Location-Aware Publish/Subscribe

Guoliang Li[†], Yang Wang[‡], Ting Wang[†], Jianhua Feng[†]
[†]Department of Computer Science, Tsinghua University, Beijing, China
[‡]National Computer Network Emergency Response Technical Team, Coordination Center of China
{liguoliang,fengjh}@tsinghua.edu.cn, aaron@ncic.ac.cn, wangting421@gmail.com

ABSTRACT

Location-based services have become widely available on mobile devices. Existing methods employ a pull model or user-initiated model, where a user issues a query to a server which replies with location-aware answers. To provide users with instant replies, a push model or server-initiated model is becoming an inevitable computing model in the next-generation location-based services. In the push model, subscribers register spatio-textual subscriptions to capture their interests, and publishers post spatio-textual messages. This calls for a high-performance location-aware publish/subscribe system to deliver publishers' messages to relevant subscribers.

In this paper, we address the research challenges that arise in designing a location-aware publish/subscribe system. We propose an R-tree based index structure by integrating textual descriptions into R-tree nodes. We devise efficient filtering algorithms and develop effective pruning techniques to improve filtering efficiency. Experimental results show that our method achieves high performance. For example, our method can filter 500 tweets in a second for 10 million registered subscriptions on a commodity computer.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases

Keywords

Location-aware Publish/Subscribe, Filtering Algorithm

1. INTRODUCTION

Location-based services (LBS), thanks to global positioning systems (GPS) wired into smart phones, have recently attracted significant attention from both industrial and academic communities. Many LBS services such as Foursquare (http://foursquare.com) have been widely accepted since they can provide users with location-aware experiences.

Existing LBS systems employ a pull model or user-initiated model [13, 7], where a user issues a query to a server which responds with location-aware answers. For example, if a mobile user wants to find a seafood restaurant nearby, she can issue a query with keywords "`seafood restaurant`" to an LBS system, which returns relevant answers based on the user's location and keywords.

To provide users with instant replies, a push model or server-initiated model is becoming an inevitable computing model in next-generation location-based services. In the push model, subscribers register spatio-textual subscriptions to capture their interests, and publishers post spatio-textual messages. This calls for a high-performance location-aware publish/subscribe system to deliver messages to relevant subscribers. This new computing model brings new user experiences to mobile users, and can help users retrieve information without explicitly issuing a query.

There are many real-world applications using location-aware publish/subscribe services. The first one is Groupon. In a Groupon system, subscribers are Groupon customers and messages are Groupon messages. Groupon customers register their interests with locations and keywords (e.g., "`iphone4s`" at New York). For each Groupon message (e.g., "`iphone4s AT&T package`" at `Manhattan`), the system provider sends the message to the customers who may be potentially interested in the message by evaluating the spatial proximity and textual relevancy between subscriptions and the message. The second one is location-aware AdSense, which extends traditional AdSense (http://www.google.com/adsense) to support location-aware services, where the subscribers are advertisers and the publishers are mobile users. The advertisers register their location-based advertisements (e.g., "`seafood`" at `Manhattan`) in the system. The system pushes relevant advertisements to mobile users based on their locations and contents they are browsing (e.g., webpages). The third one is tweet delivery. Market analysts want to receive feedback of their products in a specific area from Twitter. In this case, the subscribers are market analysts and the messages are tweets. Market analysts register their interests (e.g., "`ipad2`" at `LA`). For each tweet (e.g., "`ipad2 is expensive`" at `LA Airport`), the system pushes the tweet to relevant analysts whose spatio-textual subscriptions match the tweet.

One big challenge in a publish/subscribe system is the high performance. A publish/subscribe system should support tens of millions of subscribers and deliver messages to relevant subscribers in milliseconds. Since messages and subscriptions contain both location information and textual description, it is rather costly to deliver messages to relevant subscribers. This calls for an efficient filtering technique to support location-aware publish/subscribe services.

KDD'13, August 11–14, 2013, Chicago, Illinois, USA.

To address the challenge, we propose a token-based R-tree index structure (called R^t-tree) by integrating each R-tree node with a set of tokens selected from subscriptions. Using the R^t-tree, we develop a filter-and-verification framework to efficiently deliver a message. To reduce the number of tokens associated with R^t-tree nodes, we select some *high-quality representative tokens* from subscriptions and associate them with R^t-tree nodes. This technique not only reduces index sizes but also improves the performance. We propose efficient filtering algorithms and develop pruning techniques to achieve high performance. Experiments on large, real data sets show that our method achieves high performance.

To summarize, we make the following contributions. (1) We introduce a new computing model for LBS and formalize the location-aware publish/subscribe problem. (2) We propose a novel index structure, the R^t-tree, by integrating high-quality representative tokens selected from subscriptions into the R-tree nodes. (3) Using our proposed indexes, we develop efficient filtering algorithms and develop several effective pruning techniques to improve the efficiency.

The rest of this paper is organized as follows. We formalize the problem in Section 2. In Section 3, we propose the R^t-tree index. We propose a representative token based method in Section 4. We devise an efficient filtering algorithm without a verification step in Section 5. Experiments are provided in Section 6. We review related works in Section 7 and conclude the paper in Section 8.

2. PRELIMINARIES

2.1 Problem Formulation

In a location-aware publish/subscribe system, subscribers register subscriptions to capture their interests. A subscription s includes a textual description $s.T$ and spatial information $s.R$, denoted by $s = (T, R)$. The spatial information is used to capture a subscriber's most interested region. In this paper we use the well-known minimum bounding rectangle (MBR) to denote a region $s.R$. The textual description is used to capture a subscriber's content-based interests, denoted by a set of tokens $s.T = \{t_1, t_2, \cdots, t_{|s.T|}\}$.

A message m posted by a publisher also contains a textual description $m.T$ and spatial information $m.R$, denoted by $m = (T, R)$, which respectively have the same meaning as those of subscriptions. Note that the spatial information $m.R$ of a message can be a point, e.g., mobile user's location. If the spatial information of a message is a point, we call it *point message*; otherwise we call it *range message*.

Let $\mathcal{S} = \{s_1, s_2, \cdots, s_{|\mathcal{S}|}\}$ denote the set of all subscriptions. Given a subscription $s_i \in \mathcal{S}$ and a message m, a location-aware publish/subscribe system delivers the message m to s_i (s_i is called an answer of m), if they satisfy

(1) **Spatial Constraint**: Message m and subscription s_i have spatial overlap (i.e., $s_i.R \cap m.R \neq \phi$) and;

(2) **Textual Constraint**: All tokens in subscription s_i are contained in message m (i.e., $s_i.T \subseteq m.T$).

In this paper, for textual constraint we consider the conjunctive semantics, that is any token in a subscription needs to be contained in the message. Our method can also support disjunctive semantics by decomposing a subscription into several small subscriptions. For example, we can decompose a subscription with tokens "(`iphone4s` or `ipad2`) and `AT&T`" to two subscriptions with tokens "`iphone4s` and `AT&T`" and "`ipad2` and `AT&T`". For the spatial constrain-

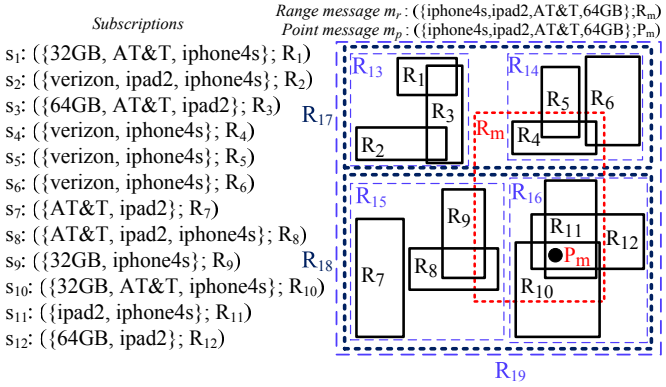


Figure 1: An example of subscriptions and messages

t, we consider the case that a message and a subscription have spatial overlap. In the future work, we want to study (1) range queries (e.g., price between 100 and 200); and (2) ranking queries: finding the subscriptions with similarity to the message larger than a given threshold (or top-k subscriptions), by considering both textual relevancy and spatial proximity. Based on these notations, we formalize the location-aware publish/subscribe problem as below.

