Hmedeh Z, Kourdounakis H, Christophides V *et al.* Content-based publish/subscribe system for web syndication. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 31(2): 359–380 Mar. 2016. DOI 10.1007/s11390-016-1632-8

Content-Based Publish/Subscribe System for Web Syndication

Zeinab Hmedeh [1], Harry Kourdounakis [2], Vassilis Christophides [2], Cédric du Mouza [1], Michel Scholl [1], and Nicolas Travers [1]

[1] *CEDRIC Laboratory, Conservatoire National des Arts et Métiers, Paris 75141, France*

[2] *FORTH/ICS, University of Crete, Heraklion, GR-70013, Greece*

E-mail: zeinab.hmedeh@cnam.fr; kourdoun@csd.uoc.gr; christop@csi.forth.gr; {dumouza, scholl, nicolas.travers}@cnam.fr

Received July 17, 2014; revised December 21, 2015.

Abstract Content syndication has become a popular way for timely delivery of frequently updated information on the Web. Today, web syndication technologies such as RSS or Atom are used in a wide variety of applications spreading from large-scale news broadcasting to medium-scale information sharing in scientific and professional communities. However, they exhibit serious limitations for dealing with information overload in Web 2.0. There is a vital need for efficient real-time filtering methods across feeds, to allow users to effectively follow personally interesting information. We investigate in this paper three indexing techniques for users' subscriptions based on inverted lists or on an ordered trie for exact and partial matching. We present analytical models for memory requirements and matching time and we conduct a thorough experimental evaluation to exhibit the impact of critical parameters of realistic web syndication workloads.

Keywords pub/sub, subscription indexing, web syndication, partial matching, scalability

1 Introduction

Web 2.0 technologies have transformed the Web from a publishing-only environment into a vibrant information place where yesterday's passive readers have become active information collectors and content generators themselves. In this context, web syndication formats such as RSS or Atom emerge as a popular way for timely delivery of frequently updated Web content. According to these formats, information publishers provide brief summaries (textual snippets) of the content they deliver on the Web, called information items, while information consumers subscribe to a number of RSS/Atom feeds (i.e., channels) and get informed about newly published items. Given that the amount and diversity of the information generated on a daily basis in Web 2.0 is unprecedented, there is a vital need for efficient real-time filtering methods across feeds which allow users to effectively follow personally interesting information. For instance, an RSS feed of a newspaper delivers around 60 items per day[1]; thus a user who is interested in news may subscribe to several feeds and be flooded by a large number of uninteresting items. For these reasons, we advocate a content-based publish/subscribe paradigm for Web 2.0 syndication in which information consumers are decoupled (in both space and time) from information providers and they can express their interest to specific information items using content-based subscriptions. Rather than flooding users with all items from a channel, keyword-based subscriptions will be matched on the fly against the content of incoming items originating from different feeds.

To efficiently check whether all keywords of a subscription also appear in an incoming item (i.e., broad match semantics), we need to index the subscriptions. Count-based (CI) and tree-based (TI) are two main indexing schemes proposed in the literature for counting explicitly and implicitly the number of contained keywords. The majority of related data structures[2-4] cannot be employed for conjunctions of keywords (rather than attribute-value pairs) due to the space high-dimensionality. In this paper, we are interested in efficient implementations of both indexing schemes us-

Regular Paper

A preliminary version of the paper was published in the Proceedings of EDBT 2012.

ing inverted lists (IL)[5] for CI and a variant for distinct terms of ordered tries (OT)[6] for TI and study their behavior for critical parameters of realistic web syndication workloads. Although these data structures have been employed to evaluate broad match queries in the context of selective information dissemination[7] and sponsored search[8] or for mining frequent item sets[9-10], their memory and matching time requirements appear to be quite different in our setting. This is due to the peculiarities of web syndication systems which are characterized[11] 1) by information items of average length (25∼36 distinct terms) which are greater than advertisement bids (4∼5 terms[8]) and smaller than documents of Web collections (12K terms[7]) and 2) by very large vocabularies of terms (up to 1.5M terms). Note also that due to broad match semantics, information retrieval techniques for optimizing ILs (e.g., early pruning[5]) are not suited in our setting.

A detailed analysis of trie structures has not been conducted in the past while the ordered trie usage has been discouraged in pub/sub systems due to the prohibiting performance exhibited in other application areas studied in related work (e.g., document filtering in [12]). In our work, we are going one step forward in identifying real setting parameters under which trie structures became competitive. In a nutshell, the main contributions of this work are:

1) In Section 2, we consider three index structures implementing different counting techniques for pruning as early as possible non-matching subscriptions to an incoming item. The first two implement the CI scheme and rely on an inverted list (IL) of terms to the subscriptions that contain them. The count-based inverted list (CIL) variant stores each subscription into the ILs of all the terms it contains while the ranked-key inverted list (RIL) stores them once in the IL of its least frequent term. We finally consider an ordered trie (OT) of distinct terms implementing TI with factorization of common subscription prefixes and a variation that compacts paths of unary nodes called POT.

2) In Section 3, we provide a detailed probabilistic analysis of the size and number of visited nodes during matching of the three indexes. This analysis takes into account the distributions of term occurrences and of subscription sizes. In particular, we show how OT depth is bounded by the length of the indexed subscriptions, while its width is tied up by the size of the vocabulary of indexed terms. To the best of our knowledge, such an analysis has not been previously reported in the literature. We finally report on the deviation from the model simulation conducted on a set of real items using a Zipf distribution of terms and different subscription lengths.

3) Section 4 presents the partial matching approach. We allow a subscription to be notified whether its more important terms are present in the incoming item. Based on the term distribution, we assign a weight to all terms of the subscription. For items' terms that match the subscription, a notification occurs only when the cumulated weight is over a given subscription's threshold. We show how different structures can handle partial matching and that CIL, opposed to other structures, could be extended to efficiently process partial matching.

4) In Section 5, we conduct a thorough experimental evaluation of CIL, RIL and POT① using generated subscriptions of terms appearing in a real RSS/Atom testbed of items[1]. To the best of our knowledge, this is the first study investigating 1) how critical workload parameters, such as terms distribution, the size of the vocabulary and the sizes of subscriptions, affect the morphology of OT, i.e., the level of achieved factorization and 2) their scalability and performance in realistic settings (e.g., for 100M of subscriptions with 1.5M of distinct terms and real distribution of terms in items) for broad-matching and partial-matching.

Related work is presented in Section 6 while a summary which details conclusions of our experiments and future work is given in Section 7.

2 Subscription Indexes

In the pub/sub paradigm for web syndication, users submit long lasting (continuous) queries under the form of keyword-based subscriptions. Whenever a news item is published, it gets evaluated against the set of subscriptions submitted to the system and for every matching subscription, the corresponding subscriber is notified. The set of stored subscriptions is denoted by $\mathcal{S}$ and their total number by $|\mathcal{S}|$. Each subscription $s \in \mathcal{S}$ includes a set of distinct terms from a vocabulary $\mathcal{V}_S = \{t_1, ..., t_n\}$. The size of s, denoted by $|s|$, is the total number of distinct terms it contains. $\mathcal{I} = [I_1, ..., I_m]$ denotes the stream of incoming news items. News item $I \in \mathcal{I}$ is also formed by a set of terms ($I \subseteq \mathcal{V}_I$, with $\mathcal{V}_I$ as the vocabulary of items). Like [7], we make the com-

① As expected, our experiments confirm that the regular OT (ROT) is outperformed in terms of memory and matching time requirements.

mon assumption that $\mathcal{V}_S \subseteq \mathcal{V}_I$. However, it is worth noting that in reality, $\mathcal{V}_S$ may diverge from $\mathcal{V}_I$ significantly. In this context, a match occurs if and only if all of the terms (keywords) of a subscription s are also present in a news item I (i.e., broad match semantics).

Consider the set of subscriptions $\mathcal{S}$ illustrated in Table 1. Matching item $I = \{t_2, t_4\}$ against $\mathcal{S}$ will result in the set of matched subscriptions $\mathcal{S}_{\mathrm{M}} = \{s_4\}$ since t_2 and t_4 of s_4 are contained in I. A naive matching approach consists in testing whether the terms of every subscription are contained in the incoming news item. Clearly, this simple solution does not scale to millions of subscriptions. For this reason, to avoid testing non-matching subscriptions, an structured index has to be chosen. A widely used structure is the inverted list (IL) which maintains an inverse mapping for each term t_j to all subscriptions s that contain this item. It essentially confines the original search space only to subscriptions containing at least a term present within the item being matched. In particular, two variants of IL, namely the "count-based inverted list" and the "ranked-key inverted list", are studied in this paper. Additionally, an "ordered trie" structure is considered to exploit the term subset relations between subscriptions. To accommodate large vocabularies exhibited by web syndication systems, we also study a well-known variant of tries, namely "Patricia trie".

Table 1. Example of Keyword-Based Subscriptions

Subscription	Term
s_1	$t_1 \wedge t_2 \wedge t_4$
s_2	$t_1 \wedge t_3$
s_3	$t_1 \wedge t_2 \wedge t_5$
s_4	$t_2 \wedge t_4$
s_5	$t_1 \wedge t_3 \wedge t_6$

2.1 Count-Based Inverted List

Count-based inverted list (CIL) is essentially a mapping dictionary whose key is a term $t_j \in \mathcal{V}_S$ and whose value is the corresponding postings list $Postings(t_j)$, i.e., the set of subscriptions that contain the term. Furthermore, to implement broad match semantics, an additional structure has to be maintained: a counter per subscription keeps track of the number of remaining terms to be matched for a given subscription. The structure that maps every subscription $s \in \mathcal{S}$ to the number of remaining terms to be tested before reporting a matching is denoted by *Counter*.

Fig.1(a) depicts the CIL index for the example of Table 1. The postings list associated with t_2 is $Postings(t_2) = \{s_1, s_3, s_4\}$. Initially, $Counter = \{s_1 : 3, s_2 : 2, s_3 : 3, s_4 : 2, s_5 : 3\}$. Consider an incoming news item $I = \{t_2, t_4\}$. The subscription elements of $Postings(t_2)$ are first accessed and their corresponding counters are decremented, and thus $Counter = \{s_1 : 2, s_2 : 2, s_3 : 2, s_4 : 1, s_5 : 3\}$. Finally, after processing t_4, $Counter = \{s_1 : 1, s_2 : 2, s_3 : 2, s_4 : 0, s_5 : 3\}$ and a matching is reported for subscription s_4 since its counter becomes 0.

2.1.1 CIL — Construction

When a new subscription s is posted to the system, a new element labeled with the subscription identifier s is added to the postings lists of all its terms. Additionally, a new entry is inserted into *Counter* with the total number of distinct terms $|s|$ the subscription contains.

2.1.2 CIL — Matching

The matching process for an item I is given in Algorithm 1. Initialization consists of an exact copy of

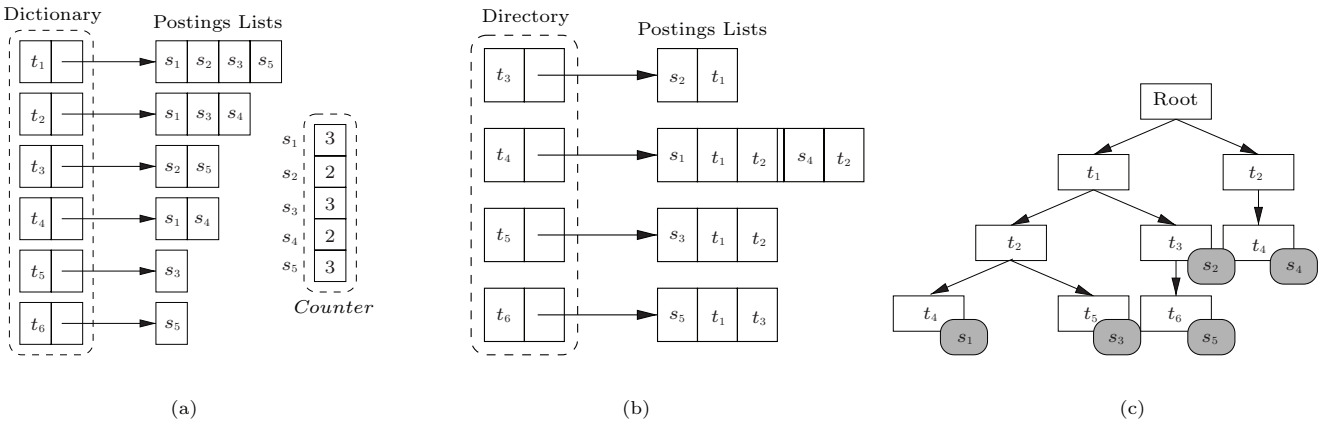

Fig.1. Subscription indexes. (a) Counted-based inverted list. (b) Ranked-key inverted list. (c) Ordered-trie index.

