A hash-Tree based algorithm for subset detection: analysis and experiments

Benjamin Nguyen and Serge Abiteboul
INRIA
Domaine de Voluceau-Rocquencourt
BP 105 - 78153 Le Chesnay FRANCE
Fax: (33) 1 39 63 56 84
Email: <firstname>.<lastname>@inria.fr

Abstract

In this paper, we present both an analysis, and experimental results of an efficient algorithm for subset detection. The data structure used in this algorithm is a hash-tree. The algorithm was used as the core of the subscription system in an XML data warehouse. This context sets the range of the parameters of the problem. Both the analysis, and experimental results concur to show that the system can process thousands of documents per second, on a data base of millions of subscriptions.
Keywords: Subscription, Subset Detection, Hash-Tree, Warehouse, Web.
This paper is submitted to Track A.

A hash-tree based algorithm for subset detection: analysis and experiments

Benjamin Nguyen and Serge Abiteboul
INRIA Rocquencourt
<firstname>.<lastname>@inria.fr

Abstract

In this paper, we present both an analysis, and experimental results of an efficient algorithm for subset detection. The data structure used in this algorithm is a hash-tree. The algorithm was used as the core of the subscription system in an XML data warehouse. This context sets the range of the parameters of the problem. Both t he analysis, and experimental results concur to show that the system can process thousands of documents per second, on a data base of millions of subscriptions.

Keywords: Subscription, Subset Detection, Hash-Tree, Warehouse, Web.
This paper is submitted to Track A.

1 Introduction

Context Webusers today are not only interested in finding pages of interest, they are also paying a growing attention to the *evolution* of these pages. Some systems, like the Internet Archive [Arc], simply take snapshots of pages at various times. More recent systems, such as *Niagara* [CDTW00, Nia], or *ConQuer* [LPT99] offer the possibility of asking continuous queries over the web. Other systems such as *Le Subscribe* [FLPS00] offer monitoring capabilities, or trigger systems [HCH$^+$99]. Our work took place in the context of the Xyleme project [Xyl, Xyl01], a data warehouse [Wid95] for XML [W3C, W3C98, Mic99, ABS00] data on the web. We studied and developed a monitoring system [NACP01]. The core of this monitoring system is a "complex event detection algorithm", whose performance is critical. In this paper, we study the average complexity of this algorithm. As we see next, the algorithm can be used in a much wider context.

General problem The problem we consider can be stated in the following general terms. Let Γ be a set of sets of atomic events. The elements are called atomic events and sets of elements are called complex events. For each set S of elements, we want to find which are the $C' \in \Gamma$ contained in S. We would like to do that with reasonable time and space costs for a wide range of the parameters of the problem that we will discuss further.

Monitoring in *Xyleme* As mentioned above, the present work was motivated by the development of a subscription system for Xyleme. For the purpose of the present paper, it suffices to say that Xyleme constantly crawls the web and stores some of the documents. Let us explain the monitoring in Xyleme with a simple example: A photographer is interested in being informed whenever a new Nikon camera appears in his favorite online catalog. This (informal) *complex event* can be broken into *atomic events* that can be detected when this particular online catalog is crawled (by the web crawler). Given the incoming document (the catalog that is being reloaded), the warehouse, that kept the version of the catalog when it was last loaded, determines if all the following conditions hold:

- The document's URL is `www.leshop.com/cameras/cat.xml`; *and*
- the document has a new Nikon camera inserted into it.

In Xyleme, users can submit *subscriptions*, i.e., disjunctions of *complex events*, along with some additional information[1]. Upon loading a document, the system detects which *complex events* are triggered. This is done in two steps. First, the system detects all the *atomic events* that the incoming document triggers. Then, it determines which complex events the incoming set of atomic events triggers. The first part, the detection of atomic events, was described in [NACP01], and we will not go into any further detail here. We focus on the algorithm that detects the *complex events*.

[1] Like how the user wishes to be informed of changes, when to send changes, what changes to extract...

What is important to consider is the context of this application. Typically, we have millions of atomic events, millions of complex events composed of a rather small (4-10) number of atomic events, and thousands of documents per second, that each raise roughly 50-100 atomic events. We analyze here the novel algorithm [NACP01] that, using little space, efficiently detects the complex events triggered by each incoming document. Indeed, the implementation of the algorithm described in [NACP01] is actually in production by Xyleme.

The algorithm The algorithm uses a structure called hash-tree. It is a tree where each node is a hash table pointing to the children of the node. We use some ordering of atomic events. Intuitively, during a pre-processing phase, each complex event in Γ is sorted and entered in the structure. To test a set S of elements, we sort it and we navigate through the structure. Because of hashing, the navigation is very fast.

Note that the data structure we use resembles hash-trees from data-mining [AS94]. In data mining terms, our problem can be stated as the detection of particular item-sets contained in given transactions. We are unaware of any thorough analysis of the kind presented here.

Contribution Our main contribution is an average case complexity analysis of the algorithm. It shows that, within some reasonable ranges of parameters of the problem, a given S can be processed on average in $\mathcal{O}(|S|^2)$. We also present experiments that confirm this theoretical result. Note that this shows, in particular, that (within this range) the complexity is on average independent of the size of the complex events and even of the number of atomic or complex events. We also discuss what happens for the algorithm outside this range.

The paper is organized as follows. In Section 2, we present the problem and in Section 3, the algorithm. The main contribution of the paper, the average case complexity analysis, is the topic of Section 4. Experiments are presented in Section 5.

2 The problem

We use the notation $\binom{N}{k}$ to represent the generalized binomial coefficient (the number of distinct ways to select k elements in a set of N elements):

$$\binom{N}{k} = \begin{cases} \frac{N!}{k!(N-k)!} & \text{if } N > k > 0 \\ 1 & \text{if } k = 0 \text{ or } k = N \\ 0 & \text{if } k < 0 \text{ or } k > N \end{cases}$$

We use lower case letters to represent elements, upper case letters to represent sets, and Greek capitals to represent sets of sets. N, M, h, ℓ represent integers. Let us formalize the problem:

- Let $A = \{a_i | i \in [1; N]\}$ be the finite set of *atomic events*, and $N = |A|$.
- We call *complex event* a subset of A. To simplify the complexity calculation, we will assume at times that all complex events have the same cardinality h. (We will consider the impact of the variations of h).
- Let $\Gamma = \{C_j \mid j \in [1; M]\}$ represent a finite set of complex events. (M is the number of complex events.) We will assume that $A = \cup\{C_j\}$. It is easy to see that $M < 2^N$ and if all complex events are of same length $M \leq \binom{N}{h}$. This is important to bear in mind[2].
- Let $S = \{s_p \mid p \in [1; \ell], s_p \in A\}$ represent a subset of A. The set S is called the set of *incoming atomic events*. Let $\ell = |S|$ represent the number of incoming atomic events.