DEFINITION 1 (LOCATION-AWARE PUBLISH/SUBSCRIBE). *Given a set of subscriptions $\mathcal{S} = \{s_1, s_2, \cdots, s_{|\mathcal{S}|}\}$ and a message m, a location-aware publish/subscribe system delivers m to $s_i \in \mathcal{S}$ if $s_i.R \cap m.R \neq \phi$ and $s_i.T \subseteq m.T$.*

EXAMPLE 1. Consider the 12 subscriptions and 2 messages in Figure 1. For point message m_p = ({`iphone4s`, `ipad2`, `AT&T`, `64GB`}, P_m), subscription s_{12} = ({`64GB`, `ipad2`}, R_{12}) is an answer. s_{10} = ({`32GB`, `AT&T`, `iphone4s`}, R_{10}) is not an answer as it has a token "`32GB`" which does not appear in m_p. s_7 = ({`AT&T`, `ipad2`}, R_7) is not an answer as it has no spatial overlap with m_p. The answers of m_p are s_{11} and s_{12}. For a range message m_r = ({`iphone4s`, `ipad2`, `AT&T`, `64GB`}, R_m), its answers are s_8, s_{11}, and s_{12}.

2.2 Straightforward Methods

Keyword-first method: It first uses existing content-based publish/subscribe techniques to generate the candidates that satisfy the textual constraint, e.g, using inverted lists [25]. Then it verifies the candidates to check whether they satisfy the spatial constraint. Obviously this method generates large numbers of candidates and leads to low performance.

Spatial-first method: Different from the keyword-first method, it first generates the location-based candidates that satisfy the spatial constraint, using existing methods, e.g., segment tree or R-tree [22]. Then it filters candidates which do not satisfy the textual constraint. This method also generates huge numbers of candidates and has poor performance.

Spatial keyword search based method: There are several studies on spatial keyword search by using a pull model [30, 13, 7, 3]. In this model, the underlying data are a set of objects with locations and keywords. A user submits a spatial keyword query, and the system returns top-k relevant objects by considering spatial and textual proximity between the query and objects. They incorporate keywords (e.g., signature files [13] or inverted lists [7]) into R-tree nodes. We can extend these approaches to support our application by traversing tree based indexes and pruning tree nodes using textual and spatial information. Experimental results (Section 6) show that our method outperforms these approaches.

3. R^t-tree BASED METHOD

We first propose an index structure in Section 3.1 and then devise efficient algorithms in Section 3.2.

3.1 R^t-tree Index Structure

R-tree is a well-known index structure to index spatial data, which is a balanced search tree where all leaf nodes are at the same level. As the standard R-tree has no textual pruning power, we propose a token-based R-tree, called R^t-tree, by integrating tokens of subscriptions into R-tree nodes.

Like R-tree, R^t-tree is also a balanced search tree. Each leaf node contains between b and B data entries, where each entry is a subscription. Each internal node has between b and B node entries. Each entry is a triple ⟨Child, MBR, TokenSet⟩, where Child is a pointer to its child node, MBR is the minimum bounding rectangle of all entries within this child, and TokenSet is a set of tokens selected from subscriptions (or a pointer to the token set). A leaf node's token set is the union of tokens of all subscriptions within this node and an internal node's token set is the union of token sets of all entries within this node. As an entry corresponds to a node, for simplicity a node is mentioned interchangeably with its corresponding entry if the context is clear.

EXAMPLE 2. Figure 2 shows an R^t-tree for the subscriptions in Figure 1. Nodes n_4, n_5, n_6, n_7 are leaf nodes. Each leaf node contains three subscriptions. For instance, n_4 contains three subscriptions s_1, s_2, s_3. Node n_2 has two entries (n_4, R_{13}, {iphone4s, ipad2, AT&T, verizon, 32GB, 64GB}) and (n_5, R_{14}, {iphone4s, verizon}). The first entry points to its child n_4 with MBR R_{13} and the token set is the union of the subscriptions of entries in its children, i.e., $s_1.T \cup s_2.T \cup s_3.T$. The second entry points to its child n_5 with MBR R_{14} and the token set is $s_4.T \cup s_5.T \cup s_6.T$.

Suppose the height of the R^t-tree is $\mathcal{H}$ and the average number of tokens in subscriptions is $\mathcal{S}_{avg}$. Each token of a subscription is stored at most $\mathcal{H}$ times[1]. Thus the token-set complexity is $\mathcal{O}(\mathcal{H} \times \mathcal{S}_{avg} \times |\mathcal{S}|)$. There are at most $\frac{|\mathcal{S}|}{b}+\frac{|\mathcal{S}|}{b^2}+\cdots+\frac{|\mathcal{S}|}{b^{\mathcal{H}}}=|\mathcal{S}|\times\frac{\frac{1}{b}-\frac{1}{b^{\mathcal{H}+1}}}{1-\frac{1}{b}}\approx\frac{1}{b-1}\times|\mathcal{S}|$ nodes (Suppose the root also has b child nodes). The space complexity of a node is $\mathcal{O}(B)$ for storing MBRs and child pointers. Thus the overall space complexity is

$$\mathcal{O}\big(\frac{B}{b-1}\times|\mathcal{S}|+\mathcal{H}\times\mathcal{S}_{avg}\times|\mathcal{S}|\big).$$

3.2 Filtering Algorithms

We discuss how to use the R^t-tree to filter a message. We want to prune unnecessary nodes and only visit a small number of "pivotal" nodes, where a node is a *pivotal node* if there exist answers in its leaf descendants. Thus when traversing the R^t-tree, we only need to visit the *pivotal nodes*. However an R^t-tree node may have large numbers of leaf descendants and it is expensive to check whether a node is a pivotal node. Based on this observation, we propose a filter-and-verification framework. In the filter step, we find a set of *candidate nodes* which is a superset of pivotal nodes. In the verification step we verify the subscriptions in the *leaf candidate nodes* generated in the filter step. Next we introduce two filters.

[1]In this paper, we suppose the entries in the same internal node have no overlap, e.g., R^+-tree.

Filters: Given a node n, let $n.R$ denote its MBR and $n.T$ denote its token set which can be obtained from the corresponding entry in its parent node. We prune node n, if

(1) **MBR Filter**: $n.R \cap m.R = \phi$. It invalidates spatial constraint, as any subscription under node n (i.e., subscriptions in n's leaf descendants) has no overlap with m; or

(2) **Token Filter**: $n.T \cap m.T = \phi$. It invalidates the textual constraint. The reason is that any subscription under node n must contain a token in $n.T$ which does not appear in $m.T$, thus the subscription does not satisfy the textual constraint.

The nodes that are not pruned by the MBR filter and token filter are called *candidate nodes*. The subscriptions in the leaf candidate nodes are *candidate answers*.

