUT(smap, fid, score + w_r)
23:     else
24:       HMAP-PUT(smap, fid, score + w_i)
25:   for all fid ∈ GET-ALL-KEYS(smap) do
26:     if SIZEOF(topscores) < k then
27:       TREESET-ADD(topscores, fid, w)
28:       if w < min ∨ min = 0 then
29:         min ← w
30:     else if min < w then
31:       TREESET-REMOVE-MIN(topscores)
32:       TREESET-ADD(topscores, fid, w)
33:       min ← TREESET-FIND-MIN(topscores)
34:   for all fid ∈ GET-KEYS(topscores) do
35:     SEND(MSG) to RECEIVER-OF(fid)
```

Fig. 1. Algorithm for weighted partial matching of messages to filters

[3] Purdue Computer Science TR # 12-009. See http://docs.lib.purdue.edu/cstech/