Counter into *Counter_copy* (line 1). For every term $t_j \in I$, $Postings(t_j)$ is accessed (line 3). For each subscription s in $Postings(t_j)$, the corresponding subscription value in *Counter_copy* is decremented (line 5). Whenever a counter reaches 0, a matching is reported.

Algorithm 1. CIL_MATCH(I)

```
Require: an item I
 1: Counter_copy ← copy of Counter
 2: for all terms t_j ∈ I do
 3:    PostSet ← Postings(t_j)
 4:    for all s in PostSet do
 5:       Counter_copy[s]
          ← Counter_copy[s] − 1
 6:       if Counter_copy[s] = 0 then
 7:          s_M ← s_M ∪ {s}
 8:       end if
 9:    end for
10: end for
```

2.2 Ranked-Key Inverted List

In contrast to CIL, in the ranked-key inverted list (RIL), a subscription is added only to the postings list of the least frequent among its terms. This term is called *key* of the subscription. Besides the subscription identifier, the elements of a postings list include the set of the remaining subscription terms. This variant obviates the need for an explicit *Counter* structure while accessing only the postings list of the most discriminating term of a subscription. More precisely, for every term of an incoming item, the corresponding postings list is accessed, and for each of its subscriptions, it is checked whether it contains the remaining item terms. Clearly the postings lists of the frequent terms are reduced in comparison with CIL and subscriptions are now distributed over a large number of postings lists of medium-frequent terms.

Fig.1(b) depicts the RIL index for the example of Table 1 where the term rank in frequency distribution is given by the term subscript (t_1 is the highest rank in $\mathcal{V}_S$). Not all the terms have entries in the dictionary. For instance, t_1 and t_2 do not appear as the least frequent terms in any of the subscriptions in $\mathcal{S}$. $Postings(t_4)$ has two elements (s_1 and s_4) for which the remaining (more frequent) terms to be checked are stored. With item $I = \{t_2, t_4\}$, we start by checking for subscriptions in $Postings(t_4)$. Since t_1 is not present in I which is required in subscription s_1, I does not satisfy s_1. Then in the next subscription s_4, t_2 appears in $I' = I - t_4 = \{t_2\}$ and there are no more terms in s_4, thus it is satisfied by I.

2.2.1 RIL — Construction

When a new subscription s is posted to the system, it is added to the corresponding postings lists. This implies to sort the terms of s by their rank. Then an entry labeled with the subscription identifier s is added to the postings list of the least frequent term followed by the remaining terms.

2.2.2 RIL — Matching

Matching an item I is given in Algorithm 2. When item I arrives, its terms are sorted by their rank (line 1). In the rest of the paper, unless otherwise specified, term subscripts are used to denote ordering. Then, the postings lists $Postings(t_j)$ of the least frequent terms are iteratively accessed (lines 2∼4). For every subscription element of $Postings(t_j)$, it is checked whether its remaining terms also appear in I (lines 6 and 7).

Algorithm 2. RIL_MATCH(I)

```
Require: an item I
 1: sorted_terms ← sort(I)
 2: while sorted_terms ≠ {} do
 3:    t_j ← less_frequent(sorted_terms)
 4:    sorted_terms ← sorted_terms − t_j
 5:    PostSet ← Postings(t_j)
 6:    for all s in PostSet(t_j) do
 7:       if s.remaining ⊆ sorted_terms then
 8:          s_M ← s_M ∪ {s}
 9:       end if
10:    end for
11: end while
```

2.3 Regular Ordered Trie

As an alternative to IL indexes, an OT index is capable of exploiting the term subset relations between subscriptions in order to build a hierarchical (as opposed to a flat) search space for sets of terms taking advantage of common prefixes of terms in subscriptions (i.e., factorization). A trie node represents a term and a subscription is stored at node n of the trie iff its terms are found in the path from the root to node n. Then, two subscription paths sharing a subset of k nodes can be merged in a single subpath of length k (i.e., common prefix), followed by two distinct paths representing the remaining subscription terms.

Clearly, in this structure, subscriptions are stored only once and there is no need anymore for an explicit *Counter* structure. Compared with the commonly used trie structures for storing sentences on a given vocabulary[6], two features characterize the ordered trie: 1) there is no repetition of terms in any

sentence (i.e., a subscription is a set of terms); 2) terms in the subscriptions and therefore in the trie are totally ordered. This total order could be random, follow the ranking of the terms occurrence distribution in subscription/new items whenever available, etc. This structure referred to as regular ordered trie (ROT) has been investigated in a different setting[7], as discussed in Section 6. It was also used more recently in data mining[10,13].

Fig.1(c) depicts the ROT index for subscription examples of Table 1 where the term rank is given by its subscript (t_1 has the highest rank in $\mathcal{V}_S$). Factorization leads to a single node t_1 for all subscriptions that share this term. Consider now an incoming item $I = \{t_2, t_4\}$ (terms already sorted). Initially, term t_2 is searched as the child of the root. Since such a node exists, navigation continues by looking for a t_4 child node. Since such a node also exists, subscription s_4 is reported as matching. Finally, term t_4 of I is processed and since no such path from the root exists, the matching concludes. Collapsing single paths in the trie to single nodes will reduce the number of nodes and consequently the memory occupied by the index, and accelerate matching. A variant of a Patricia ordered trie (POT) obeys this principle. Single paths corresponding to multiple nodes are compacted into a single node. Each compact node is labeled with the nodes' terms in the path. For instance, nodes t_2 and t_4 in Fig.1(c) are merged in a single node labeled with t_2 and t_4 in POT respectively.

2.3.1 ROT — Construction

Initially, the terms of a new subscription s are sorted according to their ranking order. Then the path corresponding to the first term of the ordered set of terms is followed from the root. This procedure is repeated for every term $t_j \in s$. If a particular path does not exist, then a new node labeled with the term under consideration is created and inserted into the trie structure. The node at which the top-down traversal concludes, after consuming the whole set of terms, stores s.

2.3.2 ROT — Matching

The matching process for an item I is given in Algorithm 3. When a news item I arrives, its terms are also sorted. Paths corresponding to all the terms of I, whose ranks are superior to that of the term assigned to the currently considered node, are followed (lines 5∼10). For every visited node, all linked subscriptions are reported as matching (lines 2∼4).

Algorithm 3. TRIE_MATCH($TNode, I$)

Require: $TNode$: the current trie node, I: an item
1: $sorted\_terms \leftarrow sort(I)$
2: **if** $TNode$ contains subscriptions **then**
3: $s_{\text{MATCHED}} \leftarrow s_{\text{MATCHED}} \cup \{s | s \in TNode\}$
4: **end if**
5: $sorted\_terms \leftarrow sorted\_terms - Term(TNode)$
6: **for all** term $t_j \in sorted\_terms$ **do**
7: $childNode \leftarrow$ get child for term t_j
8: **if** $childNode \neq$ NULL **then**
9: $TRIE\_MATCH(childNode, sorted\_terms - \{t_j\})$
10: **end if**
11: **end for**

2.4 Index Dependent Definition of Nodes

The three indexes presented previously essentially implement a relation R of vocabulary terms within subscriptions. The specific technique employed for counting the matching terms in subscriptions to satisfy broad match semantics identifies each index. As a consequence, the node definition, inherent to each data structure, is also different. In CIL, an index node is a tuple (t_j, s) expressing that subscription s contains term t_j. If s has k terms, k nodes (so k R-tuples) are needed to represent s. In RIL, an index node is a nested tuple $(t_k, \{(s, \{t_j\})\})$ where t_k is the key term, s the corresponding subscription and $\{t_j\}$ the remaining terms to be checked. In ROT, an index node is a nested tuple $(\{t_j\}, s)$ where $\{t_j\}$ is the set of terms encountered in the path from root to this node which forms the actual content of subscriptions in s. When subscriptions share the same prefix, it is expected to create less index nodes in trie-based indexes than in inverted list-based indexes. The rationale in studying the index behavior in terms of abstract nodes is to understand the impact in the morphology of the three indexes (and thus of the search space) of critical parameters of web syndication systems (i.e., the distribution of term occurrences or subscription sizes).

3 Analytical Models

This section is devoted to analytical modeling for predicting the number of nodes of the CIL, RIL and ROT structures presented in Section 2 as well as for predicting the number of visited nodes upon matching. Parameters and notations that affect construction time, memory requirements and matching time are summarized in Table 2.

Table 2. Parameters Characterizing the Workload

Parameter	Definition
$\|\mathcal{S}\|$	Total number of subscriptions
$\|\mathcal{V}_I\|$, $\|\mathcal{V}_S\|$	Vocabulary size of items and subscriptions
$\|s\|_{\text{avg}}$, $\|s\|_{\max}$	Average and maximal subscription size respectively
$\|I\|$	Average news item size
$P(t_j)$	Frequencies distribution of terms in $\mathcal{V}_I$
$\theta(k)$	Probability for a subscription to have a size k
σ_i	Probability to have a term with a rank $\leqslant i$
$w(c), w(v)$	Size of counter/dictionary entry
$w(p), w(n)$	Size of subscription posting entry/trie node

3.1 Building Time

$|s|_{\text{avg}} = \Sigma_{k=1}^{|s|_{\max}} \theta(k) \times k$, is the average size of a subscription where $\theta(k)$ is the probability that a subscription has a size k with $k \in [1, |s|_{\max}]$. Thus the time required to insert a subscription into CIL is $O(|s|_{\text{avg}})$ (insert $|s|_{\text{avg}}$ new postings and a counter). RIL requires to sort the terms before insertion to determine the less frequent one which is performed in $O(|s|_{\text{avg}} \times \log |s|_{\text{avg}})$. The time required to insert a subscription into ROT is $O(|s|_{\text{avg}} \times \log |s|_{\text{avg}})$. This is due to the fact that using a hash-based implementation of trie nodes, the time required to sort the terms of subscriptions dominates. Observe that indexing time for all structures is independent of the total number of stored subscriptions $|\mathcal{S}|$.

3.2 Memory Requirements

Let $P(t_j)$ denote the frequency of occurrences of term $t_j \in \mathcal{V}_I$. It is assumed that the choice of $t_j \in s$ is independent of the choice of any other term $t_m \in s$. In addition, let $\theta(k)$ be the probability that a subscription has a length k with $k \in [1, |s|_{\max}]$. The probabilities that t_j is one of the terms of a subscription s, denoted by $\Pr(t_j \in s)$ and that t_j is one of the terms of at least one subscription in $\mathcal{S}$, denoted by $\Pr(t_j \in \mathcal{S})$, are:

$$\begin{aligned}\Pr(t_j \in s) &= 1 - \Pr(t_j \notin s)\\ &= 1 - \sum_{k=1}^{|s|_{\max}} \theta(k) \times (1 - P(t_j))^k, \qquad (1)\\ \Pr(t_j \in \mathcal{S}) &= 1 - \Pr(t_j \notin \mathcal{S})\\ &= 1 - \prod_{i=1}^{|\mathcal{S}|} (1 - \Pr(t_j \in s))\\ &= 1 - (\sum_{k=1}^{|s|_{\max}} \theta(k) \times (1 - P(t_j))^k)^{|\mathcal{S}|}.\end{aligned}$$

Then the number of terms in the vocabulary of subscriptions $\mathcal{V}_S$ is equal to: $|\mathcal{V}_S| = \sum_{j=1}^{|\mathcal{V}_I|} \Pr(t_j \in \mathcal{S})$.