For MBR-filter, to check whether $n.R \cap m.R = \phi$, we examine whether $n.R$ has a vertex contained in $m.R$. For token filter, to check whether $n.T \cap m.T = \phi$, we use the hash table of n's TokenSet to do the checking as follows. For each token in $m.T$, if it is contained in the hash table, $n.T \cap m.T \neq \phi$; otherwise we check the next token in $m.T$.

Verification: For each subscription s on leaf candidate nodes, we check whether $s.R \cap m.R \neq \phi$ and $s.T \subseteq m.T$. If yes, s is an answer. To check $s.T \subseteq m.T$, we first sort the tokens in $m.T$ (e.g., document frequency order, and we will discuss different sorting strategies in Section 4.2). Then we use each token in $s.T$ to do a binary search in $m.T$.

Algorithm: Given a message m, we traverse the R^t-tree in pre-order. From the root node, we scan each of its entries, e.g., node n. If node n satisfies one of the two filters, i.e., $n.R \cap m.R = \phi$ or $n.T \cap m.T = \phi$, we prune node n; otherwise we visit n's children and repeat the above steps. Iteratively, we can find all leaf candidate nodes.

EXAMPLE 3. Consider the R^t-tree in Figure 2 and a message $m = (\{\texttt{ipad2}, \texttt{AT\&T}, \texttt{32GB}, \texttt{64GB}\}, R_m)$, where R_m is the dashed MBR in Figure 1. The root node has two entries: node n_2 and node n_3. Node n_2 is not pruned by the MBR filter as it has overlap with $m.R$. n_2 is not pruned by the token filter as it contains "ipad2" which appears in message m. Thus node n_2 is a candidate node. Node n_2 has two entries: n_4 and n_5. As node n_4 has no overlap with $m.R$, it is pruned by the MBR filter. Node n_5 has spatial overlap with $m.R$ and cannot be pruned by the MBR filter. As n_5's token set {iphone4s, verizon} has no common token with $m.T$, n_5 is pruned by the token filter. For node n_3, we access its first entry n_6. As n_6 is not pruned by the MBR filter and token filter, it is a leaf candidate node. We verify subscriptions s_7, s_8, s_9 in n_6. We prune s_7 as it has no spatial overlap with $m.R$. We prune s_8 and s_9 as they have a token "iphone4s" which does not appear in $m.T$. Similarly n_7 is also a leaf candidate node and we verify s_{10}, s_{11}, s_{12} in n_7. We prune s_{10} and s_{11} as their tokens are not contained in $m.T$ and get an answer s_{12}.

Notice that an algorithm should satisfy (1) Completeness: any subscription satisfying the spatial constraint and textual constraint must be found by the algorithm; and (2) Correctness: any subscription found by the algorithm must satisfy the two constraints. We prove that the R^t-tree based algorithm satisfies the two properties as stated in Theorem 1.

THEOREM 1. *The R^t-tree based algorithm satisfies completeness and correctness.*

PROOF. We omit the proofs due to space constraints. □

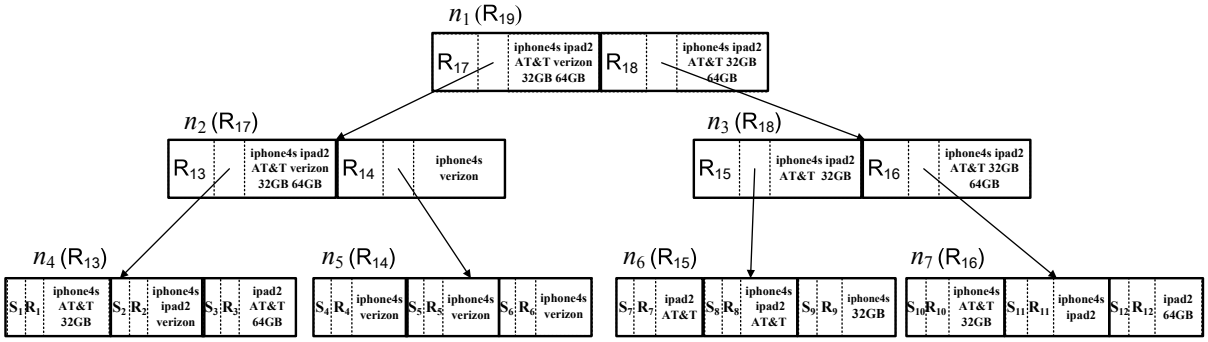


Figure 2: R^t-tree index for subscriptions in Figure 1

4. SELECTING REPRESENTATIVE TOKENS TO IMPROVE THE PERFORMANCE

In this section, we propose an effective technique to reduce the number of tokens associated with each node, which not only reduces index sizes but also improves performance.

4.1 Representative Tokens

Different from TokenSets on R^t-tree nodes, we select high-quality *representative tokens* and use *representative-token sets* to replace TokenSets. For each subscription, we select a single token as its *representative token.* For each leaf node, its representative-token set is the set of representative tokens of subscriptions within this node. For each internal node, its representative-token set is the union of representative-token sets of its child nodes. To differentiate this method from R^t-tree, we call it R^{t+}-tree.

We have an observation that if the representative-token set of a node has no common token with a message, we can prune the node as stated in Lemma 1. This is because any subscription under this node cannot satisfy textual constraint as it contains tokens which are not in the message.

LEMMA 1. *Given an R^{t+}-tree node n and a message m, if n's representative-token set has no common token with $m.T$, any subscription under node n cannot be an answer of m.*

EXAMPLE 4. Consider the subscriptions in Figure 1. We construct an R^{t+}-tree as shown in Figure 3(a). For subscriptions s_1, s_2, s_3, we respectively select representative tokens "iphone4s", "iphone4s", "ipad2". Thus the representative-token set of node n_4 is {iphone4s, ipad2}. For s_4, s_5, s_6, we select representative tokens "iphone4s". Thus the representative-token set of node n_5 is {iphone4s}. The representative-token set of n_2 is {iphone4s, ipad2} which is the union of the representative tokens of its child nodes n_4 and n_5. For a message m=({AT&T, 32GB, 64GB}, R_m), R^t-tree cannot prune node n_2 as its token set shares a token "AT&T" with m(Figure 2). Instead R^{t+}-tree can prune n_2 as its representative-token set shares no common token with m.

There are at most $|\mathcal{S}|$ representative tokens and each representative token is stored at most $\mathcal{H}$ times. Thus the complexity of representative-token sets is $\mathcal{O}(\mathcal{H} \times |\mathcal{S}|)$. The R^{t+}-tree has the same MBR size with the R^t-tree. Thus the space complexity of the R^{t+}-tree is

$$\mathcal{O}\big(\frac{B}{b-1} \times |\mathcal{S}| + \mathcal{H} \times |\mathcal{S}|\big).$$

We still adopt the filter-and-verification framework in Section 3.2 to filter a message. Theorem 2 shows correctness and completeness of the R^{t+}-tree based algorithm.

THEOREM 2. *The R^{t+}-tree based algorithm satisfies completeness and correctness.*

Comparison of R^t-tree and R^{t+}-tree: Based on the space complexity, the R^{t+}-tree involves much smaller index sizes than the R^t-tree. More interestingly, we prove that the R^{t+}-tree has larger pruning power than the R^t-tree. That is if the R^t-tree prunes a node, then the R^{t+}-tree must also prune the node as stated in Lemma 2.