Our problem is simple. Let A and Γ be fixed. For a given S, we seek to determine all the $C_j \subseteq S$, i.e., $\{j \mid C_j \subseteq S\}$. We want an efficient algorithm to solve this problem even when N and M are quite large, say order of millions.

Complexity of a "naïve" solution

We first consider the complexity as a function of M, h and ℓ. Let h and ℓ be fixed. The input of the algorithm is Γ and S, where each complex event in Γ consists of h atomic events and S consists of ℓ atomic events.

The output of our algorithm will be the set $R = \{j \mid C_j \subseteq S\}$. First, we do not consider any preprocessing. We suppose that $S \subseteq \cup\{C_j\}$. Let us consider the following (naïve) algorithm.

Algorithm For each set S of atomic events, Γ set of sets of atomic events, the result set R is $\mathit{Result}(S, \Gamma)$ where the (iterative) function *Result* is defined as follows:

[2] If the complex events are not all of the same length, and if the longest event has length h_{max} then $M < \sum_{i=1}^{h_{max}} \binom{N}{h} < 2^N$.

FUNCTION $Result(S,\Gamma)$

1. $R \leftarrow \emptyset$
2. FOR every set $C_j \in \Gamma$
   (a) DO
   (b) $B \leftarrow TRUE$
   (c) FOR every $a \in C_j$
      i. DO
      ii. LOOK UP a in S
      iii. IF it is NOT found
      iv. THEN $B \leftarrow FALSE$ ENDIF
      v. DONE
   (d) IF B = TRUE
   (e) THEN add j to R ENDIF
   (f) DONE
3. RETURN R

The lookup of an element in the list can be done in $\mathcal{O}(1)$, using a hash-table for instance. Therefore, the time complexity of this algorithm is $\mathcal{O}(M \times h)$.

Lemma 2.1 (Naïve complexity) $c_{naive} = \mathcal{O}(M \times h)$
□

Let us recall the figures given in the introduction: $\ell \approx 50, h \approx 4, M \approx 10^7$. This "naïve" solution involves a crucial dependency in M that we *want to avoid*. We will see that our algorithm is indeed *independent* of M and N in a broad enough spectrum to be used for our applications.

Main contribution of the paper We propose an algorithm, based on the use of hash-trees, whose average complexity, in a large range of values for N, M, ℓ, h is $\mathcal{O}(\ell^2)$. We present both the complexity calculation, and experimental results that confirm our calculations. Note that in the context we are interested in, h, ℓ are orders of magnitude smaller than M, thus the average complexity is much better that the naïve one and in particular, does not depend on M.

3 The algorithm

For the purpose of describing the algorithm, we assume that a complex event is a *sorted list* of atomic events[3]. The ordering on the atomic events is purely arbitrary. We also use a data structure in the spirit of hash trees.

[3] This can be trivially done in $\mathcal{O}(\ell \log \ell)$ so does not change the final complexity result.

The data structure

Observe Figure 1. The data structure is entered via table H. This is what we call the ground table. Each other table corresponds to a prefix of complex events, e.g., $H_{1,5}$ corresponds to the complex events starting with $a_1 a_5$. A mark C_j in the corner of a cell indicates that the word that lead to this cell corresponds to complex event C_j. More precisely, a mark C_k in an a_i cell of H indicates that the k-th complex event is the set consisting simply of a_i. Similarly, a mark C_k in an a_i cell of table $H_{i_1 \ldots i_k}$ indicates that the C_k complex event is $a_{i_1} ... a_{i_k} a_i$. For instance, if you look at Figure 1, you see that cell a_0 in table H is marked with the value C_0. This means that the complex event $C_0 = \{a_0\}$. In the same way, cell a_6 of table $H_{1,5}$ is marked with value C_{43}. This means $C_{43} = \{a_1, a_5, a_6\}$. However, cell a_5 in table H_1 is not marked. This means the set $\{a_1, a_5\}$ does not constitute a complex event.

We consider that each table is stored as a *hash-table*. This makes every look-up in the table cost $\mathcal{O}(1)$. It is noteworthy that the size of the structure is small enough to reside in main memory in our experiments. This size is less or equal to Mh. The upper limit is met in the case where no complex events share the same nonempty prefix. Finally, such a data-structure can easily be updated by adding or deleting complex events.

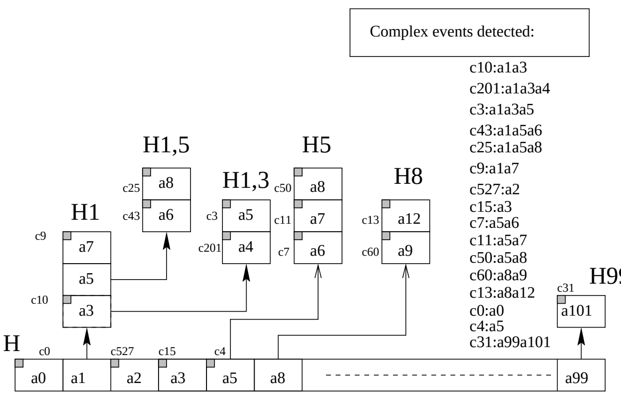


Figure 1: The data structure with complex events of variable length

Example

Let us demonstrate how the algorithm runs using an example. Suppose the document that is being processed has triggered the ordered atomic event set $S = \{a_1, a_3, a_5\}$. We first enter the data structure by $H[a_1]$, i.e., the first event of S. (Since the box is not marked, a_1 alone is not a complex event.) We proceed to H_1.

We find a_3. Since the box is marked, we detected the complex event $C_{10} = \{a_1, a_3\}$. Then we proceed to table $H_{1,3}$. We see that the cell containing a_5 is marked, this means we also detect the complex event $C_3 = \{a_1, a_3, a_5\}$. Back in table H_1 we find a_5, but the cell is not marked. Since we have no more atomic events left to process in this sub table, we exit it. We now reenter the data structure with a_3, to find $C_{15} = \{a_3\}$. Since this cell does not point to a sub table, we stop its processing here. Then we enter cell a_5 and detect the complex event $C_4 = \{a_5\}$. We have no more atomic events to process, so our task is over. The document triggered 4 complex events: C_{10}, C_3, C_{15} and C_4.