3.2.1 CIL Memory Requirement

Recall that the count-based index is composed of two structures, *Counter* and the inverted lists that are themselves further decomposed into the dictionary and the subscription postings. Thus the overall memory required by the index is:

$$\begin{aligned}Size(CIL) = {} & Size(Dictionary) +\\ & Size(Postings) + Size(Counter).\end{aligned}$$

The memory required by *Counter* is equal to:

$$Size(Counter) = |\mathcal{S}| \times w(c). \qquad (2)$$

We assume a low collision rate. Then the memory occupied by *Dictionary* is equal to:

$$Size(Dictionary) = |\mathcal{V}_S| \times w(v). \qquad (3)$$

Finally, since each term of any subscription leads to an entry in the corresponding postings list, the expected total number of entries in the postings list is equal to $|\mathcal{S}| \times |s|_{\text{avg}}$ and the size of the postings list is equal to:

$$Size(Postings) = |\mathcal{S}| \times |s|_{\text{avg}} \times w(p). \qquad (4)$$

In addition, for computing the matching time, we need to know the size $Size(Postings(t_j))$ of the postings list of a term t_j:

$$Size(Postings(t_j)) = \frac{\Pr(t_j \in \mathcal{S})}{\sum_{i=1}^{|\mathcal{V}_S|} \Pr(t_i \in \mathcal{S})} \times |\mathcal{S}| \times |s|_{\text{avg}}, \qquad (5)$$

where $\Pr(t_j \in \mathcal{S}) / \sum_{k=1}^{|\mathcal{V}_S|} \Pr(t_k \in \mathcal{S})$ is the normalized frequency of t_j in the postings. Thus the space consumed by the index is:

$$\begin{aligned}Size(CIL) \overset{(2,3,4)}{=} {} & |\mathcal{V}_S| \times w(v) + |\mathcal{S}| \times |s|_{\text{avg}} \times w(p) +\\ & |\mathcal{S}| \times w(c).\end{aligned}$$

Here (2, 3, 4) in the above formula refers to equations used to deduce the right part of the equation.

3.2.2 RIL Memory Requirements

RIL is decomposed into *Dictionary* which stores the set of terms that are the less frequent in at least one $s \in \mathcal{S}$, and the postings lists which store the set of subscriptions. Since virtual nodes in CIL and RIL

consist in subscription or term IDs, we assume that both structures share the same size $w(p)$ for posting entry. Thus $Size(Postings)$ is computed as for CIL ((4)). $Size(Dictionary)$ is however less than the CIL's one.

$$Size(RIL) = Size(Dictionary) + Size(Postings).$$

A subscription s belongs to a postings list $Postings(t_j)$ iff $t_j \in s$ and there is no term $t_i \in s$ with $i < j$. Thus $Post(s, t_j)$, the probability that a subscription s belongs to $Postings(t_j)$, is:

$$Post(s, t_j) = \sum_{k=1}^{|s|_{\max}} \theta(k) \times k \times P(t_j) \times (\sigma_{j-1})^{k-1},$$

where $\sigma_j = \Sigma_{i=1}^{j} P(t_i)$ is the probability to have a term with a rank higher than i. Indeed, if the subscription length is k, then there are k ways of choosing t_j, the remaining terms being chosen among the terms with higher rank. The probability to have a term t_j in *Dictionary* is the probability to have at least one subscription in corresponding $Postings(t_j)$:

$$\Pr(t_j \in Dictionary) = 1 - (1 - Post(s, t_j))^{|\mathcal{S}|}.$$

Thus the size of *Dictionary* is:

$$\begin{aligned} &Size(Dictionary) \\ &= \sum_{j=1}^{|\mathcal{V}_I|} (1 - (1 - Post(s, t_j))^{|\mathcal{S}|}) \times w(v). \end{aligned} \tag{6}$$

The overall memory required by the index is:

$$\begin{aligned} Size(RIL) \overset{(4,6)}{=} &\sum_{j=1}^{|\mathcal{V}_I|} 1 - (1 - Post(s, t_j))^{|\mathcal{S}|} \times w(v) + \\ &|\mathcal{S}| \times |s|_{\text{avg}} \times w(p). \end{aligned}$$

Finally, when considering the length of each subscription to be stored in the postings list, we deduce:

$$\begin{aligned} Postings(t_j) = |\mathcal{S}| &\sum_{k=1}^{|s|_{\max}} k \times \theta(k) \times k \times \\ &P(t_j) \times (\sigma_{j-1})^{k-1}. \end{aligned} \tag{7}$$

3.2.3 ROT Memory Requirements

Although the analysis of the regular trie can be found in particular in [14], to our knowledge, the following is the first attempt to predict the expected number of nodes of a regular ordered trie. The analysis takes into account any term distribution and any distribution of the subscriptions size. However it does not provide a closed form. Therefore its applicability is limited to vocabularies with size $|\mathcal{V}_I| < 100$ and short subscriptions with size $|s| < 12$. It turns out that this is the case when the vocabulary is restricted to the terms of an item. We show that the analysis is useful for computing the expected time to match an item against a set of subscriptions. Let $\mathcal{P}$ be a path from the root to a node in the trie representing term (labeled with) t_i. Its label Λ is defined as follows:

1) $\Lambda = \lambda$ is the empty path label,

2) if $\mathcal{P}$ with label Λ is a path from root to a node labeled with t_i, then $\Lambda \bullet j$ is the path label ending at node labeled with t_j whose parent is tail($\mathcal{P}$) labeled with t_i.

In the following, for short, we shall say that node j has for a prefix Λ or that j (tail($\mathcal{P}$)) has for an address $\Lambda \bullet j$ (Λ). Let s be a subscription. There is a path in the trie with the ordered sequence of ranks of terms in s. It is noteworthy to mention that there are possibly $\binom{k}{k-|P|}$ subscriptions sharing prefix Λ. Given s, $Q(\Lambda, j)$ denotes the probability that the node with address $\Lambda \bullet j$ belongs to s.

Lemma 1. $Q(\Lambda, j) = \sum_{k=|\mathcal{P}|+1}^{|s|_{\max}} \theta(k) \times Q(\Lambda, j, k)$ *where* $Q(\Lambda, j, k)$ *denotes the probability that the node with address* $\Lambda \bullet j$ *belongs to a subscription with size* k *and is equal to:*

$$\begin{aligned} Q(\Lambda, j, k) = &\binom{k}{k - |\mathcal{P}|} \prod_{m \in \mathcal{P}} P(t_m) \times P(t_j) \times \\ &(1 - \sigma_j)^{(k - |\mathcal{P}| - 1)}. \end{aligned} \tag{8}$$

Proof. The probability that term t_j belongs to s at address $\Lambda \bullet j$ is $\Pi_{m \in \mathcal{P}} P(t_m) \times P(t_j) \times (1 - \sigma_j)^{(k - |\mathcal{P}| - 1)}$, since the remaining $k - |\mathcal{P}| - 1$ nodes of s must be drawn in $\mathcal{V}_I - \{t_1, \ldots, t_j\}$ (recall σ_j denotes the sum of probabilities of the first j terms). Since there are $\binom{k}{k-|\mathcal{P}|}$ possible subscriptions, we obtain (8). □

We denote $P(\Lambda, j)$ as the probability that node n with address $\Lambda \bullet j$ exists in at least one subscription. Thus node n is said to be occupied with probability $P(\Lambda, j)$. Then the probability that a node $\Lambda \bullet j$ is never occupied by any subscription of $|\mathcal{S}|$ is denoted by:

$$P(\Lambda, j) = 1 - (1 - Q(\Lambda, j))^{|\mathcal{S}|}. \tag{9}$$

Finally let $E(\Lambda, j)$ denote the expected number of nodes of the trie with root $\Lambda \bullet j$ where j has for a prefix Λ.

Theorem 1. $E(\Lambda, j) = \sum_{m=i+1}^{|\mathcal{V}_I|-|s|_{\max}+|\mathcal{P}|} P(\Lambda \bullet j, m) \times (1+E(\Lambda \bullet j, m))$ *with* $E(\Lambda, j) = 0$ *if* $|\mathcal{P}| > s$ *or* $j \geqslant |\mathcal{V}_I|$. *Indeed, if node with address* $\Lambda \bullet i \bullet j$ *is occupied, the expected size of the trie with root* $\Lambda \bullet i \bullet j$ *is equal to* $E(\Lambda \bullet i, j)$. *Last the expected size of ROT is expressed as:*

$$Size(ROT) = E(\lambda, 0) \times w(n).$$

3.3 Matching Time

3.3.1 CIL Matching Time Requirements

The time complexity of matching an item I against the set of indexed subscriptions $\mathcal{S}$ with Algorithm 1 is equal to the time needed to copy the counter, $Time(Copy\_counter)$, and the time for dealing with postings lists entries. The latter depends on the number of times the critical inner loop (lines 4∼9) is executed, i.e., the sum of the sizes of all the postings lists corresponding to terms t_j in item I. The constant time required to perform a counter decrement or test (resp. to copy an entry of the counter) is denoted by τ_{decr} (resp. τ_{copy}). Then the time needed to perform matching is:

$$\begin{aligned}
&TimeMatch(CIL)\\
=\ &Time(Copy_counter) + Time(Postings)\\
=\ &|\mathcal{S}| \times \tau_{\text{copy}} + \sum_{j=1}^{|I|} Size(Postings(t_j)) \times \tau_{\text{decr}}\\
\overset{(5)}{=}\ &|\mathcal{S}| \times \tau_{\text{copy}} +\\
&\left(\sum_{j=1}^{|I|} \frac{\Pr(t_j \in \mathcal{S})}{\sum_{i=1}^{|\mathcal{V}_S|} \Pr(t_i \in \mathcal{S})} \times |\mathcal{S}| \times |s|_{\text{avg}} \right) \times \tau_{\text{decr}}.
\end{aligned}$$

3.3.2 RIL Matching Time Requirements

The time needed to match an item I depends on the number of its terms and the size of the corresponding postings lists. First we must sort its terms (set up phase) and then run through the postings lists to check the inclusion of the subscriptions. The matching cost is estimated as:

$$\begin{aligned}
&TimeMatch(RIL)\\
=\ &Time(Sort) + Time(Postings)\\
=\ &|I| \times \log|I| + |I| \times \sum_{j=1}^{|\mathcal{V}_I|} Size(Postings(t_j)) \times \tau_{\text{chk}}\\
\overset{(7)}{=}\ &|I| \times \log|I| + |I| \times \sum_{j=1}^{|I|} |\mathcal{S}| \times\\
&\sum_{k=1}^{|s|_{\max}} k \times \theta(k) \times (k \times P(t_j) \times \sigma_{j-1})^{k-1} \times \tau_{\text{chk}},
\end{aligned}$$

where τ_{chk} is the time needed to check term inclusion in I.

3.3.3 ROT Matching Time Requirements

From the previous set of subscriptions $\mathcal{S}$ whose size obeys distribution $Dist_k$ with vocabulary $\mathcal{V}$, the resulting ROT is denoted by $T(\mathcal{S}, Dist_k, \mathcal{V})$.

Definition 1 (Restriction of a Trie). *The restriction* $T'(\mathcal{S}, Dist_k, \mathcal{V}') = \Delta_{\mathcal{V}'}(T(\mathcal{S}, Dist_k, \mathcal{V}))$ *is the sub-trie of* T *on vocabulary* $\mathcal{V}' \subset \mathcal{V}$ *with the same maximal depth as the maximal size of a subscription* $|s|_{\max}$.

By definition, T' is pruned when its terms are not in $\mathcal{V}'$. Subscriptions s are contained into T' when there terms are all in $\mathcal{V}'$, as well as subscriptions whose prefix is defined in $\mathcal{V}'$ but tail is out of $\mathcal{V}'$. All nodes of the prefix of the latter subscriptions are occupied as well. Theorem 2 allows to predict the number of nodes visited upon matching an item against the set of subscriptions. The set of terms of an item ($\mathcal{V}(I)$) is very small: it is on average equal to 25 in our datasets for experiments. Then the expected number of visited nodes of the restriction of the ROT to the vocabulary of item $\mathcal{V}(I)$, $T'(\mathcal{S}, Dist_k, \mathcal{V}(I))$, is the expected number of nodes visited for matching I. T is used as a short cut for $T(\mathcal{S}, Dist_k, \mathcal{V})$.

Theorem 2. *The expected number of visited nodes for matching item* I *against the subscriptions of trie* T (*Algorithm* 3) *is equal to the expected number of nodes of the restriction* $\Delta_{\mathcal{V}(I)}(T)$ *where* $\mathcal{V}(I)$ *is the vocabulary of* I.

Then the expected number of visited nodes for matching item $I = \{t_1, \ldots, t_k\}$ which has been sorted on the term ranks, is expressed, using $P(\Lambda, j)$ in (9), as:

$$\begin{aligned}
&VisitedNodes = E(\lambda, 0), \qquad \text{where}\\
&E(\lambda, j) = \sum_{m=j+1}^{|I|-|s|_{\max}+1} P(\lambda \bullet j, m) \times (1 + E(\Lambda \bullet j, m)).
\end{aligned}$$

Proof. Theorem 2 evaluates the average number of nodes occupied. A node n to be visited by Algorithm 3 is a node occupied. Either n or the sub-trie of root n possibly contains subscriptions to be checked for matching. A node has no child (line 8 of Algorithm 3) when it reachs maximal depth or when remaining terms are not in $\mathcal{V}'$. There are no other nodes to be visited. □

Table 3 validates this model against actual measures on real items obeying the Zipf distribution and 1M subscriptions with fixed sizes ($|s|_{\max}$). An average was taken over 5K real items chosen randomly from the English items crawled in [1]. We notice that the deviation slightly increases with subscriptions size. With large subscriptions, the depth of the trie is higher, thereby the approximations of the computations at each level are propagated. For a large number of leaf nodes, the sum of approximations done becomes more significant.