LEMMA 2. *Let $\mathcal{C}_{R^t}$ and $\mathcal{C}_{R^{t+}}$ respectively denote the candidate node sets of the R^t-tree based method and the R^{t+}-tree based method, we have $\mathcal{C}_{R^t} \supseteq \mathcal{C}_{R^{t+}}$.*

EXAMPLE 5. Recall the R^t-tree in Figure 2. Consider a message m=({AT&T, 32GB, 64GB}, R_m). Node n_3 shares common tokens with m, thus the R^t-tree cannot prune node n_3. However all subscriptions under n_3 contain tokens "iphone4s" or "ipad2" which do not appear in m. Thus n_3 can be pruned. In the R^{t+}-tree, the representative-token set of n_3 is {iphone4s, ipad2} which has no common token with m, thus the R^{t+}-tree can prune node n_3.

4.2 Selecting Representative Tokens

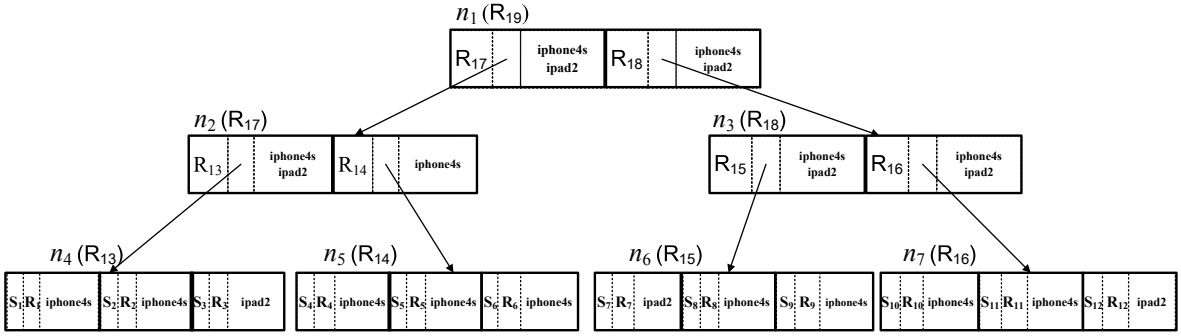
Given a subscription, there are multiple ways to select a representative token from this subscription. We introduce several methods to select high-quality representative tokens.

Random selection: A naive method is to randomly select a representative token for each subscription. This method does not utilize the token distribution and can be optimized.

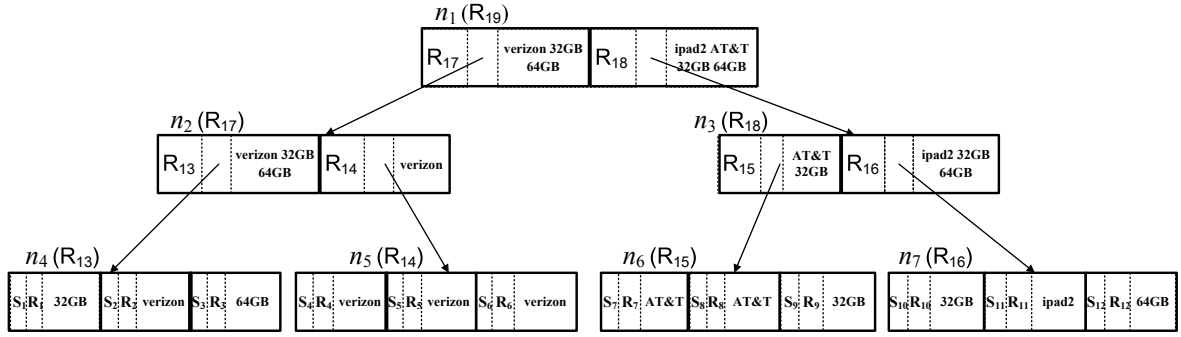
The df-based method: Intuitively the smaller sizes of representative-token sets, the larger pruning power, and the smaller number of candidates. Thus it is important to reduce the sizes of the representative-token sets. To this end, we select representative tokens based on the df order as follows. We first select the token with the largest df and take it as the representative token of subscriptions that contain the token. Then we remove all such subscriptions and select the token with the largest df in the remainder subscriptions. Iteratively we generate representative token.

The idf-based method: A token with higher frequency usually has larger probability appearing in a message. If we add such a token into the token set, the token set has larger probability sharing tokens with the message, thus the token set has lower pruning power. As the df order selects the representative tokens with the highest frequencies, the df based method may lead to low pruning power. To address this issue, for each subscription, we select the token with the largest idf as its representative token.

EXAMPLE 6. Consider the subscriptions in Figure 1. If we use the df order, the R^{t+}-tree is shown in Figure 3(a). As "iphone4s" has the largest df (its df is 9) and it appears in $s_1, s_2, s_4, s_5, s_6, s_8, s_9, s_{10}, s_{11}$, we select "iphone4s" as their representative tokens. Then in the remainder subscriptions, i.e., s_3, s_7, s_{12}, "ipad2" has the largest df, we select "ipad2" as their representative tokens. For node n_4, the



(a) Representative tokens are selected by the df order



(b) Representative tokens are selected by the idf order

Figure 3: R^t-tree with representative tokens (df of tokens: iphone4s:9, ipad2:6, AT&T:5, verizon:4, 32GB:3, 64GB:2)

subscriptions under n_4 are s_1, s_2, s_3, thus its representative-token set is {iphone4s, ipad2}. On the R^{t+}-tree there are 23 representative tokens and on the R^t-tree there are 59 tokens. Obviously the df order can reduce token-set sizes. Figure 3(b) shows the R^{t+}-tree with representative tokens selected by the idf order. We respectively select "32GB, verizon, 64GB" for subscriptions s_1, s_2, s_3 with the largest idf. The representative-token set of node n_2 is the union of these three representative tokens. Consider message m = ({iphone4s, ipad2, AT&T, 64GB}, Rm). The df order cannot prune node n_5 as its representative-token set {iphone4s} shares a common token with message m. However the idf order can prune node n_5 as its representative-token set {verizon} shares no common token with message m.

Locality-aware method: Tokens usually have different frequencies in different locations. For example, in Figure 1 token "verizon" has a large frequency in node $n_5(R_{14})$ and low frequency in nodes $n_6(R_{15})$ and $n_7(R_{16})$. We should consider the locality-aware token distribution to count token frequencies. To address this issue, we partition the whole region into several grids and count the frequencies of tokens for each grid. Thus we can capture the locality-aware frequencies. Then for each subscription, we first sort its tokens based on by their frequencies in the grid where the subscription locates in and then use the df-based method or the idf-based method to select representative tokens.

5. SELECTING MULTIPLE REPRESENTATIVE TOKENS

In this section, we propose selecting multiple representative tokens and devise an efficient filtering algorithm, which directly finds answers and avoids the verification step.

5.1 Multiple Representative Tokens

Like the R^{t+}-tree, we also associate each node with a representative-token set. Different from the R^{t+}-tree, for subscription s, we do not repeatedly use a single token for all ancestors of its corresponding leaf node. Instead we select different representative tokens for different ancestor nodes and each ancestor node contains a token of s. If s has larger than $\mathcal{H}$ tokens, we insert its last $|s|-\mathcal{H}+1$ tokens into the leaf node. If s has smaller than $\mathcal{H}$ tokens, only the ancestors in the first $|s|$ levels contain a token. Formally, let $s.T[i]$ denote the i-th token in s and we assign $s.T[i]$ to its ancestor at the i-th level. For each node n on the i-th level, its representative-token set is the set of i-th tokens of subscriptions under node n, i.e., $\cup_{s\in\mathcal{S}_n} s.T[i]$, where $\mathcal{S}_n$ is the set of subscriptions under node n. Let $n.T$ denote the representative-token set of node n. For each token $t \in n.T$, we build an inverted list $\mathcal{I}_n[t]$ of subscriptions in $\mathcal{S}_n$ whose i-th token is t, i.e., $\mathcal{I}_n[t]$=$\{s|s\in\mathcal{S}_n$ and $s.T[i]$=$t\}$.