Presentation of the algorithm

For each table T (which is a hash-table) and each atomic event a_k, we denote by $T[a_k]$ the a_k entry of the table. Starting from the *ordered set* S of incoming atomic events, we produce the set of complex events that are triggered using the following algorithm:

Algorithm For each ordered set S of atomic events, the result set R is processed by*Notif*(H, S) where the (recursive) function *Notif* is defined as follows:

T is the H table or an H_ω table and

$a_{i_1}...a_{i_n}$ is an ordered set of atomic events.

FUNCTION *Notif*(T,$a_{i_1}...a_{i_n}$)

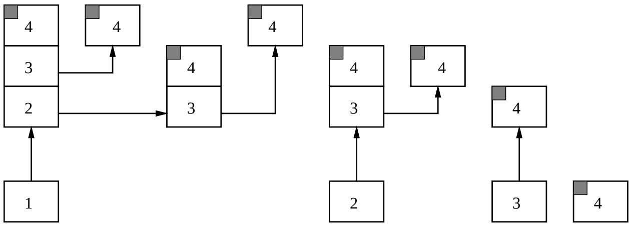
1. $R \leftarrow \emptyset$
2. FOR each k in $[1..n]$
   (a) DO
   (b) IF $T[a_{i_k}]$ is marked THEN add its mark to R ENDIF
   (c) IF $T[a_{i_k}]$ points to some table T' THEN add *Notif*(T', $a_{i_{k+1}}...a_{i_n}$) to R ENDIF
   (d) DONE
3. RETURN R

To obtain the complex events detected by a set S we simply call $Notif(H, S)$. Let us recall that since we use hash-tables, the costs of table lookups in 2-b, 2-c, is $\mathcal{O}(1)$.

We will briefly consider in the following paragraphs the worst case complexity of our algorithm, considering two parameters, the cardinality ℓ of S and h of each C_j.

Worst case complexity in ℓ

In the worst case, for every subset of S we look for, we find an entry in the table. Thus, the number of lookups in the table is $C_{wc_1} = \mathcal{O}(2^\ell)$. For example, this situation arises for the instance of the problem depicted in Figure 2.



S={1,2,3,4}

Figure 2: Worst case complexity example

Worst case complexity in h

The cardinality h of each C_j acts as a cut-off point for the worst case complexity. To obtain a complexity in $\mathcal{O}(2^\ell)$, we need to explore the data structure up to level ℓ. This supposes that $\ell < h$. On the contrary, if $h < \ell$ then we will stop at level h in the worst case. Thus:

$$C_{wc_2} = \mathcal{O}(\ell^h)$$

This is a more realistic bound in the context of the applications that motivated the present work, where $h \approx 5$ and $\ell \approx 100$.

4 Average case analysis

In this section, we study the influence of the length ℓ of the incoming subset S. To simplify the analysis, we suppose in what follows that all complex events are of length h. We show results on the average case time complexity of the algorithm.

4.1 Preamble

We assume that the ℓ atomic events in S have been uniformly randomly selected. Recall that the tables in the data structure are stored as hash-tables, and that S is stored as a list. Since a look up in a hash table is an $\mathcal{O}(1)$ action, the lookup time of a sublist of ℓ_i elements of S has a complexity $\mathcal{O}(\ell_i)$. The four critical parameters in the problem are N, M, ℓ, h. We found that a direct analysis based on N, M, ℓ, h was too complex. Instead, we present an analysis of the algorithm by focusing on the influence of the ℓ parameter. We will revisit the influence of N, M and h at the end of this section.

We assume that the time complexity of the algorithm is composed of the sum of the time spent making the look-ups in the hash tables.

The data structure we use can be seen as made of *levels*. The *ground* level consists of table H that we will call the *ground table*. The first level consists of all tables H_i. More precisely, we call *first level of* a_i the table H_i (that element a_i in table H points to). In the same way, we call *n-th level of* $a_{i1}..a_{in}$ the table $H_{i_1,\ldots,i_n}$. We will see that under rather general assumptions, the cost is dominated by the cost of computing up to the first level since further levels will not be reached most of the time.

We consider separately the cost function computations at each level. This motivates the following definition:

Notation 4.1 *Let N, M, h be fixed. The complexity cost function in ℓ is denoted $cost^{N,h,M}(\ell)$. The cost of processing events in the j-th level of the data-structure is denoted $cost_j^{N,h,M}(\ell)$; $cost_0^{N,h,M}(\ell)$ is the cost for level 0, the ground level. When N, h, M are understood, we simply use $cost(\ell)$ and $cost_j(\ell)$.*

Using this notation, $cost(\ell) = \sum_{j=0}^{\ell} cost_j(\ell)$. Observe that, $cost_0(\ell)$ represents the number of look-ups we will make in table H; $cost_1(\ell)$ represents the number of look-ups we will make in tables $H_1, H_2 \ldots H_{N-h}$; $cost_2(\ell)$ represents the number of look-ups we will make in tables[4] $H_{1,2}, H_{1,3} \ldots H_{X_1,X_2}$. As we will see later, some of these tables may not exist, so the corresponding look-up cost is null.

We use the following notation:

Notation 4.2 *Given h, N, the probability that one (random) complex event of size h starts with a_i is noted: $START_{N,h}(i)$.*

The proof of the following lemma can be found in Appendix A:

Lemma 4.3 *Given N, h, i, we have:*

$$\begin{cases} \textit{if } i \le N-h+1: & \\ \quad START_{N,h}(i) & = \frac{\binom{N-i}{h-1}}{\binom{N}{h}} \\ & = \frac{1}{(h-1)! \times \binom{N}{h}} \prod_{j=0}^{h-2} (N-j-i) \\ \textit{if } i > N-h+1: & \\ \quad START_{N,h}(i) & = 0 \end{cases}$$

and $START_{N,h}(i) = \mathcal{O}(\frac{h}{N}(1-\frac{i}{N})^{h-1})$

□

For an experimental illustratration of this result, see Figure 3(a). We plot on those figures on the Y-axis, the number of complex events that start with the atomic event i, where i is on the X-axis. The plot represents the experimental points, i.e. the exact number of complex events beginning with a given atomic event, after generating a sample M complex events. The full line curves represent the theoretical results. On the second row of figures, we plot on a logarithmic y-axis, for better readability. Note that, for $i > N - h$, there cannot be a complex event of length h that starts with a_i. Observe also that for some i, table H_i may be empty. This is the case if each complex event containing a_i also contains some a_j for $j \le i$. In the conditions of our application, this case does not occur often as we see next. We use the following notation:

Notation 4.4 (Existence of table H_i) *Let $LAYER^1_{N,h,M}(i)$ represent the probability that table H_i exists (i.e. is not empty), after randomly generating M complex events.*

For an analysis of the $LAYER$ function, see Appendix B.