Table 3. Real vs Estimated Number of Visited Nodes

$\|s\|_{\max}$	Real	Calculated by Theorem 1	Deviation (%)
2	426.70	426.89	+0.043
3	538.99	538.86	−0.024
4	576.52	575.77	−0.130
5	594.66	590.03	−0.770
6	600.15	595.64	−0.750
7	603.36	596.06	−1.200
8	599.80	592.17	−1.270

The total matching time, given by (10), is equal to the time needed to sort the item's terms and the sum of the time spent on the visited nodes. The average time spent on a single node is τ_n.

$$\begin{aligned} & TimeMatch(ROT) \\ &= Time(Sort) + Time(Nodes) \\ &= |I| \times \log|I| + E(\lambda, 0) \times \tau_n. \end{aligned} \tag{10}$$

4 Partial Matching

The broad-match semantics can be too restrictive for some subscriptions, especially for long ones where the probability of matching is very low. We investigate in this section how different structures are able to manage the partial matching, i.e., they notify a subscription when the "most characteristic" terms are present in an item. We show that different structures support partial matching when processing preliminary to subscriptions' decomposition in sub-subscriptions of interest. The structure of CIL also allows an extension to support the processing of partial matching.

4.1 Term Weight

Our partial matching relies on scores for a matching between the subscription and an item, thus we need to define how to weight terms of the subscriptions. We assign a weight to each term in $|\mathcal{V}_S|$, the vocabulary of subscriptions which represents their importance. Several term weighting models are proposed in literature like the term frequency (TF) combined with inverse documents frequency (IDF)[15], the term discrimination value (TDV)[16] or the term precision[17].

In our context, we rely on the TDV weighting function which is more adapted to the quality of vocabularies in web syndications systems. In fact, an item is a short set of terms where term frequencies cannot be used, and thus the tf/idf standard function is unsuitable. Moreover, the TDV weighting function measures how a term helps to distinguish a set of documents (i.e., the term influence on the global entropy). Therefore basically for our subscriptions set, neither a very frequent term (present in many subscriptions, thus this term is not a selective filter for subscriptions), nor a very uncommon term (present in very few subscriptions, thus assuming this is not a typo, it will probably never lead to a notification) has an important TDV value. Finally, the simplicity of computation is all the more important, since we only have to compute a sum of weights for each subscription as what we will see in Subsection 4.2.

More precisely, the discrimination value for a term t_k is the difference between the occurrence matrix's vector-space density and the matrix's vector-space without t_k. Therefore, assuming a similarity distance $sim(I_1, I_2)$ between items, like the cosine of the euclidian distance, we compute the density as the average pairwise similarity between distinct items:

$$\Delta(\mathcal{I}) = \frac{1}{|\mathcal{I}| \times (|\mathcal{I}| - 1)} \sum_{i=1}^{|\mathcal{I}|} \sum_{j=1 \wedge j \neq i}^{|\mathcal{I}|} sim(I_i, I_j),$$

where $|\mathcal{I}|$ is the size of the itemset.

Finally the TDV value for a term t_k is:

$$tdv(\mathcal{I}, t_k) = \Delta(\mathcal{I} - \{t_k\}) - \Delta(\mathcal{I}).$$

We denote for simplicity reason $tdv(t_k)$ instead of $tdv(\mathcal{I}, t_k)$ whenever there is no ambiguity. Based on this function, we can weight the different terms of the query. Each term weight is the TDV value normalized by the sum of TDV values of the query terms.

Definition 2 (Query Term Weight). *Let $s = \{t_1, t_2, \ldots, t_n\}$ be a subscription, the query term weight $\varrho(t_k, s)$ is:*

$$\varrho(t_k, s) = \frac{tdv(t_k)}{\sum_{i \in s} tdv(t_i)}.$$

4.2 Partial Matching Extension

Assuming the existence of a matching threshold κ, for an incoming item I, we intend to notify a subscription s whose matching score is greater than κ. The matching score $\mu(s, I)$ is defined as the sum of the query term weights of all terms of s that are matched by I, thus $\mu(s, I) = \sum_{t \in s \cap I} \varrho(t, s)$.

4.2.1 Subscriptions Decomposition Approach

A first approach consists of indexing for each given subscription, not only the subscription itself but also all the subsets of terms whose matching score is greater than κ. Consequently, in the worst case, for a subscription s, we have $2^{|s|} - 1$ possible partial matchings for s (i.e., the power set of s excluding the empty set). Note that the containment matching property allows to reduce the number of subsets to be indexed.

Property 1 (Containment Matching Property). *Let s' and s'' be subsets extracted from s. If $s'' \subset s'$ and $\mu(s'', I) \geqslant \kappa$, then $\mu(s', I) \geqslant \kappa$. In other words, all matchings of s' leading to the notification of s are already covered by s''.*

Thus we index for each given subscription, only subsets of its terms whose matching scores are greater than κ, and not included in any other subset candidate (including the whole subscription).

While the decomposition approach is the simplest way to handle partial matchings for all structures, CIL can also be extended to efficiently support partial matching. We can notice that both RIL and POT rely on a terms order, thus the most unfrequent or the first ranked term is mandatory. Then they are not candidates for an extension similar to CIL.

4.2.2 Extending CIL

We propose an extension of the CIL index to manage partial matching named CIL_p. The mapping dictionary in CIL_p is unchanged compared with CIL, but postings lists need to store the query term weight of different terms. Thus an entry in a postings list $Postings(t_j)$ is now a couple $(s_i, \varrho(t_j, s_i))$ where s_i is the identifier of the subscription that contains term t_j whose query term weight for s_i is $\varrho(t_j, s_i)$.

The CIL *Counter* is now replaced by a cumulative counter with an entry for each subscription that could produce a possible match (i.e., their matching score is greater than κ). These entries are initially set to 0. Note that we do not need to store *Counter* and to copy it before any matching attempt like in CIL, but we create a new one for each matching attempt. Then during the matching attempt, for every term $t_j \in I$, $Postings(t_j)$ is accessed. For each subscription s in $Postings(t_j)$, the corresponding subscription value in the cumulative counter is incremented by $\varrho(t_j, s)$. Whenever a counter's value goes over κ, a matching is reported.

4.2.3 CIL_p with Users' Thresholds

We shortly introduce a variant, CIL_pu, where each user is allowed to fix the matching threshold for its subscription. This choice only impacts *Counter*: like the basic CIL, *Counter* stores information for each subscription, i.e., the matching threshold κ_i for each subscription s_i. With this variant, a copy of the counter is performed for each matching, and then every term $t_j \in I$ leads to lists $Postings(t_j)$ of subscriptions s. The value of each corresponding subscription s in the *Counter* copy is decremented with $\varrho(t_j, s)$. Whenever a counter reaches 0, a matching is reported.

5 Performance Evaluation

The core implementation choices and the characteristics of the dataset of items and subscriptions are first presented, and then experiments illustrate the impact of different workload parameters on the morphology, the space requirement and the matching time. All experiments were run under Linux on a 3.60 GHz quad-core processor with 16 GB of memory.

5.1 Implementation

All indexes were implemented using the standard Java Collection Framework v1.6.0_20. We describe the implementation of each index and the motivation behind the choice of data structures and their different parameters.

5.1.1 Inverted Lists

Dictionary representing the inverted list in both CIL and RIL indexes is implemented using a static hash table inherited from Java hash map (see Fig.2). Subscription IDs are encoded in all structures as 4-byte integers. In CIL, due to collisions, a dictionary entry may correspond to more than one term. For this reason, each entry is associated with a linked list of term node, whose nodes are $(t_{\text{id}}, \uparrow subS_{\text{list}}, \uparrow next)$, where t_{id} is the term's identifier, $\uparrow subS_{\text{list}}$ is a pointer to a subscription list implemented as an array list and $\uparrow next$ a pointer to

the next node. Finally, *Counter* is a byte array of size $|\mathcal{S}|$ (a subscription size cannot exceed 2^8 terms). RIL is implemented in a similar way with two noteworthy differences: 1) the lack of *Counter*, and 2) besides subscriptions' IDs, the elements of subscription lists also keep track sequentially of the terms that remain to be checked per subscription.

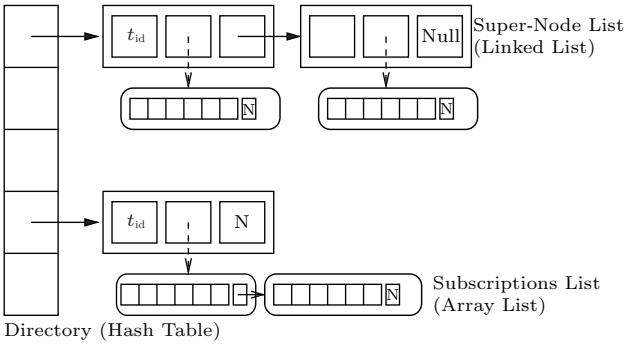

Fig.2. CIL/RIL implementation. N: Null.

5.1.2 Ordered Trie

Many different data structures (e.g., linked lists, arrays or trees) have been suggested for implementing efficiently a trie structure. We choose a hash tree based[18] implementation of the ROT (see Fig.3). Every internal node includes: 1) a 4-byte integer that stores the term's rank, 2) an array list for storing the corresponding subscriptions, and 3) a Java hash map for storing the children nodes. A leaf node only consists of the term's ID and the subscription list. We have paid particular attention to optimizing the memory requirements of internal and leaf nodes w.r.t. the number of their children. Therefore, we distinguish nodes with or without associated subscriptions, and with zero, one or several children. Each node type is equipped with a different implementation. For instance, node B is an internal node with a hash map to index its children, but has no subscription list associated with, unlike node C not holding any subscription. Note that leaf nodes like D or E do not contain any hash map since they have no child. Since nodes may store only one subscription, we further reduce the node size by using a single subscription's ID (node E) instead of an array list (node D). Finally, in the POT variant, the paths of unary nodes are compacted into a compact node labeled with the set of terms of the compacted unary ones. A 4-byte array is used to store the terms sorted by their ranks (node F). Despite its factorization gain in terms of abstract nodes (see Subsection 5.2), the memory requirements of a concrete POT node are clearly more important than those of CIL and RIL: while for CIL and RIL, a node occupies only 4 bytes, and it occupies on average 128 bytes for POT (a complex Java object with a hash map, pointers, array lists, etc.).

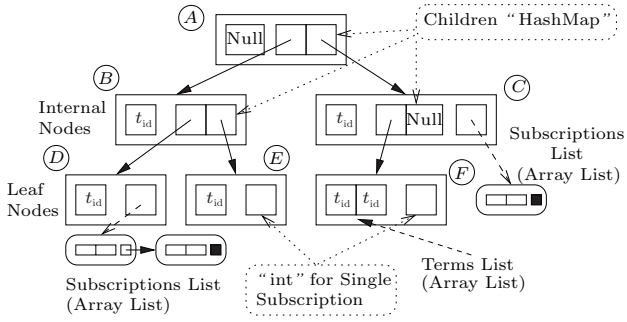

Fig.3. ROT/POT implementation.

5.1.3 Description of Synthetic and Real Datasets

The experimental evaluation relies on a large-scale testbed acquired over an 8-month campaign from March 2010 to October 2010[1]. A total number of 10.7M items were collected originating from 8K productive feeds (spanning over 2K different hosting sites)[1]. From the textual content of items, a vocabulary of 1.5M distinct terms was extracted and has been used for the synthetic generation of subscriptions. More precisely, we rely on the ALIAS sampling method[19] to generate subscriptions whose distinct terms follow a given occurrence distribution *Dist*. Three distributions are chosen for the subscriptions in the experiments: real (subscriptions obey the same term distribution as the news items term distribution), uniform, and inverse (subscriptions follow the inverse term occurrence distribution of items). Generated subscriptions are characterized by three features: 1) the vocabulary size and the occurrence distribution of terms in subscriptions $\mathcal{V}_S$, 2) the total number of generated subscriptions $|\mathcal{S}|$, and 3) the subscription size k that can be constant for all subscriptions, or follow a particular distribution. When not specified, we use the size distribution of web queries reported in [20]. It is characterized among others by a maximal size equal to 12 and an average equal to 2.2.