EXAMPLE 7. Figure 4 shows the R^{t++}-tree for subscriptions in Figure 1. For s_1, $s_1.T[1]$ = "32GB", $s_1.T[2]$ = "AT&T", $s_1.T[3]$ = "iphone4s". Consider the first entry at the first level (the root node), i.e., node n_2. Node n_2 contains six subscriptions $s_1, s_2, \cdots, s_6$, thus the subscription set under node n_2 is $\mathcal{S}_{n_2} = \{s_1, \cdots s_6\}$. $s_1.T[1]$ = "32GB", $s_2.T[1]$ = "verizon", $s_3.T[1]$ = "64GB", $s_4.T[1]$ = "verizon", $s_5.T[1]$ = "verizon", and $s_6.T[1]$ = "verizon". The representative-token set of node n_2 is $n_2.T = s_1.T[1] \cup s_2.T[1] \cup \cdots \cup s_6.T[1]$ = {verizon, 32GB, 64GB}. The inverted list of "verizon" in node n_2 is $\mathcal{I}_{n_2}$[verizon] = $\{s_2, s_4, s_5, s_6\}$. Similarly for node n_5, it contains three subscriptions, s_4, s_5, s_6. The second tokens of these subscriptions are "iphone4s". Thus the token set of n_5 is {iphone4s} and the inverted list of iphone4s in node n_5 is $\{s_4, s_5, s_6\}$.

The R^{t++}-tree has the following good property.

PROPERTY 1. *For any subscription s, s appears exactly $|s|$ times on the inverted lists in the R^{t++}-tree. For any $i < |s|$, s appears exactly i times in the first i levels.*

For example, consider subscription s_8 with tokens {AT&T, ipad2, iphone4s}. s_8 appears three times in the R^{t++}-tree, i.e., the inverted list of AT&T in node n_1 at level 1, the

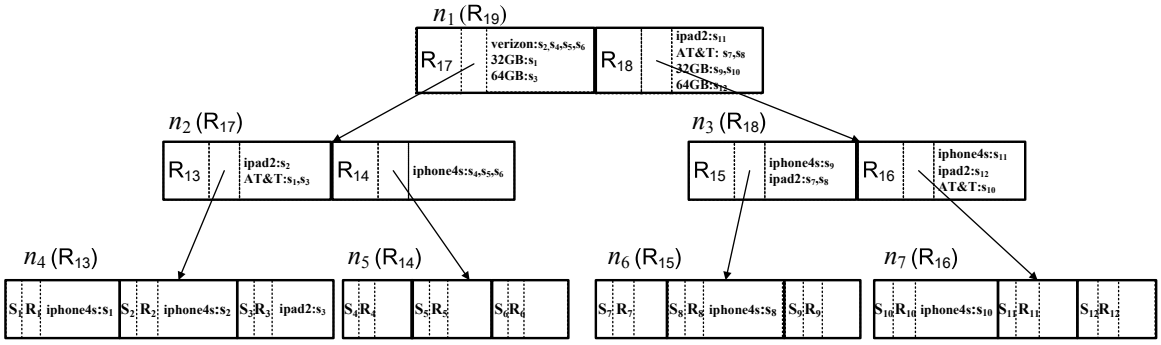


Figure 4: An R^t-tree with multiple representative tokens (iphone4s:9, ipad2:6, AT&T:5, verizon:4, 32GB:3, 64GB:2)

inverted list of ipad2 in node n_3 at level 2, and the inverted list of iphone4s in node n_6 at level 3. Obviously s_8 appears once in the first level and twice in the first two levels.

The R^{t++}-tree has the same MBR size with the R^t-tree but has much smaller token-set size. We first analyze the token-set size. For each token of a subscription, it appears in exactly one R^{t++}-tree node. Thus the total size of representative-token sets is $\mathcal{O}(|\mathcal{S}_{avg}| \times |\mathcal{S}|)$. Then we consider inverted-list size. Each subscription s appears in exactly $|s.T|$ inverted lists, and the total inverted-list size is $\mathcal{O}(|\mathcal{S}_{avg}| \times |\mathcal{S}|)$. Obviously the size of token sets and inverted lists is proportional to the data size. Plus MBR sizes, the overall space is

$$\mathcal{O}\big(\frac{B}{b-1} \times |\mathcal{S}| + \mathcal{S}_{avg} \times |\mathcal{S}|\big).$$

5.2 Filtering Algorithms

Consider a message m. We traverse the R^{t++}-tree from the root. For a node n in the i-th level, if $n.R \cap m.R = \phi$, we prune node n as the message invalidates the spatial constraint; otherwise for each token $t \in n.T \cap m.T$, we retrieve the corresponding inverted list $\mathcal{I}_n[t]$. For each subscription $s \in \mathcal{I}_n[t]$, we count its occurrence number in the first i levels (i.e., the number of inverted lists of ancestors of node n that contain the subscription), denoted by cand_s. We will discuss how to compute cand_s later. Here based on cand_s, we have the following observations.

Case 1 - $\mathsf{cand}_s < i$: In this case, subscription s appears smaller than i times in the first i levels. We can prove that s is not an answer as formalized in Lemma 3. The reason is as follows. As s appears in the i-th level, it contains at least i tokens. For each node on the path from the root to node n, s must have a token in the node. If $\mathsf{cand}_s < i$, s must have a token which does not appear in m. Thus s invalidates the textual constraint and cannot be an answer.

LEMMA 3. *Consider a message m and a node n at the i-th level. For any subscription s on the inverted lists of node n, if $\mathsf{cand}_s < i$, s cannot be an answer of m.*

For example, consider message $m = (\{\texttt{ipad2}, \texttt{64GB}\}, \texttt{R}_\texttt{m})$ and node n_3 at the second level in Figure 4. For subscription s_7, we have $\mathsf{cand}_{s_7} = 1$. We can deduce that s_7 cannot be an answer of the message. The main reason is as follows. As s_7 has 2 tokens, it must appear twice in the first two levels and its occurrence number should be two. However its occurrence number cand_{s_7} is 1, thus s_7 must contain a token which does not appear in m and s_7 is not an answer.

Case 2 - $\mathsf{cand}_s = i$: We consider the following subcases.

Case 2.1 - $\mathsf{cand}_s = i = |s|$: In this case, if $s.R \cap m.R \neq \phi$, s must be an answer of m as stated in Lemma 4. The basic idea is as follows. Each token of s appears in the R^{t++}-tree once. As $\mathsf{cand}_s = |s|$, all tokens in s are contained in m, thus s satisfies the textual constraint. As $s.R \cap m.R \neq \phi$, s satisfies the spatial constraint. Thus s must be an answer.

LEMMA 4. *Consider a message m and a node n at the i-th level. For any subscription s on the inverted lists of node n, if $\mathsf{cand}_s = |s|$ and $s.R \cap m.R \neq \phi$, s must be an answer.*

For example, consider the message $m_r = (\{\texttt{iphone4s}, \texttt{ipad2}, \texttt{AT\&T}, \texttt{64GB}\}, R_m)$ and node n_6 at the third level in Figure 4. For subscription s_8, we have $\mathsf{cand}_{s_8} = 3$ and $s_8 \cap m_r.R \neq \phi$. We can deduce that s_8 must be an answer of m_r. The reason is as follows. First s_8 has 3 tokens and $\mathsf{cand}_{s_8} = 3$. That is all of its tokens must be contained in the message. Thus s_8 satisfies the token constraint. Second, as $s_8 \cap m_r.R \neq \phi$, s_8 satisfies the MBR constraint.