To simplify the analysis of the problem, we will make the *pessimistic* following assumption:

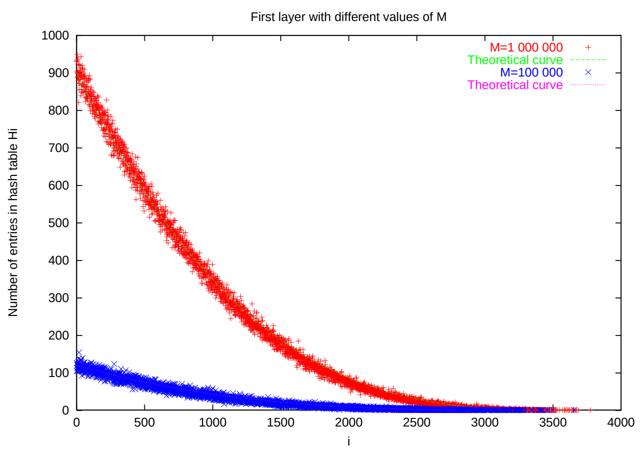
Assumption 4.5 (Full ground layer assumption) *For all $i \in [1, N-h], LAYER^1_{N,h,M}(i)$ is approximated by 1.*

This means we simply ignore the fact that certain entries in the ground table may be empty. This is a pessimistic assumption since in practical cases, some of the tables will be. If we find such a cell pointing to an empty table, the search with the cell as prefix will be over with at no extra cost. However, for the range of N and M we are interested in, the impact is negligible. See Figure 3(b) for values of N, M, h for which many H_i tables are empty. For instance, in the case of $N = 10000$ we see that most of the last 1000 tables are empty, but that the first ones contain at least one element.

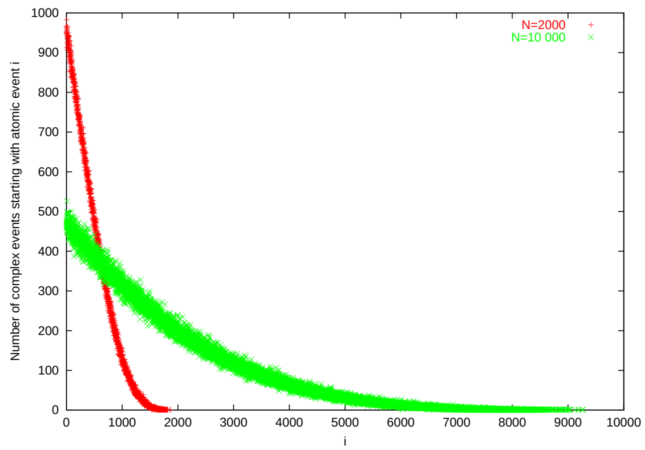
Computing up to the first layer

Let us first ignore the cost of further layers. We calculate the value of $cost_0(\ell) + cost_1(\ell)$. The operations that are executed are the following: For each element in the incoming list, we lookup in the first layer all the elements that have an index greater than it in the list, thus for the i-th element of the incoming list, we will

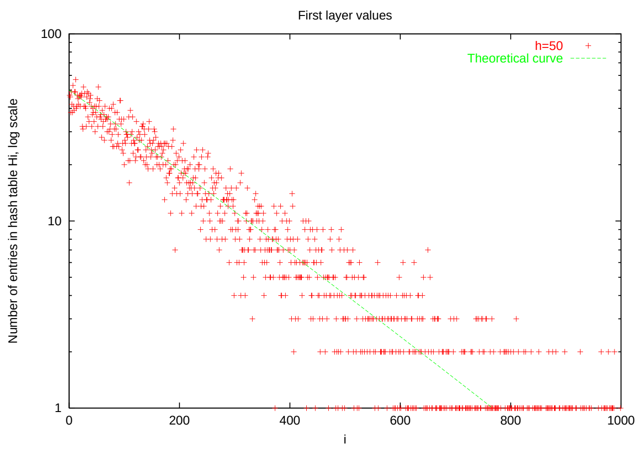
[4] We cannot enumerate all the tables that we are going to do lookups in, in a general way, since they depend on the complex events that are stored.



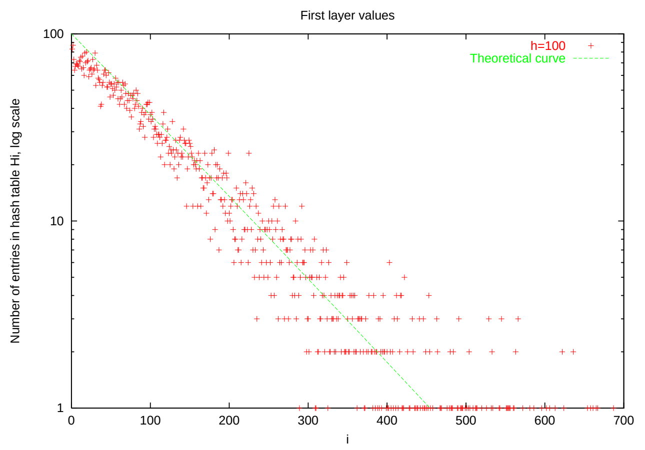
(a) Number of complex events beginning with atomic event a_i for N=4000, h=4, M=100 000, 1 000 000 000



(b) Number of complex events beginning with atomic event a_i for M=1000000,h=4,N=2000,10000



(c) Number of complex events beginning with atomic event a_i for N=10 000, h=50, M=10 000



(d) Number of complex events beginning with atomic event a_i for N=10 000, h=100, M=10 000

Figure 3: The experimental results show that the number of events in the first layer is indeed proportional to $M \times START_{N,h}(i)$. Figures 3(c),3(d) are presented with a logarithmic y-axis, to stress the polynomial fit, according to Lemma 4.3. Figure 3(a) presents the polynomial curve in the power $h-1$.

have to lookup $\ell - i$ elements in the first layer table. Assuming that the first layer is full, we have:

$$\begin{aligned}
cost_0(\ell) &\leq \ell \times \mathcal{O}(1) \\
cost_1(\ell) &\leq \sum_{j=1}^{\ell-1} j \times \mathcal{O}(1) \\
cost_0(\ell) + cost_1(\ell) &\leq \sum_{j=1}^{\ell} (1+j) \times \mathcal{O}(1) \\
cost_0(\ell) + cost_1(\ell) &= \mathcal{O}(\ell^2)
\end{aligned}$$

Computing further...