5.2 Size and Morphology of Indexes

This subsection is devoted to the impact of the subscriptions size and the terms' occurrence distribution on the index size and morphology. These parameters determine the degree of factorization achieved by ROT (or POT) compared with CIL (or RIL) on common subscription prefixes as well as the rank of terms for which

factorization is actually taking place. Both are essentially affecting the pruning opportunities (i.e., nodes visited) of the indexes during matching. In order to provide a common basis for comparing the morphology of the indexes, the number of index nodes is measured (for the index dependent definition of abstract nodes, see Section 2).

5.2.1 Expected Factorization Gain

Fig.4 depicts the number of ROT nodes created per term rank compared with CIL when indexing the same set of 10M subscriptions with a vocabulary $\mathcal{V}_S = 4.7 \times 10^5$ following the real distribution of terms in items[1]. Clearly, the distribution of the size of the CIL postings lists is identical to the distribution of terms' occurrences in subscriptions. We observe that the number of ROT nodes is significantly reduced not only w.r.t CIL (due to the factorization of common prefixes) but also w.r.t. the complete ordered trie (COT) with the maximal depth for the subscription of size 12.

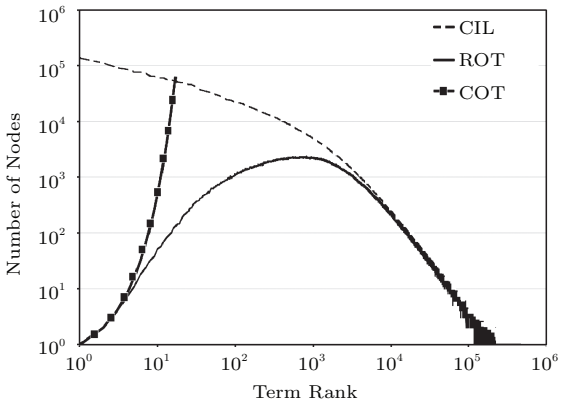

Fig.4. Number of nodes vs term rank.

Table 4 highlights the gain achieved in the number of nodes per term rank tr. More precisely, the gain is defined as $G_{tr} = (N_{tr}(CIL) - N_{tr}(ROT)) \ / \ N_{tr}(CIL)$, where $N_{tr}(CIL)$ and $N_{tr}(ROT)$ denote the number of nodes in CIL and ROT for term rank tr respectively. The number of occurrences, for a given term rank in the generated subscriptions as well as the number of ROT nodes that hold this term, is also given in this table. As expected, the gain decreases from almost 1 for rank 1 (most frequent term) to 0 for rank 470 000 (no factorization). The closer the number of nodes in ROT to CIL, the smaller the factorization. In this experiment, for all terms having a rank greater than 18 789, the gain is equal to 0.

Table 4. Gain per Term Rank

Rank	Number of Occurrences	Number of ROT Nodes	Gain (%)
1	138 090	1	99.99
10	60 469	52	99.91
1 000	4 967	2 201	55.69
10 000	251	218	13.15
470 000	1	1	0.00

5.2.2 Impact of Subscription Size

We now focus on how the size of subscriptions affects nodes factorization in ROT. The sets of subscriptions are generated using the same vocabulary as previously mentioned, but with a fixed size $k \in \{3, 6, 9, 12, 24, 36\}$. To provide a common comparison ground, the total number of term occurrences T in each set of subscriptions is fixed to $T = |\mathcal{S}| \times k = 1.5$M.

We observe in Fig.5 that the number of ROT nodes increases with k, i.e., factorization decreases with k. Indeed, the larger the subscriptions, the deeper the ROT. In this context, the probability that two subscriptions share the same terms decreases for larger subscriptions and the ROT behavior is close to the COT one. In other words, larger subscriptions imply more distinct paths. For instance, for subscription size $k > 24$, even for frequent terms (i.e., rank around 100), there is no gain. Oppositely, for subscription sizes $k \leqslant 24$, more occurrences of frequent terms are encountered in subscriptions and thus the structure fully benefits from the factorization gain. Of course, for $k \leqslant 2$, the number of subscriptions associated with leaves becomes large and the structure degenerates to an inverted list over term combinations (rather than individual terms).

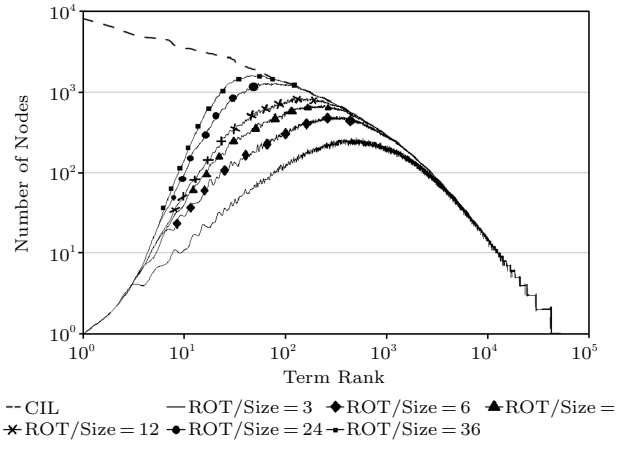

Fig.5. Number of nodes vs term rank for different subscriptions sizes.

5.2.3 Impact of Terms Order

To estimate the impact of the total order of terms when building ROT or POT, four orders are considered: 1) a frequency order (descending order in the number of term occurrences in subscriptions), 2) a reverse order (ascending order in the number of occurrences), 3) arrival order (order of arrival of terms in the subscriptions), and 4) random order (the term rank is randomly drawn).

The terms order has a limited impact on the size of ROT (see Fig.6). Compared with the frequency order, reverse, random and arrival order require 23.9%, 12.1% and 1.6% more nodes respectively. The difference between the frequency order and the arrival order is expected since more frequent terms appear statistically earlier than infrequent ones and there is an important factorization on the terms with the lowest ranks. For the reverse order, there is now a (low) factorization on the highly ordered terms but since these terms are infrequent, more subscriptions will be indexed in the sub-tries rooted at medium-frequency terms where factorization is more important. This explains why this order leads to more trie nodes, but the increase is limited.

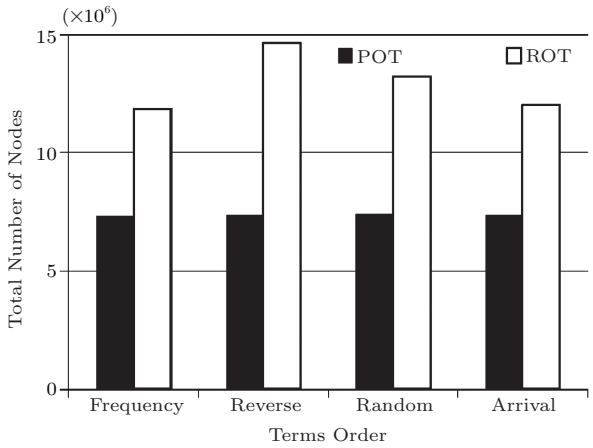

Fig.6. Impact of terms order on the number of created nodes.

Fig.7 shows that the terms order impacts more seriously the number of nodes visited during matching: e.g., an incoming item visits 36.1% more nodes in the frequency order than in the reverse one. The reverse order allows a better pruning than the frequency order since subscriptions featuring less frequent terms can be quickly filtered out. A similar result, in a different context, is reported in [7]. Surprisingly enough the order has an almost negligible effect on both the POT size and the number of nodes visited during matching. The number of trie nodes differs by less than 1% w.r.t. the considered order, and the difference between the numbers of visited nodes does not exceed 8%. Indeed, whereas the reverse order requires more nodes for ROT, subscriptions in sub-tries rooted at the low-ordered terms are poorly factorized. This poor factorization leads to many unary paths that benefit from the path compaction of POT.

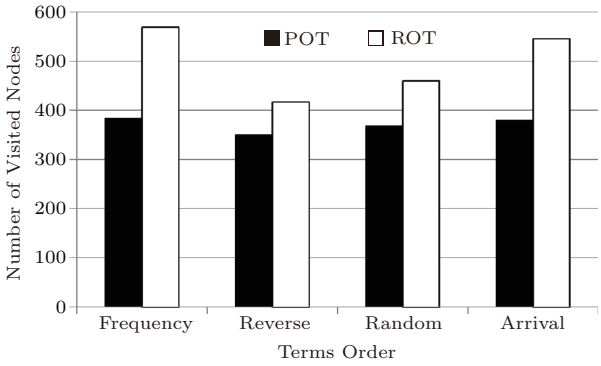

Fig.7. Impact of terms order on the number of visited nodes.

5.2.4 Impact of Terms Distribution

To investigate the impact of terms' occurrence distribution on the indexes, in addition to the real distribution employed previously, two more distributions are considered when generating subscriptions: 1) uniform and b) inverse where the most frequent terms in subscriptions correspond to the least frequent terms in items. Note that terms are ranked according to their frequency order in items.

Fig.8 shows that the terms' occurrence distribution in subscriptions has no impact on the size of the four indexes. For CIL and RIL, the number of nodes corresponds to the number of terms in subscriptions and thus it is independent from their occurrence distribution. Regarding ROT, real or inverse distributions have no impact on the index size. As a matter of fact, the left part of the trie structure becomes unbalanced for the former and this unbalance is shifted to the right for the latter. But in all cases, a similar factorization gain is observed. As expected, an uniform distribution results in a significantly larger ROT index. More combinations of terms are drawn, resulting in a more balanced trie with more paths rooted at internal nodes, and thus less factorization opportunities. Paths compaction attenuates this effect for POT.

In contrast, distributions seriously impact the numbers of visited nodes during matching (Fig.9). For CIL,

this number reaches 300 000 for real, but only 1 400 for uniform and it drops to 8 for inverse. ROT and POT exhibit a similar behavior with 400 nodes, 26 nodes, and one node visited on the average for the different distributions. For CIL, inverse leads to the scanning of shorter subscription lists leading to a lower matching cost. For a uniform distribution, subscription lists have almost the same size, while in real, subscription lists of frequent terms are particularly large with a high probability to be scanned for an incoming item. Regarding ROT and POT, uniform and inverse distributions yield a fast pruning, because of low factorization. For inverse, frequent item terms correspond to quite few subscriptions, and it is quite rare to match more than one term. For uniform, less subscriptions with a given prefix lead to less nodes having to be visited.

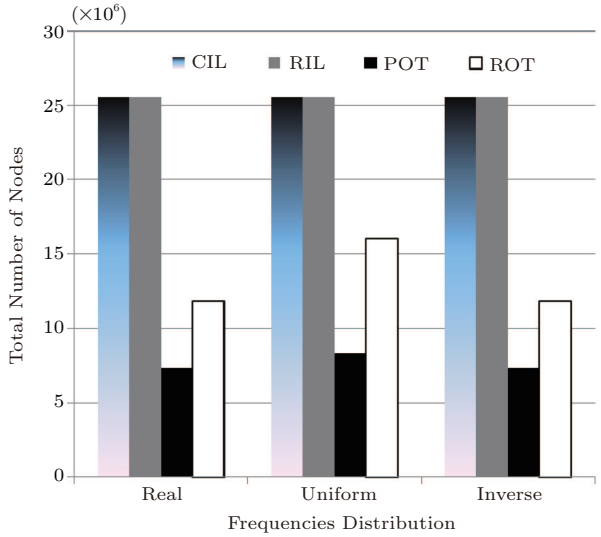

Fig.8. Impact of the distribution on the number of created nodes.

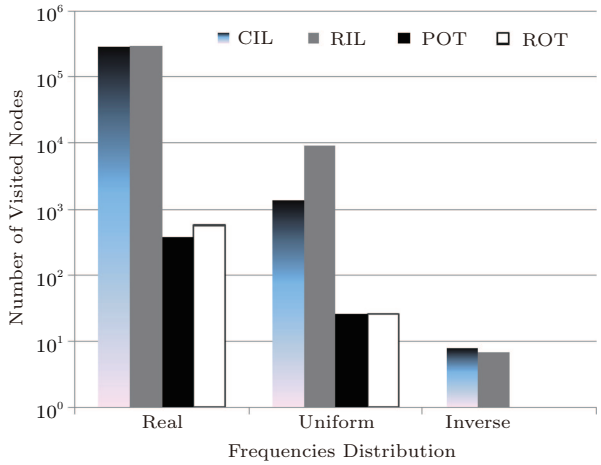

Fig.9. Impact of the distribution on the number of visited nodes.