Case 2.2 - $\mathsf{cand}_s = i < |s|$: In this case, if n is an internal node, there may exist answers under this node, and we need to visit n's children and repeat the above steps. On the contrary, if n is a leaf node, s cannot be an answer. This is because s only has $i < |s|$ tokens that appear in message m (i.e., it has at least one token which does not appear in m).

Case 3 - $\mathsf{cand}_s > i$: In this case n must be a leaf node. This is because if n is an internal node, $\mathsf{cand}_s \leq i$ since each internal node contains at most one token of s. If $\mathsf{cand}_s = |s|$, it is similar to Case 2.1; If $\mathsf{cand}_s < |s|$, it is similar to Case 2.2.

Computing cand_s: To efficiently compute the occurrence number cand_s, we use a hash map $\mathcal{M}$ to maintain the occurrence numbers. Given a message m and a node n, if $n.R \cap m.R \neq \phi$, we use the hash-based method (Section 3.2) to compute $m.T \cap n.T$. For each token $t \in m.T \cap n.T$, we access its inverted list $\mathcal{I}_n[t]$. For each subscription s in $\mathcal{I}_n[t]$, we increase $\mathcal{M}[s]$ by 1. Notice that as s may be in multiple inverted lists on a leaf node, when computing its occurrence number, we consider all such lists. Obviously $\mathsf{cand}_s = \mathcal{M}[s]$.

The R^{t++}-tree based Algorithm: Based on the above analysis, we devise an efficient algorithm. We still traverse the R^{t++}-tree from the root in pre-order. Given a node n, for each token $t \in n.T \cap m.T$, we retrieve the corresponding inverted list $\mathcal{I}_n[t]$. For each subscription $s \in \mathcal{I}_n[t]$, we count its occurrence number cand_s. If $\mathsf{cand}_s = |s|$ and $s.R \cap m.R \neq \phi$, s is an answer and added into the result set. If there exists a subscription s such that $\mathsf{cand}_s = i < |s|$ and n is not a leaf node, there may exist answers under node n. We access the node and repeat the above steps; otherwise if there has no such subscription, we prune node n. The main reason is as follows. First, all subscriptions with no smaller than i tokens under node n cannot be an answer. Second, for subscriptions with smaller than i tokens, if they are answers, they are added as results when accessing n's ancestors.

Figure 5 illustrates the pseudo-code. R^{t++}-TREE first constructs an R^{t++}-tree with root r (line 1) and initializes

Algorithm 1: R^{t++}-Tree $(\mathcal{S}, m)$

Input: $\mathcal{S}$: A subscription set; m: A message
Output: $\mathcal{R}$: Answers of m

```
1 Build an R^{t++}-tree with root r;
2 Initialize a hash map M ;
3 R^{t++}-Tree-Prune (r, m, R, M) ;
```

Function R^{t++}-Tree-Prune(r, m, $\mathcal{R}$, $\mathcal{M}$)

Input: r: An R^{t++}-tree node; m: A message
$\mathcal{R}$: Answers of m; $\mathcal{M}$: Hash map
Output: $\mathcal{R}$: Answers of m

```
1  visitFlag = false;
2  for each entry n in node r do
3      if n.R ∩ m.R = φ then return;
4      for token t ∈ n.T ∩ m.T do
5          for each subscription s in I_n[t] do
6              M[s] = M[s] + 1;
7              if !visitFlag & M[s] = i < |s| & n is not a
               leaf node then
8                  visitFlag = true;
9              if M[s] = |s| & s.R ∩ m.R ≠ φ then
10                 R ← s ;
11     if visitFlag then
12         R^{t++}-Tree-Prune (n, m, R, M);
```

Figure 5: R^{t++}-Tree based algorithm

a hash map $\mathcal{M}$ (line 2). Then it calls function R^{t++}-Tree-Prune to filter message m. R^{t++}-Tree-Prune first scans each entry (node n) from the root. If n does not satisfy spatial constraint (line 3), R^{t++}-Tree-Prune prunes the node (line 3); otherwise R^{t++}-Tree-Prune computes the intersection of its representative-token set and $m.T$ (line 4). Then for each token t in the intersection, R^{t++}-Tree-Prune accesses its inverted list $\mathcal{I}_n[t]$ and for each subscription s on $\mathcal{I}_n[t]$, it increases $\mathcal{M}[s]$ by 1 to count its occurrence number (line 6). If $\mathcal{M}[s] = i < |s|$ and n is not a leaf node, R^{t++}-Tree-Prune visits n's children (To avoid repeatedly visiting n's children, we first set visitFlag as true in line 8 and then if visitFlag is true, we visit such children in line 12). If $\mathcal{M}[s] = |s|$ and $s.R \cap m.R \neq \phi$, s is an answer and added into the result set (line 10).

Example 8. Consider the R^{t++}-tree in Figure 4 and the message $m_r = (\{\texttt{iphone4s}, \texttt{ipad2}, \texttt{AT\&T}, \texttt{64GB}\}, \texttt{R}_\texttt{m})$ in Figure 1. We first access the root. The first entry, i.e., node n_2, satisfies the spatial constraint. Thus we compute the intersection of the representative-token set of node n_2 and $m_r.T$. Here we get $\{\texttt{64GB}\}$ and the corresponding inverted list $\mathcal{I}_{n_2}[\texttt{64GB}] = \{\texttt{s}_3\}$. Thus $\texttt{cand}_{s_3} = 1$ and s_3 is a candidate. Next we access node n_2. We prune its first entry, node n_4, based on the spatial constraint. For its second entry, n_5, we get the intersection of the representative-token set of node n_5 and $m_r.T$, i.e., $\{\texttt{iphone4s}\}$. For each subscription in its inverted list $\mathcal{I}_{n_5}[\texttt{iphone4s}] = \{\texttt{s}_4, \texttt{s}_5, \texttt{s}_6\}$, as their occurrence number is 1, which is smaller than the node level (i.e., 2). Thus the three subscriptions are not candidates. As there is no candidate, we prune node n_5.

Theorem 3. *The R^{t++}-tree based algorithm satisfies completeness and correctness.*

6. EXPERIMENTAL STUDY

In this section we report experimental results. We compared with state-of-the-art method IRTree [7]. We extended IRTree to support our problem as discussed in Section 2.2.

We used two datasets. The first one was a real dataset Twitter. We collected 60 million tweets from May 2012 to August 2012, in which 13 million tweets had locations. We selected 10 million tweets with region information (denoted by polygon in the twitter dataset) as subscriptions and used the others as messages. Each subscription contained an MBR and had 1-5 tokens selected from the tweets. The average token number of subscriptions was 3. The token distribution follows a Zipf's law.

Table 1: Dataset statistics.

	Twitter	USA
Subscription number	10 million	10 million
Subscription length	1-5	1-5
Avg Subscription length	3	3
Subscription size	0.54 GB	0.65 GB
Token distribution	Zipf	Uniform
R^t-tree size	1.63 GB	1.76 GB
R^{t+}-tree size	0.79 GB	0.85 GB
R^{t++}-tree size	0.89 GB	0.92 GB

We generated four groups of messages and each group contained 10,000 messages.