We will now show that, under reasonable conditions, the contribution of the higher levels is of the same order of cost. We will start with the case of the second layer. We will use the following notation:

Notation 4.6 (Existence of table $H_{i,j}$) *Let $LAYER^2_{N,H,M}(i,j)$ represent the probability that table $H_{i,j}$ exists, after randomly generating M complex events.*

The problem is that, of course, this probability is dependent not only of i, the index of the first atomic event, but also of j, the index of the second atomic event. Nevertheless, to calculate an upper bound on the complexity cost of the second layer, we only need to know *how many times* we will enter into *any* hash-table of the second layer. Therefore, we only need to know how many *different* atomic events appear in table H_i, to be able to determine how many times (on average) we are going to find an element in this table, and ultimately, how many times we will have to proceed to the second layer.

If we neglect (pessimistically) the fact that we can generate complex events that share the same first two atomic events, the number of events in hash-table H_i will be smaller than $START_{N,h}(i) \times M$. Therefore, the probability of finding a given element in the first layer will be equal to $LAYER^2_{N,h,M}(i) = \frac{START_{N,h}(i) \times M}{N-i}$. Since we assumed that all the tables H_i exist, for all the ℓ elements, we will try all the others greater than them (once again ℓ is clearly an upper bound), and if they are found in the table, pay an extra cost for the second layer[5], that is clearly $\mathcal{O}(\ell)$. Thus we have an upper bound for the cost of the second layer:

$$cost_2(\ell) \leq \ell^3 \times \frac{START_{N,h}(i) \times M}{N-i} \times \mathcal{O}(1)$$

[5]The cost to look up a maximum of ℓ elements in the next layer. Of course, there will in fact be less than ℓ elements, since at each level, all the elements in S that have an index smaller to the one processed will have been set aside

We prove in Appendix C the following lemma:

Lemma 4.7 $\forall (N,h,i) \in (\mathbb{N}^*)^3, if\ (h+i+1) < N,$ *then* $\frac{\binom{N-1}{h-1}}{\binom{N}{h} \times (N-1)} \geq \frac{\binom{N-i}{h-1}}{\binom{N}{h} \times (N-i)}$

□

This lemma shows that that an upper bound to $cost_2(\ell)$ can be given by

$$cost_2(\ell) \leq \ell^3 \times \frac{START_{N,h}(1) \times M}{N} \times \mathcal{O}(1)$$

To simplify our computation, let us assume the following:

Assumption 4.8 (Relative upper bound of M) $M \leq \frac{N^2}{2(\ell-2)h}$.

This is a realistic assumption since, for instance, for $N = 40\,000, h = 4, \ell = 20$, this yields $M < 10\,000\,000$, exactly the kind of bounds we need in our application. The interpretation can be seen as wanting to avoid the case depicted in Figure 2, where only a small number of atomic events constitute many complex events.

Under Assumption 4.8, we see that

$$\begin{aligned}
cost_2(\ell) &\leq \ell^2 \times START_{N,h}(1) \times \frac{N}{2h} \\
cost_2(\ell) &\leq \ell^2 \times \frac{\binom{N-1}{h-1}}{\binom{N}{h}} \times \frac{N}{2h} \\
cost_2(\ell) &\leq \ell^2 \times \frac{(N-1)! \times h! \times (N-h)!}{(h-1)! \times (N-h)! \times N!} \times \frac{N}{2h} \\
cost_2(\ell) &\leq \ell^2 \times \frac{1}{2}
\end{aligned}$$

We can conduct the same upper bound calculation on the higher levels of the data structure, and in a similar way, produce the following result:

Lemma 4.9 (Cost of layer j) *Under Assumption 4.8,*

$$\forall j \geq 1, cost_j(\ell) \leq \ell^2 \times (\frac{1}{2})^{j-1}$$

□

We see that this gives us an upper bound on the global complexity:

$$\begin{aligned}
cost(\ell) &\leq (\sum_{j=1}^{\ell} cost^j_{N,h,M}) + \ell \\
cost(\ell) &\leq \ell + \ell(\ell-1) \times \sum_{j=1}^{\ell} (\frac{1}{2})^j \\
cost(\ell) &\leq \ell + 2\ell(\ell-1)
\end{aligned}$$

And so we conclude our average case analysis with the following result:

Theorem 4.10 (Average case complexity) *Under Assumption 4.8,*

$$cost_{N,h,M}(\ell) = \mathcal{O}(\ell^2)$$

□

Experimental measures will confirm this result.

Remark 4.11 *(Parameter h) As shown in the previous section, if we consider h small, it does not come into account in the calculation of the average complexity, or the lower bound complexity, its only influence is in the worst case.*

Remark 4.12 *(Dependency on M and N.) Each complex event is constructed with atomic events, so given N, we have $N/h \leq M \leq \binom{N}{h}$. We will see experimentally that the algorithm indeed behaves in $\mathcal{O}(\ell^2)$ for a wide range of values. However, it behaves differently at the extremes:*

lower end *When M is much above the threshold given in our analysis (see Assumption 4.8 $M_{cutoff} > \frac{N^2}{2h\ell}$), the cost of the computation in deeper layers than the first one starts dominating. In such cases, we should expect the average complexity to approach the worst case complexity $C_{wc}(h)$. This will be confirmed experimentally.*

upper end *When M is very small, our analysis is too pessimistic. Most cells we examine are empty. At the limit, the complexity becomes on average independent of ℓ. This will be confirmed experimentally.*

5 Experimental results

Implementation We have implemented the algorithm in C++, using a hash-tree library we developed. It performs very well in the context of our application. As a matter of fact, it is now being used in production in the Xyleme system. We also performed some extensive tests. We would like to stress that the tests were performed on a very standard machine: a Linux PC with 700MHz Pentium III processor and 1GB of RAM.

Random complex event generation We briefly describe the construction of complex events for our tests.

We start by fixing N_0. This represents the universe from which we will be able to select atomic events. We then construct M distinct complex events by calling the following function: $Generate(N_0, M, h)$ given in Figure 4.