5.2.5 *ROT vs POT by Scaling Vocabulary Size*

ROT will be excluded from the experiments of Subsection 5.3 due to its requirements on memory space and matching time. To understand this behavior, we studied the morphology of the trie indexes for different vocabulary sizes. As shown in Fig.10, the number of nodes in the trie increases linearly with the size of the vocabulary but is greater for ROT. That is why this index requires much more memory space and does not allow the scaling by the number of subscriptions indexed.

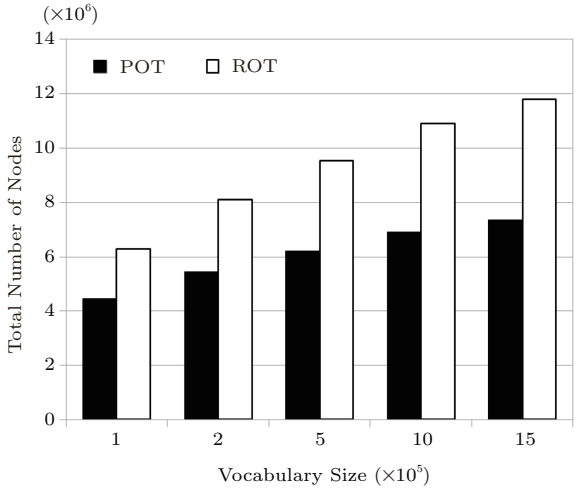

Fig.10. Number of nodes for different vocabulary sizes.

Regarding the number of visited nodes for matching an item, Fig.11 shows that the number of paths to compute evolves exponentially for the two structures. In fact, each trie is composed of much more nodes for large vocabularies, and thus more nodes have to be visited on matching items.

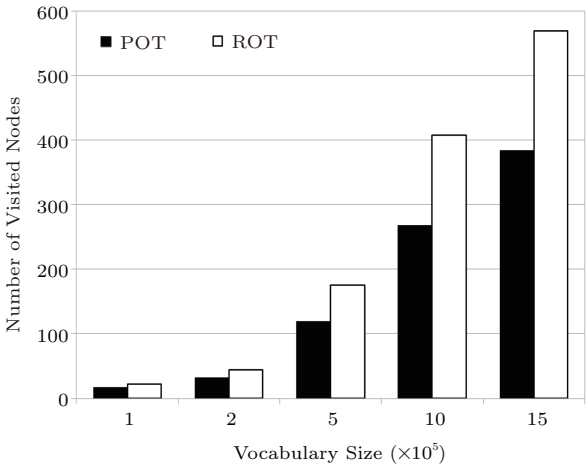

Fig.11. Number of visited nodes for different vocabulary sizes.

5.3 Scalability and Performance in Broad Matching

We turn now our attention to benchmarking memory space and matching/indexing time for CIL, RIL and POT. In particular, index scalability w.r.t. critical workload parameters of web syndication systems such as $|\mathcal{V}_S|$ and $|\mathcal{S}|$ is investigated. As said previously, ROT, whose space consumption reveals quickly scalability issues, has been discarded.

5.3.1 Memory Requirements

Fig.12 illustrates the evolution of the memory space for the three indexes for 10M of subscriptions when scaling vocabulary size $\mathcal{V}_S$. Using vocabularies of items $\mathcal{V}_I$ ranging from 100K to 1.5M terms, we generate subscriptions whose vocabularies $\mathcal{V}_S$ range from 87 839 to 471 324 terms. In general, IL indexes require a third of the memory required by POT. CIL and RIL space requirements slightly increase with vocabulary size, from 250 MB to 280 MB when the vocabulary size triples corresponding to 10% of increase. This is due to the fact that a larger vocabulary leads to a larger hash table for the *Dictionary* (we fix it as half size of the vocabulary) while subscription lists are constant, with $|s|_{\text{avg}} \times |\mathcal{S}|$ nodes for CIL and RIL. On the other hand, POT is more sensitive to the vocabulary size, since its space requirement grows from 710 MB to 925 MB, about a 30% increase. This is due to the appearance of more terms combinations in subscriptions which leads to more paths in the trie and less factorization opportunities.

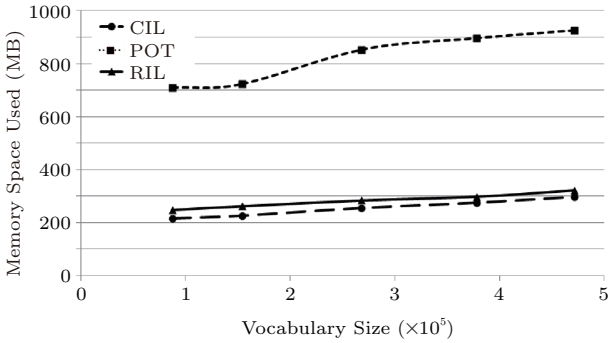

Fig.12. Memory footprint by scaling $|\mathcal{V}_S|$.

Fig.13 illustrates the memory space consumed by the three indexes for a vocabulary $\mathcal{V}_I$ (resp. $\mathcal{V}_S$) of 1.5M (resp. 1.2M) terms when scaling the number of subscriptions from 5M to 100M. As expected, the memory space consumed by CIL and RIL increases linearly with the number of subscriptions. Since the *Dictionary* size is fixed to half of the size of $\mathcal{V}_S$, only subscription lists consume more space to store the incoming subscriptions (i.e., a 4-byte id per new subscription). Surprisingly enough, POT's also exhibits a linear size growth. While a sub-linear growth is expected with factorization, this effect competes with the creation of new nodes. This happens when a subscription does not match with any of the trie paths or with alteration of existing ones when adding a new subscription list, resizing the array of an existing subscription list, or adding to a new hash map for its children. The gradient of the memory curve is four times larger for POT than for CIL and RIL. For instance for 100M subscriptions, the first requires 9 200 MB while the other two only 2 320 MB. Remember that despite its factorization gain in terms of abstract nodes (see Subsection 5.2), the memory requirements of a concrete POT node in our implementation are on average 128 bytes versus 4 bytes for CIL and RIL.

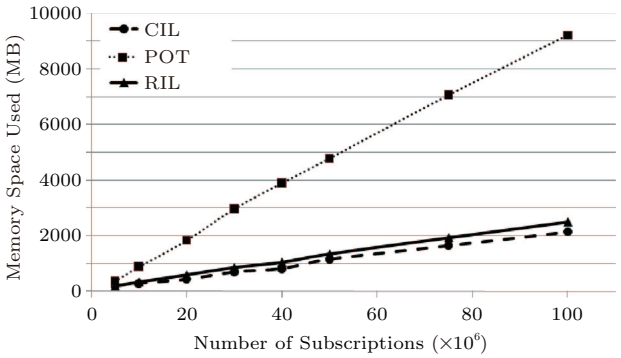

Fig.13. Memory footprint by scaling $|\mathcal{S}|$.

5.3.2 Matching Time

Fig.14 reveals that for 10M of subscriptions, RIL and POT outperform CIL by one or two orders of magnitude for all vocabulary sizes. For instance, a matching is performed in 4.33 ms (resp. 6.21 ms) for CIL with $|\mathcal{V}_S|$ equal to 87K (resp. 378K), while it requires only 0.55 ms (resp. 0.89 ms) for RIL and 0.03 ms (resp. 0.94 ms) for ROT. CIL matching leads to scanning large subscription lists, and consequently to decrement many counters, especially for items with 25 terms on average that are likely to contain several frequent terms. RIL takes advantage of the terms distribution to scan smaller subscription lists than CIL, since by construction only few subscriptions appear in the postings lists of frequent terms. POT benefits from a more drastic

pruning of the search space, since despite the theoretically large number of paths, in POT, not many term combinations actually exist (see the difference between the complete trie COT and ROT in Fig.4).

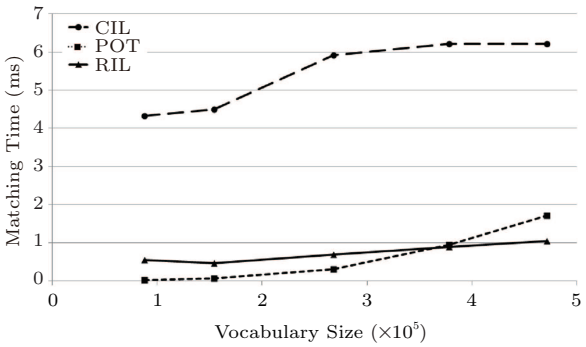

Fig.14. Matching time by scaling $|\mathcal{V}_S|$.

The vocabulary size $\mathcal{V}_S$ also affects indexes. CIL exhibits a convergent behavior: with larger $\mathcal{V}_S$, more subscription lists are scanned but since each one is small, the same number of nodes is visited. RIL matching cost grows linearly with $\mathcal{V}_S$: for large vocabularies, subscriptions are less likely to contain the least frequent terms, and consequently the subscription lists of the medium frequent terms are larger; since these lists have a higher probability to be scanned when matching an incoming item, more nodes are expected to be visited. In POT, this number increases exponentially with $\mathcal{V}_S$: trie has less factorization opportunities (nodes for frequent terms degenerate to inverted lists) and for an incoming item, more paths need to be explored. As a consequence, while POT outperforms RIL with one magnitude order for small vocabularies, RIL provides better performances for large ones (i.e., $|\mathcal{V}_S| > 378$K).

Fig.15 depicts the matching time for a vocabulary $\mathcal{V}_I$ of 1.5M terms when scaling the number of indexed subscriptions from 5M to 100M. In the three indexes, matching time scales linearly with the number of subscriptions. CIL and RIL subscription lists grow linearly with $|\mathcal{S}|$, thus achieving a constant gain which explains this linear behavior (the gradient remains 6.5 times higher for CIL). In POT, the index size grows with the number of subscriptions, and consequently the pruning effect decreases since more paths are possibly explored when matching an incoming item. Observe that POT slightly outperforms RIL for a large number of subscriptions: for 10^8 subscriptions, matching time is only 7.2 ms for POT versus 10 ms for RIL.

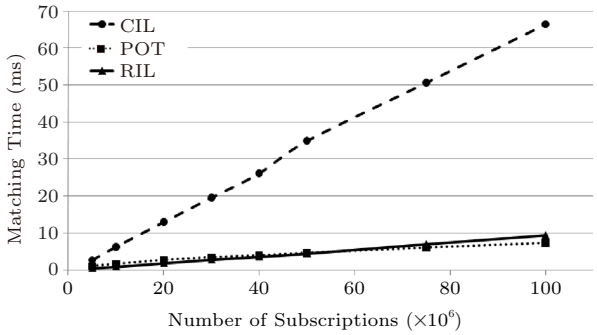

Fig.15. Matching time by scaling $|\mathcal{S}|$.

5.3.3 *Matching Time Analysis*

Previous results show that POT is much faster than CIL in matching time. To explain this order of magnitude between their matching time, we measured the time needed to execute the different instructions of their algorithm. Table 5 (resp. Table 6) illustrates the percentage of some of these instructions for POT (resp. CIL) algorithm. These tables present the average values of results of experiments on several sets of subscriptions (5, 10 and 25 millions of subscriptions).

Table 5. Matching Time for POT

Instruction	Percentage of Matching Time
Sort terms' IDs	1
Searching for nodes	89
Recursive retrieval	10

Table 6. Matching Time for CIL

Instruction	Percentage of Matching Time
Copy of counter	40
Scan postings list	10
Counter's decrements	50

As shown in Table 5, 89% of the time needed to match an item in POT is spent in searching and testing nodes and the pre-processing phase (search and sort terms' identifiers) does not need more than 1% of the time. In the CIL index, only 10% of time is used to search and scan postings lists. However, almost 50% of this time is spent in counter's decrements, which is due to the fact of processing all subscriptions. Another important thing is that 40% of the matching time is used in the copy of the counter, and oppositely this instruction does not exist in the RIL which gives the difference of time.

5.3.4 Indexing Time

Last, we measure the average time required to index 10M subscriptions generated from a vocabulary $\mathcal{V}_I$ of 1.5M terms. IL building time is in general faster than that of OT, with only 0.7 µs (resp. 1.0 µs) required to insert a subscription to RIL (resp. to CIL) while POT needs 1.7 µs. The additional POT overhead stems from the cost of converting a trie node from one type to the other (to accommodate added subscriptions, children, etc.). On the contrary, CIL requires only to add the new subscription to the corresponding postings lists of its terms while RIL indexing is even simpler since a subscription is added only to one postings list of its key term.