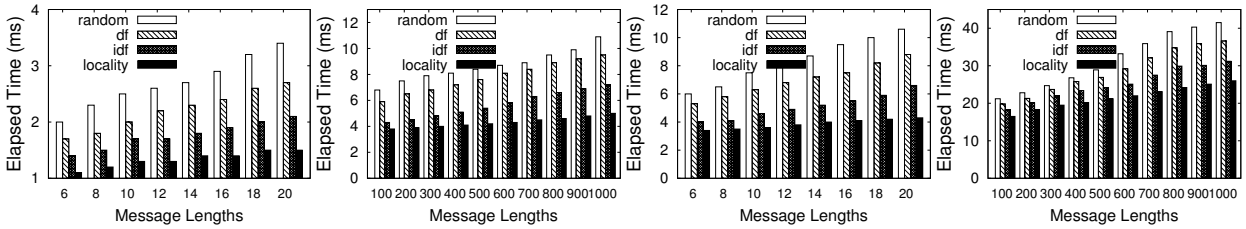
(1) Short Point Messages: Each message contained 6-20 tokens and had a point location. **(2) Long Point Messages:** Each message contained 100-1000 tokens and had a point location. **(3) Short Range Messages:** Each message contained 6-20 tokens and had an MBR region. **(4) Long Range Messages:** Each message contained 100-1000 tokens and had an MBR region.

We also used a synthetic dataset by combing Point of Interests (POIs) in USA and publications in DBLP. The USA dataset contained 17 million POIs and DBLP had 1.5 million publications. We generated MBRs from the POIs by selecting a POI as the center and extending a random width and height. Each subscription was generated by selecting an MBR and 1-5 tokens from DBLP. Each message was generated by selecting an MBR and a publication. We also generated four groups of messages. Table 1 summarized datasets (the subscription length is the number of tokens).

All the algorithms were implemented in C++. All the experiments were run on a Windows Server 2008 machine with an Intel Core E5410 2.33GHz CPU and 16 GB memory. In the experiments we set $b = 25$ and $B = 50$.

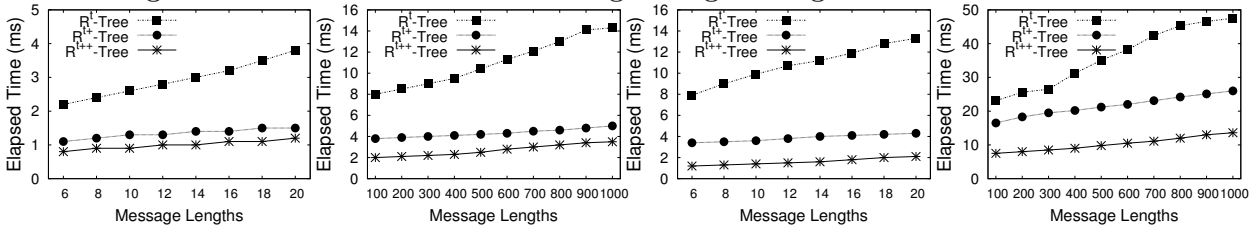
6.1 Evaluating Different Sorting Strategies

We evaluated different sorting strategies: random, df, idf, and locality-aware (We generated 10,000 grids and used the idf order) on the R^{t+}-tree as discussed in Section 4.2. The R^{t+}-tree sizes for the four methods were respectively 0.76 GB, 0.72 GB, 0.79 GB, and 0.83 GB. This is because the df order shared many tokens in upper-level nodes and the idf order and the locality-aware method shared few tokens. Figure 6 shows the results. We can see that idf outperformed df which in turns was better than random. This is because idf can prune many unnecessary nodes as infrequent tokens were on the upper-level nodes which had low probability to be contained in messages. df reduced token-set sizes by sharing many common tokens, thus df outperformed random. The locality-aware method was better than df and idf as it used the locality-aware token distributions.



(a) Short Point Messages (b) Long Point Messages (c) Short Range Messages (d) Long Range Messages

Figure 6: Evaluation on different sorting strategies using R^{t+}-tree on Twitter dataset



(a) Short Point Messages (b) Long Point Messages (c) Short Range Messages (d) Long Range Messages

Figure 7: Evaluation on R^t-tree by using different token sets on Twitter dataset

6.2 Evaluating R^t-tree with Different Token Sets

We evaluated R^t-tree with different token sets, i.e., R^t-tree with token sets, R^{t+}-tree with representative tokens, R^{t++}-tree with multiple representative tokens. For R^{t+}-tree and R^{t++}-tree, we used the locality-aware method. Figure 7 shows the results. We can see that R^{t++}-tree outperformed R^{t+}-tree which was better than R^t-tree. This is because R^{t++}-tree had larger pruning power and did not involve an expensive verification step. R^{t+}-tree reduced the token-set sizes and decreased the number of candidates against R^t-tree, thus it achieved higher performance than R^t-tree. For example, in Figure 7(d), for messages with 1000 tokens, R^t-tree took 50 milliseconds, and R^{t+}-tree decreased the time to 28 milliseconds, and R^{t++}-tree further reduced the time to 12 milliseconds. For short messages, e.g., tweets, in Figures 7(a) and 7(c), R^{t++}-tree only took 1-2 milliseconds.

6.3 Comparison with Existing Methods

We compared our best method R^{t++}-tree with existing approaches, the keyword-first method [25], the spatial-first method [22], and state-of-the-art spatial keyword search method IRTree [7] as discussed in Section 2.2. All algorithms employed an in-memory setting. Figures 8 and 9 show the results on the Twitter and USA datasets respectively.

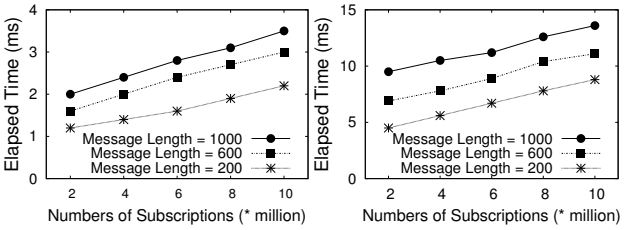
An observation is that the spatial-first method outperformed the keyword-first method for point messages (Figures 8(a), 8(b), 9(a), 9(b)). The reason is that the spatial-first method efficiently found candidate nodes using spatial index structures while the keyword-first method had no spatial pruning power. Another observation is that the spatial-first method had lower performance than the keyword-first method for range messages (Figures 8(c), 8(d), 9(c), 9(d))), as the spatial-first method had no textual pruning power.

In addition, notice that IRTree also achieved low performance and was even worse than the spatial-first method. There are two main reasons. First, it associated each R-tree node with a rather large inverted index and it was very expensive to traverse the R-tree by using the large inverted index. Second, it was designed for spatial keyword search and had to access larger numbers of unnecessary nodes. Thus IRTree was inefficient for the filtering problem.

Our R^{t++}-tree based algorithm always achieved the highest performance for any types of messages. This is because R^{t++}-tree seamlessly integrated the spatial and textual information and had large pruning power.

6.4 Scalability

We evaluated the scalability of the R^{t++}-tree based algorithm by varying the numbers of subscriptions. Figure 10 shows the results. We can see that our method scaled very well, and with the increase of the numbers of subscriptions, the elapsed time increased sublinearly. This is because even if the number of subscriptions increased, our indexes still pruned large numbers of unnecessary subscriptions.