FUNCTION $GenerateComplex(N_0, h)$	FUNCTION $Generate(N_0, M, h)$
$C \leftarrow \emptyset$	$\Gamma \leftarrow \emptyset$
$i \leftarrow 0$	$j \leftarrow 0$
WHILE $i < h$ DO	WHILE $j < M$ DO
Randomly select a in $[1..N_0]$	$C \leftarrow GenerateComplex(N_0, h)$
IF $a \notin C$ THEN	IF $C \notin \Gamma$ THEN
$C \leftarrow C \cup \{a\}$	$\Gamma \leftarrow \Gamma \cup \{C\}$
$i \leftarrow i + 1$ ENDIF	$j \leftarrow j + 1$ ENDIF
DONE	DONE
RETURN C	RETURN Γ

Figure 4: Generation of events algorithm

We see that the algorithm simply generates complex events until it finds one that does not yet belong to the set that has been constructed so far. We stress that we do not control exactly the number N of atomic events used, some integers between 1 and N_0 may not have been selected. In the results we present in this section, we use the exact count of atomic events, i.e., N and not N_0. (During all the experiments, we selected a-priori N_0 so that $N_0 \approx N$.) These functions are used to generate the set M of complex events. To generate the incoming sets of atomic events, we also use the function *GenerateComplex*. To be precise, we also have to test that each atomic event appears in some complex one, i.e., is not in the $N_0 - N$ atomic that were not selected.

Let us discuss two simplifying assumptions:

equiprobability The analysis we made in the previous sections assumes that all atomic events have the same probability of appearing in a complex event. The random generation we use guarantees such a distribution. This has been assumed for the sake of simplicity. In practice, this is far from being the case. Some atomic events are much more likely to occur in complex events than others. It would be interesting to run experiments with a different random event generation, where we would for instance define atomic events by the percentage of complex events they appear in.

fixed length We assumed so far that all complex events are of the same length. We also ran tests with variable length complex events. First, we determined the length of a complex event, by picking a random value equally distributed between 1 and $2h - 1$. Then we randomly constructed a complex event of that length. The experimental results were analogous to the ones presented here.

As already mentioned, we have measured the number of complex events that begin with a given atomic event in a number of cases. Some such measures are given in Figures 3(a), 3(c), 3(b). We see that, as proved in Lemma 4.3, this number is proportional to $START_{N,h}(i)$.

Influence of ℓ As shown in Theorem 4.10, the complexity of the algorithm, given N and M within certain bounds, is $\mathcal{O}(\ell^2)$.

Param	Value	Asymptotic Std Err
a_2	2.2×10^{-4}	$\pm 4.7 \times 10^{-6}$ (2.1%)
a_1	4.5×10^{-1}	$\pm 4.6 \times 10^{-3}$ (1.0%)
a_0	5.6×10^{-1}	$\pm 6.5 \times 10^{-1}$ (115%)

Table 1: Results of the polynomial fit

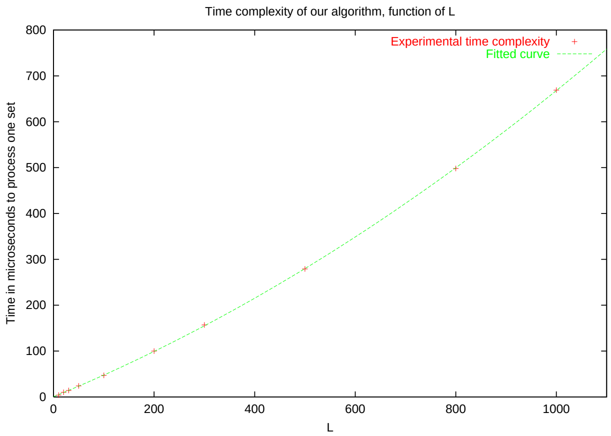


Figure 5: Time complexity function of ℓ, with $N = M = 10^6, h = 4$

The experimental results confirm our complexity calculations. We have conducted extensive tests, with values of M and N ranging from 1000 to 10^7, and they all exhibited the same parabolic behavior. To illustrate the experiments, we show in Figure 5 the variation of the computational time for the following values: $N = 10^6, M = 10^6, h = 4$ and ℓ ranging from 1 to 1000. Figure 5 and Table 1 show that the complexity is closely fitted by a curve of the form $f(\ell) = a_2\ell^2 + a_1\ell + a_0$. We plot the results using gnu-plot in Figure 5, and we provide the results of the fit in Table 1. We see that for an incomming set of 1000 atomic events, the time to process it is 0.5s, which means about 2000 documents per second, approximately 175 million documents per day. This can be compared to the rate of a single crawling machine, which is of a few million documents per day. This means *a single* complex event detection system can process the documents loaded by many different crawlers simultaneously.

Influence of parameters N and M Figures 6 show that for N and M ranging from 1000 to 1000000, their influence on the complexity of the algorithm is minimal. We have also conducted experimentations to see the influence on M and N in areas where our assumptions were not valid anymore. As previously mentioned, when $M >> N$, there are more and more chances of finding atomic events in lower layer tables, and we are close to the worst case scenario we discussed in Section 2. For $M = 10^6$, our analysis is correct for $N_{min} > 12000$. If we look at the zoom in Figure 6(b), we see that we in fact our analysis is correct for $N > 2000$. This confirms experimentally Lemma 4.3.

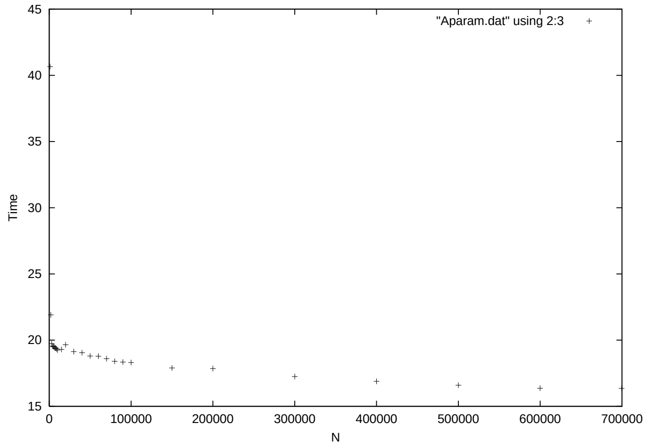
Figure 6(c) shows that for very few complex events compared to the number of different atomic events, then the performances of the algorithm are even better than the ones we expect. The high cutoff point for M is not shown in Figure 6(c), its approximate value is $M_{cutoff} = 10^{12}$. We can also note a 'low' cutoff point for M, for which the algorithm performs even better than expected. This value has not been calculated, but graphically we see it is around $M = 200000$. In between the high and low cutoff points, the time complexity is approximately constant (of M).