5.4 Partial Matching

We study here the experimentation of the partial matching semantics on our structures. More precisely we compare CIL, RIL and POT with the subscription decomposition approach, and CIL extension with threshold named CIL_p (see Subsection 4.2). We first measure the number of partial subscriptions (produced by the decomposition) indexed regarding different matching thresholds assuming the containment property (see Section 4) in Table 7. In the following, unless precised, we set the threshold to 0.75.

Table 7. Ratio (Partial Subscriptions Indexed/Initial Subscriptions) w.r.t. the Matching Threshold

Threshold	Ratio
0.5	2.50
0.6	2.22
0.7	1.75
0.8	1.33
0.9	1.08

5.4.1 Memory Requirements

We depict in Fig.16 the memory requirements for CIL, RIL and ROT when indexing all partial subscriptions whose matching score is greater than the matching threshold, along the variants CIL_p and CIL_pu. We observe that the memory requirement increases linearly with the number of subscriptions for all indexes. Compared with the memory space used in the broad matching, CIL, RIL and ROT require around 50% more space for storing all partial subscriptions, as expected (remember that for a threshold of 0.75, we have a ratio of 1.5, see Table 7). For CIL_p and CIL_pu, the array that stores the terms' weight consumes linearly more memory. For 10M subscriptions, the array of bytes reaches 20 MB (on average 2.2 entries are added for each subscription).

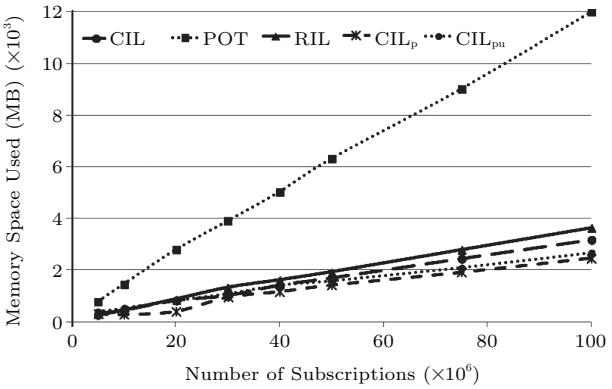

Fig.16. Memory used in broad vs partial matching by scaling $|\mathcal{S}|$.

5.4.2 Matching Time

Fig.17 shows that the matching time also scales linearly with the number of subscriptions for all indexes. While CIL_pu and CIL exhibit a similar matching time, we observe that they are outperformed by CIL_p. The rationale is that, unlike CIL_pu and CIL, CIL_p does not perform a copy of *Counter*. This results in a gain which increases with the size of *Counter*, i.e., $|\mathcal{S}|$.

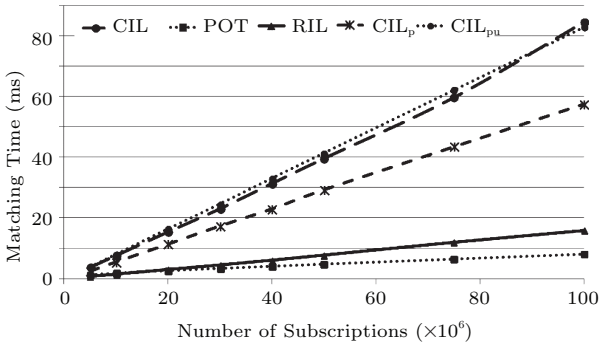

Fig.17. Matching time with partial matching by scaling $|\mathcal{S}|$.

Fig.18 illustrates that the matching time in the inverted lists indexes (CIL and RIL) is particularly sensitive to the matching threshold, while it has a small impact on POT and even no impact on CIL_p. Indeed small thresholds lead to more partial subscriptions, thereby for ILs indexes it results in larger postings list to scan. Since the matching time in the trie POT depends essentially on the number of visited nodes and not on the

number of subscriptions indexed, and partial subscriptions have a high probability to be included in existing subscriptions, the increase in matching time with the threshold remains moderate.

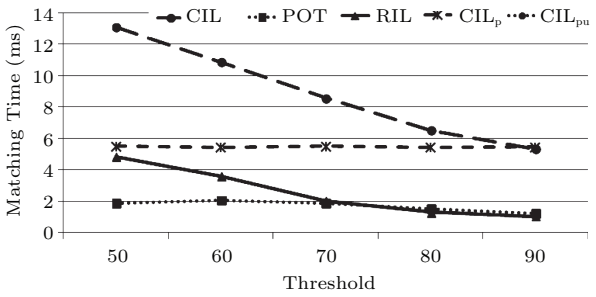

Fig.18. Matching time in partial matching w.r.t. threshold of matching.

5.4.3 *Indexing Time*

To add a subscription in CIL_p and its variant, we need to add the weight of the subscription's terms in the postings lists. That is why we observe that CIL_p requires 20% more time (0.85 μs) than CIL to index a subscription. Determining all partial subscriptions to be indexed largely increases the indexing time: 6.37 μs (resp. 6.28 μs) for CIL (resp. RIL) and 8.7 μs for POT with an average time needed to decompose a subscription equal to 5.4 μs.

6 Related Work

Several indexes have been proposed for matching efficiently structured (e.g., attribute-value pairs) or unstructured subscriptions (e.g., keywords) with incoming information items. In their majority, they rely on count-based (CI) schemes rather than on a tree-based (TI) structure. For example, LeSubscribe[2] Publish/Subscribe system employs a CI index to match subscriptions: first, the set of predicates satisfied by the incoming event is located using adequate predicate indexes, and second the set of matching subscriptions is obtained by explicitly counting the number of contained predicates. Subscriptions are systematically examined even when some of their predicates are not satisfied by the incoming event. For this reason, [3] proposes to group subscriptions according to their size as well as common (conjunctive) sub-expressions. Authors in [3] devised several cost-based algorithms for computing the optimal predicate clustering given knowledge regarding the statistics of subscriptions and incoming events while during matching, the cache capabilities of modern processors are exploited. An extension of the index LeSubscribe for disjunctive predicates (in limited contexts) has been proposed in [21].

A TI index for a two-phase matching of conjunctive subscriptions has been proposed in [4] which assumes a fixed total ordering among subscription predicates. In the first phase (pre-processing), a matching tree is built over the subscription predicates based on their ordering. Each tree node represents a test of some type while edges materialize the results of those tests. Lower levels are refinements of the tests performed in the higher levels of the tree while leaves store the subscriptions. When subscriptions are composed only of equality attribute/value predicates, [4] achieves matching time and space complexity that is sub-linear with respect to the number of subscriptions indexed. In the same spirit, RAPIDMatch[22] proposes a two-level partitioning on the set of indexed subscriptions which exploits the observation that in real-world applications, many events have only a few "relative" attributes. Hence, the search space of the RAPIDMatch TI is effectively pruned since it can quickly identify a small subset of matching subscriptions. A multi-dimensional TI is proposed in [23] which relies on a spatial reduction of the search space to points (for subscriptions) and range queries (for incoming events). These multidimensional range queries are then evaluated via a UB-Tree index. Furthermore, other dimensional TIs have been proposed for ranked versions of pub/sub systems such as [24], support convex indexable regions (for conjunctive predicates), and due to the inherent limitations of high-dimensional indexing do scale for a large number of attributes or attribute domains (or keywords).

[25] proposes a dynamic tree data structure named BE-tree (Boolean expression-tree) to index Boolean expressions. BE-tree scales to millions of Boolean expressions in high-dimensional space. It is a key-based index with a main memory implementation which outperforms [3-4, 7, 26]. An improved version named BE*-Tree index is proposed in [27] along analytical studies of the cost for adding a subscription, matching time and memory space used. However the time required to add a subscription depends not only on the subscription size (the number of predicates per expression) but also on the domain cardinality (the vocabulary size in our context) which is not the case for our regular ordered trie. Moreover the multi-paths BE-Tree matching algorithm has not been formally analyzed. A top-k pub/sub system based on BE*-tree is proposed in [28] for efficiently determining the most relevant matching subscriptions for a given event. Observe that all BE-tree variants do

not support partial matching unlike our proposal.

A graph-based pub/sub system is described in [29-30] to filter RSS feed. The authors represented subscriptions as tuples (subject, property, object, constraints (subject), constraints (object)) in a directed labeled graph. Vertices correspond to the subject and objects, linked by edges labeled by the properties. All subscriptions are merged in a single graph which is indexed using two-level hash tables. The first hash table contains all vertices (the name of vertices is taken as a hash key) and each entry in this hash table is a link to another hash table that contains the list of edges between these vertices. Each entry of the second hash table is a link to a subscriptions list. Based on non-deterministic finite state automata, [31] proposes a pub/sub system for matching stateful subscriptions on the stream of events such as RSS streams. Publications are considered as temporal events with start and end timestamps. Nonetheless while the solution presented allows expressive subscriptions which contain operators (e.g., when, where, only, and first), it does not scale to millions of subscriptions like our proposal.

Few pub/sub systems have been proposed for keyword-based subscriptions. Most noteworthy is the SIFT[12] selective dissemination of text documents whose alternative index schemes are thoroughly studied in [7]. In particular, in this work, authors experimentally evaluated the behavior of disk-based implementations of indices: an IL count based index, the so-called ranked key counter-less IL in which any substitution is sorted only in the postings list of its keyword with smallest rank, as well as two tries. One is a regular trie, and the other corresponds to our regular ordered trie (ROT): the latter assumes a total order on the keywords and the substitutions to be sorted according to this order②. It is expected that this ordering leads to many more common prefixes between substitutions. In contrast to the disk-based implementation in [7], we choose a central memory implementation of the indices: we argue that the index must reside in central memory, so that the performance scales to a large number of subscriptions (of the order of 10M). Besides the fact that we have designed memory-resident implementations of these indexes optimized for reads, the main differences in their observed behavior are due to 1) the size of the incoming set of words (on average 52 in web syndication items vs 12K in text documents), 2) the size of the subscription vocabulary (1.5M in web syndication vs 18 000 in text documents), and the distribution of terms' occurrences. This explains why authors in [7] observed that POT matching takes significantly more time than the two inverted list indices as the increased size in blocks leads to a higher number of I/O's per document. In contrast, we find that for small vocabularies, POT matching time is one order of magnitude faster than the best IL, namely the ranked key inverted list (RIL), while for large vocabularies, both exhibit a matching time of the same order (not studied in [7]). Moreover, they limit their performance evaluation to subscriptions with fixed size and keywords with uniform distribution. In contrast, not only do we evaluate the redundancy saving provided by a trie hierarchy with respect to ILs, but also we provide an analysis both of ILs and of the regular ordered trie (analysis which is to our knowledge novel), taking into account the statistical properties of the vocabulary as well as the subscription size distribution and several distributions of the keyword employed in the subscriptions, unlike [7] which advocates an intractable trie complexity and does not investigate the impact of different vocabulary distributions on the performance of the indexes. The only work exploiting CI rank information of terms occurrence distribution in order to optimize subscription matching is presented in [32]. However, due to our vocabulary size, the proposed query clustering techniques cannot be applied since superqueries turn out to be too large, which results in higher costs for both testing and hash table accesses.

On the other hand, ILs have been exploited in [8] for indexing advertisement bids in sponsored search. In particular, to retrieve the set of bids matching a query issued by a user, broach match semantics is also employed. Then, to overcome CI limitations when a skewed term distribution occurs in subscriptions, the authors in [8] relied on ILs built over the multi-term combinations most frequently appearing in bids, which reduces the search space to only a small fraction of candidate bids. The hash value of each IL entry is a pointer to a data node with bid specific information (i.e., its identifier, actual phrase, and metadata). In the simple case, IL indexes all terms in each bid (i.e., one data node for every indexed bid) and thus an incoming query requires retrieving data nodes associated with all subsets of the terms in the bid. Since the number of index probes grows exponentially with query size, authors in [8] considered a maximum length on indexed term combinations and proposed a mapping scheme that reorganizes bids sharing the same subset of terms to the

②They do not evaluate POT, our Patricia variant of the ordered trie.

same data nodes based on a memory access cost model. Clearly this optimization does not scale in our setting, given that the number of incoming items (on average 52 terms) is clearly larger than that of web queries (on average 2∼3 terms[20]).