(a) Long Point Messages (b) Long Range Messages

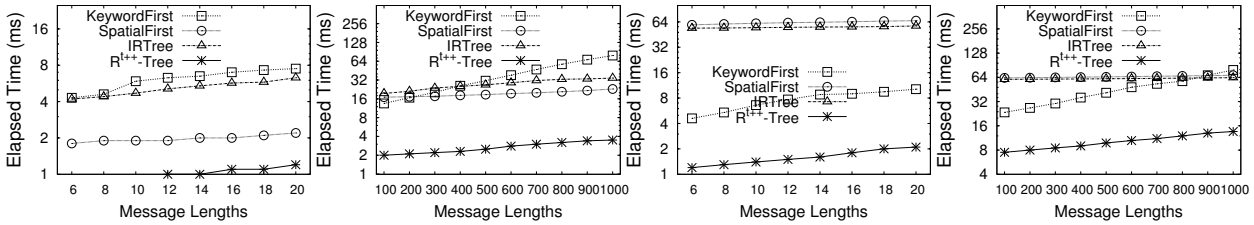
Figure 10: Scalability of R^{t++}-tree on Twitter dataset

7. RELATED WORK

Although there are some studies on location-aware publish/subscribe systems from a network perspective (e.g., routing messages) [5, 8, 10], to our knowledge there is no study on this problem focusing on performance and scalability.

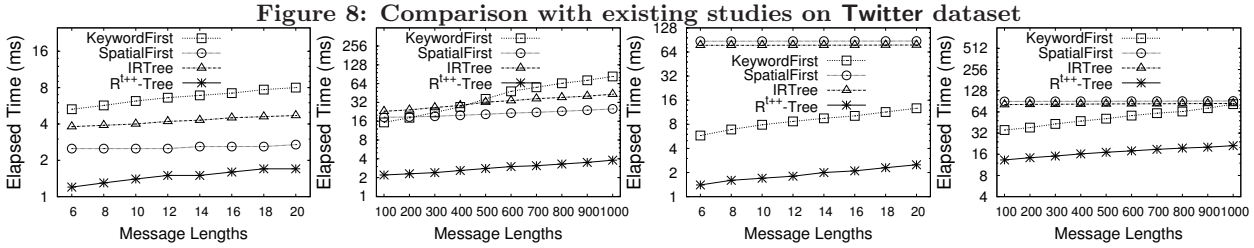
Spatial Keyword Search: Recently there are many studies on spatial keyword search [30, 6, 15, 13, 27, 7, 28, 26, 3, 23, 4, 21, 20, 17, 18, 29, 16, 12, 19]. The first problem is knn based keyword search, which, given a location and a set of keywords, finds top-k nearest neighbors by considering the distance and textual relevancy. Felipe et al. [13] integrated signature files and R-tree. Cong et al. [7] combined inverted files and R-tree. The second problem is region based keyword search, which, given a region and a keyword query, finds the relevant objects in this region. Zhou et al. [30] discussed several strategies to combine R-tree and inverted indexes. Hariharan et al. [15] integrated inverted lists into R-tree nodes. The third problem is collective keyword search, which, given a set of keywords, finds a set of close objects that match the keywords. Zhang et al. [27, 28] integrated keyword bitmap and MBR into R-tree nodes to find the closest objects.

Obviously the above problems substantially differ from our location-aware publish/subscribe problem, since they use a pull model and we employ a push model.



(a) Short Point Messages (b) Long Point Messages (c) Short Range Messages (d) Long Range Messages

Figure 8: Comparison with existing studies on Twitter dataset



(a) Short Point Messages (b) Long Point Messages (c) Short Range Messages (d) Long Range Messages

Figure 9: Comparison with existing studies on USA dataset

Publish/Subscribe Services: Foltz and Dumais [14] studied the information filtering problem from the IR perspective. Fabret et al. [11] and Aguilera et al. [1] studied the publish/subscribe problem from the database perspective. Yan and Garcia-Molina studied keyword-based filtering based on a binary model [25] and a vector space model [24]. There are some studies on XML filtering [2, 9]. Existing publish/subscribe methods only consider textual descriptions while we consider both textual and spatial information.

8. CONCLUSION

We study the location-aware publish/subscribe problem. We propose an effective index structure R^t-tree by integrating textual description into R-tree nodes. We develop a filter-and-verification framework, and devise efficient filtering algorithms. We propose reducing the number of tokens associated with each node which not only reduces index sizes but improves performance. We also devise an efficient algorithm which directly finds answers and avoids the verification step. Experimental results on real data sets show that our method achieves high performance and good scalability.

Acknowledgement: This work was partly supported by the National Natural Science Foundation of China under Grant No. 61003004 and 61272090, National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302206, a Tsinghua project under Grant No. 20111081073, Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the "NExT Research Center" funded by MDA, Singapore, under Grant No. WBS:R-252-300-001-490.

9. REFERENCES

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *PODC*, pages 53–61, 1999.

[2] M. Altinel and M. J. Franklin. Efficient filtering of xml documents for selective dissemination of information. In *VLDB*, pages 53–64, 2000.

[3] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB*, 3(1):373–384, 2010.

[4] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD Conference*, pages 373–384, 2011.

[5] X. Chen, Y. Chen, and F. Rao. An efficient spatial publish/subscribe system for intelligent location-based services. In *DEBS*, 2003.

[6] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD Conference*, pages 277–288, 2006.

[7] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2009.

[8] G. Cugola and J. E. M. de Cote. On introducing location awareness in publish-subscribe middleware. In *ICDCS Workshops*, pages 377–382, 2005.

[9] Y. Diao and M. J. Franklin. Query processing for high-volume xml message brokering. In *VLDB*, pages 261–272, 2003.

[10] P. T. Eugster, B. Garbinato, and A. Holzer. Location-based publish/subscribe. In *NCA*, pages 279–282, 2005.

[11] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In *SIGMOD Conference*, pages 115–126, 2001.

[12] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu. SEAL: Spatio-Textual Similarity Search. In *VLDB*, pages 824-835, 2012.

[13] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, 2008.

[14] P. W. Foltz and S. T. Dumais. Personalized information delivery: An analysis of information filtering methods. *Commun. ACM*, 35(12):51–60, 1992.

[15] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *SSDBM*, 2007.

[16] W. Huang, G. Li, K.-L. Tan, and J. Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *CIKM*, pages 932–941, 2012.

[17] K. W.-T. Leung, D. L. Lee, and W.-C. Lee. Personalized web search with location preferences. In *ICDE*, pages 701–712, 2010.

[18] G. Li, J. Feng, and J. Xu. Desks: Direction-aware spatial keyword search. In *ICDE*, pages 474–485, 2012.

[19] S. Liu, G. Li, and J. Feng. A Prefix-Filter based Method for Spatio-Textual Similarity Join. In *IEEE TKDE*, 2013.

[20] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD Conference*, pages 349–360, 2011.

[21] S. B. Roy and K. Chakrabarti. Location-aware type ahead search on spatial databases: semantics and efficiency. In *SIGMOD Conference*, pages 361–372, 2011.

[22] H. Samet. *Foundations of Multidimensional and Metric Data Structure*. 2006.

[23] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.

[24] T. W. Yan and H. Garcia-Molina. Index structures for information filtering under the vector space model. In *ICDE*, pages 337–347, 1994.

[25] T. W. Yan and H. Garcia-Molina. Index structures for selective dissemination of information under the boolean model. *ACM Trans. Database Syst.*, 19(2):332–364, 1994.

[26] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou. Approximate string search in spatial databases. In *ICDE*, 2010.

[27] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, 2009.

[28] D. Zhang, B. C. Ooi, and A. K. H. Tung. Locating mapped resources in web 2.0. In *ICDE*, pages 521–532, 2010.

[29] R. Zhong, J. Fan, G. Li, K.-L. Tan, and L. Zhou. Location-aware instant search. In *CIKM*, pages 385–394, 2012.

[30] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, 2005.