Influence of parameter h: We conducted two series of experiments:

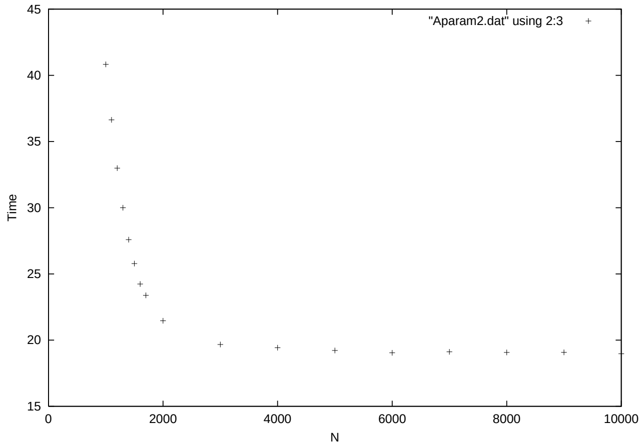
influence of the length of h Measurements made with different values of h showed that this parameter does not influence much the cost of the computation. This can be seen in Figure 6(d). (Note that for $h > \ell$, it is impossible to trigger a single complex event.)

variable length complex events We also tested this algorithm on a more "realistic" set of complex events, where all did not have the same length. Their length was randommed between 1 and $2h-1$. The experiments showed that the cost was roughly that obtained when all complex events have the same length h. We did not analyse this problem any further.

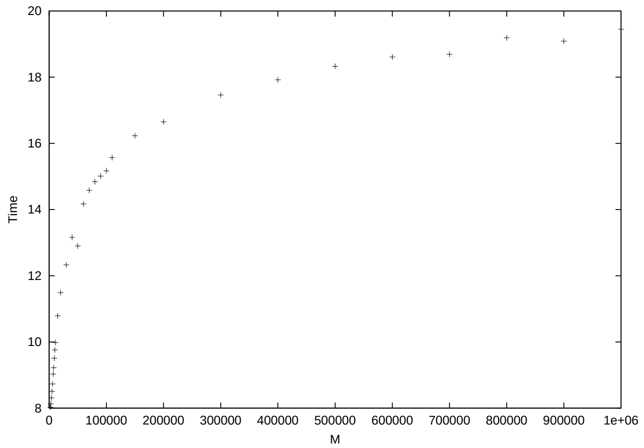
Influence of the number of complex events deteted: It can be argued that in real world applications the atomic events raised by a document are going to be correlated, and there is a greater chance to trigger at least one complex event for each document. We run



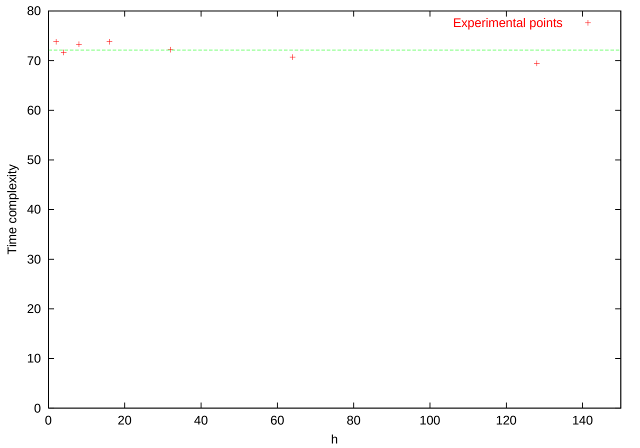
(a) Time complexity in microseconds, function of N for M=1000000,h=4,l=20



(b) Time complexity in microseconds function of N for M=1000000,h=4,l=20 (zoom)



(c) Time complexity in microseconds, function of M for N=100000,h=4,l=20



(d) Time complexity in microseconds, function of h, with M=100000, N=100000, l=50

Figure 6: Time complexity function of N, M and h

tests by skewing the elements of the incomming set of atomic events S by generating S in such a way that it contains at least x complex events. We ran tests with x ranging from 1 to 10 that give the same behaviour in $\mathcal{O}(\ell^2)$ as the non skewed ones.

6 Conclusion

The management of subscriptions in a Web warehouse motivated the study of an algorithm for subset detection whose average time cost would be independent of the number of candidate subsets. The context eliminated solutions that would require too large amounts of memory. The algorithm we presented and its implementation indeed have satisfying performances for our application. Moreover, our implementation is now being used in the Xyleme system.

Our main result is that, for a very wide range of values of M and N, our algorithm is in $\mathcal{O}(\ell^2)$. We also performed in-depth tests that we discussed here.

In this work, we assumed no a-priori knowledge on the atomic conditions. In practice, we know a lot about them. First, we may have knowledge about their distribution, and they are not equiprobable. For instance, suppose that a_1 is that the document is in English and a_2 that it contains the word "antidisestablishmentarianism". Clearly, a_1 is much more likely to happen. Note also that it is likely that every document raising a_2 would also raise a_1. By taking advantage of this knowledge, one could imagine ordering the atomic events to lower the computational cost.

The algorithm detects conjunctions of atomic conditions. It would be interesting to consider more complex conditions, *disjunctions* and *negations*. Moreover, in our setting, it would be interesting to study "temporal conditions" such as $a_1; a_2$ is raised by a document and later $a_3; a_4$ but with no a_5 occuring in between.

These are left for future research.

Acknowledgments: We would like to thank Mihaï Preda for his contribution to the algorithm, Luc Ségoufin, for his precious comments and discussion, Grégory Cobéna, Vincent Aguilera, Laurent Mignet, Cyril Nicaud and Claire Kenyon for helpful discussions on this topic.

A Behavior of $START_{N,h}(i)$

To calculate $START_{N,h}(i)$, we have to count the number of complex events P that start with a given atomic event a_i, and the total number of complex events Q. Clearly, $Q = \binom{N}{h}$ and P represents the number of ways to select $h-1$ elements in a set of $N-i$ elements, i.e., $P = \binom{N-i}{h-1}$. Hence: $START_{N,h}(i) = \frac{\binom{N-i}{h-1}}{\binom{N}{h}}$. This can be rewritten in the following formula:

$$START_{N,h}(i) = \frac{1}{(h-1)! \times \binom{N}{h}} \times \prod_{j=0}^{h-2}(N-j-i).$$

Therefore, the following relations hold:

$$\begin{array}{l} \forall h > 2, (i,N) \in \mathbb{N}^{*2}, i+h < N \\ \text{we have} \\ \prod_{j=0}^{h-2}(1-\frac{i}{N-j}) \quad < \quad (1-\frac{i}{N})^{h-1} \text{and} \\ \prod_{j=0}^{h-2}(N-j) \quad < \quad N^{h-1} \end{array}$$