A work for indexing Boolean expressions is presented in [26]. It relies on a CI approach which uses inverted lists for each pair (att/val) but removes the counting phase by creating several indexes for each subscription size (called K-index). In fact, an item is notified when it matches K pairs for the K-th index with a same subscription ID. It also avoids some unuseful tests for large subscriptions in which K values are larger than the item size. To optimize the matching process, sorts are done on matching lists and subscription IDs in lists in order to skip efficiently unmatching IDs. However, as said previously, our context brings quite large items (52 terms) and it cancels the large benefit of the K-index since all the postings lists should be accessed. Moreover, sorting very long postings lists will make the matching time grow up.

Finally, in [33], a top-k pub/sub approach is applied to news stories annotation in social news. News stories are considered as subscriptions, and tweets as published items. The matching score between a story and a tweet is based on their content, and a matching is reported if its score is greater than the k-th top scored items published previously for this specific subscription. An IL index is used to store subscriptions, with a posting for each term that contains all the identifiers of stories that contain this term. Authors in [33] also proposed to store along the story's ID, the score contribution of the term in the subscription which is used to compute the similarity between the story and an incoming tweet. This work is very similar to our extension of the CIL (CIL_p) for partial matching. However the nature of their data (100 000 long stories versus our dozens of millions of short subscriptions, 14-term incoming tweets against our 35-term items) and the matching computation lead to different optimizations and performances.

7 Conclusions

We presented and compared three index structures for web syndication: CIL and RIL, which rely on an inverted list, and POT based on a Patricia ordered trie. The technical novelty of our work comes from the thorough analysis of the complexity and experimental evaluation of the three chosen indexes. We found that for small vocabularies, POT matching time is one order of magnitude faster than the best IL (RIL), while for large vocabularies (like the one used on the Web), RIL outperforms the matching POT, which uses almost four times more memory space. The actual distribution of term occurrences has almost no impact on the size of the three indexing structures while it significantly affects the number of nodes that need to be visited upon matching something that justifies OT performance gains. The smaller the subscription length, the larger the OT factorization gain w.r.t. IL and the larger the rank of the term from which the OT substructure degenerates to an IL. We also introduced the partial matching semantics for our indexes, especially by extending CIL to count the score of satisfaction of subscriptions. Moreover, not only did we experimentally evaluate the redundancy saving provided by a trie w.r.t. IL structures, but also we proposed a first analysis of the ROT structure, especially based on the variation of the vocabulary, the subscription size distribution and several term occurrence distributions.

As future work, we propose to introduce a filtering system to reduce the number of delivered items to users. Based on the history of the item notified for each subscription, we want to filter out incoming item that does not satisfy novelty and diversity criteria as introduced in [34-35].

References

[1] Hmedeh Z, Vouzoukidou N, Travers N, Christophides V, du Mouza C, Scholl M. Characterizing web syndication behavior and content. In *Proc. the 12th WISE*, Nov. 2011, pp.29-42.

[2] Pereira J, Fabret F, Llirbat F, Preotiuc-Pietro R, Ross K A, Shasha D. Publish/subscribe on the web at extreme speed. In *Proc. the 26th VLDB*, Sept. 2000, pp.627-630.

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

[4] Aguilera M K, Strom R E, Sturman D C, Astley M, Chandra T D. Matching events in a content-based subscription system. In *Proc. the 8th PODC*, Apr. 29-May 6, 1999, pp.53-61.

[5] Zobel J, Moffat A. Inverted files for text search engines. *ACM Computing Survey*, 2006, 38(2): Article No. 6.

[6] Knuth D E. The Art of Computer Programming, Volume III: Sorting and Searching (2nd edition). Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[7] Yan T W, Garcia-Molina H. Index structures for selective dissemination of information under the Boolean model. *ACM Transactions on Database Systems*, 1994, 19(2): 332-364.

[8] König A C, Church K W, Markov M. A data structure for sponsored search. In *Proc. the 25th ICDE*, Mar. 29-April 2, 2009, pp.90-101.

[9] Bodon F. Surprising results of trie-based FIM algorithms. In *Proc. IEEE CIDM Workshop on FIMI*, Nov. 2004.

[10] Malik H H, Kender J R. Optimizing frequency queries for data mining applications. In *Proc. the 7th ICDM*, Oct. 2007, pp.595-600.

[11] Travers N, Hmedeh Z, Vouzoukidou N, du Mouza C, Christophides V, Scholl M. RSS feeds behavior analysis, structure and vocabulary. *International Journal of Web Information Systems*, 2014, 10(3): 291-320.

[12] Yan T W, Garcia-Molina H. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 1999, 24(4): 529-565.

[13] Bodon F. A trie-based APRIORI implementation for mining frequent item sequences. In *Proc. the 1st Int. Work. Open Source Data Mining* (*OSDM*), Aug. 2005, pp.56-65.

[14] Clément J, Flajolet P, Vallée B. Dynamical sources in information theory: A general analysis of trie structures. *Algorithmica*, 2001, 29(1): 307-369.

[15] Baeza-Yates R A, Ribeiro-Neto B. Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[16] Salton G, Wong A, Yang C S. A vector space model for automatic indexing. *Communications of the ACM*, 1975, 18(11): 613-620.

[17] Bookstein A, Swanson D. Probabilistic models for automatic indexing. *J. Am. Soc. Inf. Sci.*, 1974, 25(5): 312-316.

[18] Bagwell P. Ideal hash trees. Technical Report LAMPREPORT-2001-001, Ecole Polytechnique Federal de Lausanne, Switzerland, 2001.

[19] Walker A J. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software*, 1977, 3(3): 253-256.

[20] Beitzel S M, Jensen E C, Chowdhury A, Grossman D, Frieder O. Hourly analysis of a very large topically categorized web query log. In *Proc. the 27th SIGIR*, Jul. 2004, pp.321-328.

[21] Carzaniga A, Wolf A. Forwarding in a content-based network. In *Proc. the 17th SIGCOMM*, Aug. 2003, pp.163-174.

[22] Kale S, Hazan E, Cao F, Singh J P. Analysis and algorithms for content-based event matching. In *Proc. the 25th Int. Conf. Distributed Computing Systems* (*ICDCS*) *Workshops*, Jun. 2005, pp.363-369.

[23] Wang B, Zhang W, Kitsuregawa M. UB-tree based efficient predicate index with dimension transform for pub/sub system. In *Proc. the 9th DASFAA*, Mar. 2004, pp.63-74.

[24] Machanavajjhala A, Vee E, Garofalakis M N, Shanmugasundaram J. Scalable ranked publish/subscribe. *PVLDB*, 2008, 1(1): 451-462.

[25] Sadoghi M, Jacobsen H A. BE-tree: An index structure to efficiently match Boolean expressions over high-dimensional discrete space. In *Proc. the 30th SIGMOD*, Jun. 2011, pp.637-648.

[26] Whang S, Garcia-Molina H, Brower C, Shanmugasundaram J, Vassilvitskii S, Vee E, Yerneni R. Indexing Boolean expressions. *PVLDB*, 2009, 2(1): 37-48.

[27] Sadoghi M, Jacobsen H A. Analysis and optimization for Boolean expression indexing. *ACM Transactions on Database Systems*, 2013, 38(2): Article No. 8.

[28] Sadoghi M, Jacobsen H A. Relevance matters: Capitalizing on less (top-k matching in publish/subscribe). In *Proc. the 28th ICDE*, Apr. 2012, pp.786-797.

[29] Petrovic M, Liu H, Jacobsen H A. G-ToPSS: Fast filtering of graph-based metadata. In *Proc. the 14th WWW*, May 2005, pp.539-547.

[30] Liu H, Petrovic M, Jacobsen H. Efficient filtering of RSS documents on computer cluster. Technical Report, MSRG, University of Toronto, Nov. 2007.

[31] Demers A J, Gehrke J, Hong M, Riedewald M, White W M. Towards expressive publish/subscribe systems. In *Proc. the 10th EDBT*, Mar. 2006, pp.627-644.

[32] Irmak U, Mihaylov S, Suel T, Ganguly S, Izmailov R. Efficient query subscription processing for prospective search engines. In *Proc. USENIX*, Jun. 2006, pp.375-380.

[33] Shraer A, Gurevich M, Fontoura M, Josifovski V. Top-k publish-subscribe for social annotation of news. *PVLDB*, 2013, 6(6): 385-396.

[34] Hmedeh Z, du Mouza C, Travers N. TDV-based filter for novelty and diversity in a real-time pub/sub system. In *Proc. the 19th IDEAS*, Jul. 2015, pp.136-145.

[35] Hmedeh Z, du Mouza C, Travers N. FiND: A real-time filtering by novelty and diversity for publish/subscribe systems. In *Proc. the 27th SSDBM*, June 29-July 1, 2015.

Zeinab Hmedeh was a lecturer at the University Paris Ouest Nanterre La Défense. She received her Ph.D. degree in computer science in 2013 from the Conservatoire National des Arts et Métiers (CNAM) and her Master's degree in computer science from the Lebanese University and the University Paul Sabatier in 2010. Her research interests lie in the filtering of the information published on the Web 2.0.

Harry Kourdounakis received his M.S. degree in computer science from the University of Crete in 2011. His master thesis is about subscription indexes for web syndication systems. He is currently a software engineer at Brainsoft.

Vassilis Christophides is a professor of computer science at the University of Crete. He has been recently appointed to an advanced research position at INRIA Paris-Rocquencourt. Previously, he worked as a distinguished scientist at Technicolor, R&I Center in Paris. He studied electrical engineering at the National Technical University of Athens (NTUA), Greece, in July 1988. He received his DEA in computer science from the University PARIS VI, in June 1992, and his Ph.D. degree from the Conservatoire National des Arts et Métiers (CNAM) of Paris, in October 1996. His main research interests include databases and Web information systems, as well as big data processing and analysis. He has published over 130 articles in high-quality international conferences, journals, and workshops. He has been scientific coordinator of a number of research projects funded by the European Union, the Greek State, and private foundations on the Semantic Web and Digital Preservation at the Institute of Computer Science of FORTH. He has received the 2004 SIGMOD Test of Time Award and the Best Paper Award at the 2nd and the 6th International Semantic Web Conference in 2003 and 2007 respectively. He served as general chair of the joint EDBT/ICDT Conference in 2014 at Athens and as area chair for the ICDE "Semi-structured, Web, and Linked Data Management" track in 2016 at Bali, Indonesia.

Cédric du Mouza is an associate professor in the database and information system team of the Conservatoire National des Arts et Métiers (CNAM) in Paris. He received his Ph.D. degree in computer science from the CNAM in 2005 and two M.S. degrees (University Pierre et Marie Curie-Paris VI and University of Manchester). He also holds an engineering diploma from the Institut d'Informatique d'Entreprise (ENSIIE). His research work in the CEDRIC Laboratory mainly focuses on the distributed representation, indexing and querying of multi-dimensional, plain-text and Web 2.0 data. He is author or co-author of research papers published in major journals or conferences (ICDE, EDBT CIKM, SSDBM, VLDBJ, DKE, etc.).

Michel Scholl graduated from École Nationale Supérieure des Telecommunications (ENST), Paris, in 1966, and got his Ph.D. degree from University of California, Los Angeles, in 1977, and Docteur d'état in computer science from Institut polytechnique de Grenoble (INPG), Grenoble, in 1985. Michel Scholl spent more than 25 years at INRIA, Rocquencourt. Since 1989, he has been a full professor in computer science at CNAM, Paris, where he heads the SIBD research team at CEDRIC Laboratory, and is a member of the Wisdom PPF that he created and headed in 2007. He has been prime contractor for more than 15 French and European research projects and supervised about 15 Ph.D. theses. He was vice-president of the ANR scientific committee (programme blanc and jeunes chercheurs) and expert of the French agency AERES. He co-authored two books and about 90 research papers. He has been a member of the program committee of major database conferences (ACM SIGMOD, VLDB, and ICDE) and received a Test of Time Award of ACM SIGMOD in 2004.

Nicolas Travers is an associate professor in the Vertigo team of the CEDRIC Laboratory, at Conservatoire National des Arts et Métiers (CNAM), Paris. He received his Ph.D. degree in computer science from University of Versailles-Saint-Quentin-En-Yvelines (UVSQ) in 2006. His main focuses are query optimization in databases: indexing technics, continuous filtering, data and query distribution, NoSQL databases, query languages, multi-query optimization, etc. His researches deal mainly with Web 2.0 data, especially with short content like RSS, Twitter or Pinterest. He is author or co-author of some research papers published in major journals or conferences (EDBT, CIKM, WISE, DASFAA, IDEAS, SSDBM, IJWIS, etc.).