We have:

$$\begin{aligned} START_{N,h}(i) &= \frac{\prod_{j=0}^{h-2}(N-j)}{(h-1)! \times \binom{N}{h}} \times \prod_{j=0}^{h-2}(1-\frac{i}{N-j}) \\ START_{N,h}(i) &= \frac{(\prod_{j=0}^{h-2}(N-j)) \times h!(N-h)!}{(h-1)!N!} \\ &\quad \times \prod_{j=0}^{h-2}(1-\frac{i}{N-j}) \\ START_{N,h}(i) &= \frac{h!(N-h)!}{(h-1)!(N-h+1)!} \times \prod_{j=0}^{h-2}(1-\frac{i}{N-j}) \\ START_{N,h}(i) &= \frac{h}{N-h+1} \times \prod_{j=0}^{h-2}(1-\frac{i}{N-j}) \end{aligned}$$

Since $h << N$, we can write the result simply as:

$$START_{N,h}(i) \quad = \quad \mathcal{O}(\frac{h}{N}(1-\frac{i}{N})^{h-1})$$

This proves Lemma 4.3.

B Existance of table H_i

In this appendix, we prove show that it is reaseonable to follow Assumption 4.5.

The exact value for $LAYER^1_{N,h,M}(i)$ The exact calculation for $LAYER^1_{N,h,M}(i)$, if we do not accept the possibility of generating twice the same atomic event is:

$$\begin{aligned} LAYER^1_{N,h,M}(i) &= \frac{\textit{hash trees of size } M \textit{ w/o table } H(i)}{\textit{total nr hash trees of size } M} \\ LAYER^1_{N,h,M}(i) &= \frac{(i-1)\binom{2^{N-1}}{M} + \binom{2^{N-i}}{M}}{\binom{2^N}{h}} \end{aligned}$$

This value is complicated to compute, this is why we use the approximate value that follows. If we do not exclude the possibility that some of these complex events may be identical, it is easy to see that: $LAYER^1_{N,h,M}(i) = 1 - (1 - START_{N,h}(i))^M$. We will now prove that the probability of generating the same complex events is negligeable, and that the formula just given can be approximated by 1 in most cases. In any case, by saying that the table exists, we are just making a pessimistic assumption.

Probability of generating the same complex events Of course, in our 'live' application, we will have exactly M *distinct* complex events. We argue here that given the figures of 'live' applications, the probability of generating two identical complex events is very small. Let us calculate this probability. The probability of generating the same complex event after having generated x distinct ones is: $P_{identical}(N,h,x) = \frac{x}{\binom{N}{h}}$. We can easily calculate the probability of generating M distinct complex events, it is:

$$
\begin{aligned}
P_{distinct}(N,h,M) &= \prod_{i=1}^{M-1} (1 - i/\binom{N}{h}) \\
P_{distinct}(N,h,M) &\geq (1 - \frac{M}{\binom{N}{h}})^M
\end{aligned}
$$

We give a numerical application: $N = 10^6, M = 10^7, h = 4$, with $\binom{N}{h} \approx N^h = 10^{24}$ thus $P_{distinct} \geq (1 - 10^{-17})^{10^7}$, $P_{distinct} \geq 1 - 10^{-10}$. This shows that neglecting the fact we can generate two identical complex events is a very pertinent assumption, given the range of values we have[6].

C Demonstration of Lemma 4.7

The demonstration is as follows. $\forall N, h, i \in (\mathbb{N}^*)^3, (h + i + 1) < N$ we have:

$$
\begin{aligned}
\frac{\binom{N-1}{h-1}}{N-1} &= \frac{(N-1)!}{(h-1)! \times (N-h)! \times (N-1)} \\
\frac{\binom{N-1}{h-1}}{N-1} &= \frac{(N-2)!}{(h-1)! \times (N-h)!}
\end{aligned}
$$

and

$$
\begin{aligned}
\frac{\binom{N-i}{h-1}}{N-i} &= \frac{(N-i)!}{(h-1)! \times (N-h+1-i)! \times (N-i)} \\
\frac{\binom{N-i}{h-1}}{N-i} &= \frac{(N-i-1)!}{(h-1)! \times (N-h+1-i)!}
\end{aligned}
$$

We see that:

$$
\frac{\binom{N-1}{h-1}}{N-1} = \frac{\binom{N-i}{h-1}}{N-i} \times \lambda
$$

where

$$
\begin{aligned}
\lambda &= \prod_{j=1}^{i} \frac{N-1-j}{N-1-h-j} \\
\lambda &\geq 1
\end{aligned}
$$

This proves Lemma 4.7

[6] This comes from the fact that with our values of N and h, $\binom{N}{h} = \mathcal{O}(\frac{N^h}{h!}) \approx N^h$, which in our case is $\Omega(M)$.

References

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web.* Morgan Kaufmann, California, 2000.

[Arc] The internet archive. http://www.archive.org/.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[CDTW00] Jianjun Chen, David DeWitt, Fend Tian, and Yuan Wang. Niagaracq: A scalable continous query system for the internet databases. *ACM SIGMOD*, page 379, 2000.

[FLPS00] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Publish/subscribe on the web at extreme speed. *Very Large Databses, VLDB*, 2000.

[HCH+99] Eric Hanson, Chris Carnes, Lan Huang, Mohan Konyala, Lloyd Noronha, Sashi Parthasarathy, J.B. Park, and Albert Vernon. Scalable trigger processing. *Proceedings of the 15th International Conference on Data Engineering*, pages 266–275, 1999.

[LPT99] Ling Liu, Calton Pu, and Wei Tang. Continual queries for internet scale event-driven information delivery. *IEEE TKDE*, 11(4):610, 1999.

[Mic99] Alain Michard. *XML, langage et applications.* Eyrolles, Paris, 1999.

[NACP01] Benjamin Nguyen, Serge Abiteboul, Gregory Cobena, and Mihai Preda. Monitoring XML data on the web. *SIGMOD*, 2001.

[Nia] Niagara webpage. http://www.cs.wisc.edu/niagara/.

[W3C] World Wide Web consortium page on XML. http://www.w3c.org/TR/REC-XML.

[W3C98] W3C. *eXtensible Markup Language (XML) 1.0*, february 1998.

[Wid95] Jennifer Widom. Research problems in data warehousing. *International Conference on Information and Knowledge Management (CIKM)*, 1995.

[Xyl] Xyleme home page. http://www.xyleme.com/.

[Xyl01] Lucie Xyleme. A Dynamic Warehouse for XML Data of the Web. *IEEE - Data Engineering Bulletin*, 24(2):40–47, June 2001.