The View Selection Problem for XML Content Based Routing

Ashish Kumar Gupta
University of Washington

Dan Suciu
University of Washington

Alon Y. Halevy
University of Washington

ABSTRACT

We consider the view selection problem for XML content based routing: given a network, in which a stream of XML documents is routed and the routing decisions are taken based on results of evaluating XPath predicates on these documents, select a set of views that maximize the throughput of the network. While in view selection for relational queries the speedup comes from eliminating joins, here the speedup is obtained from gaining direct access to data values in an XML packet, without parsing that packet. The views in our context can be seen as a binary representation of the XML document, tailored for the network's workload.

In this paper we define formally the view selection problem in the context of XML content based routing, and provide a practical solution for it. First, we formalize the problem; while the exact formulation is too complex to admit practical solutions, we show that it can be simplified to a manageable optimization problem, without loss in precision. Second we show that the simplified problem can be reduced to the Integer Cover problem. The Integer Cover problem is known to be NP-hard, and to admit a $\log n$ greedy approximation algorithm. Third, we show that the same greedy approximation algorithm performs much better on a class of workloads called 'hierarchical workloads', which are typical in XML stream processing. Namely, it returns an optimal solution for hierarchical workloads, and degrades gracefully to the $\log n$ general bound as the workload becomes less hierarchical.

1. INTRODUCTION

In *XML content based routing*, a stream of XML packets is sent from a set of data producers to a set of data consumers. Consumers subscribe to the data by means of predicates, or filters, typically expressed in XPath, and then receive a copy of all packets that satisfy those filters, independently of where the packets originated. Packets are delivered through a network of routers [1], or severs, in which each router receives and forwards XML packets, based on the values of some predicates stored at that server. This style of routing is called *content-based routing* [3, 17], because the packets are routed based on their content, and not based on any destination address.

There are several applications of XML packet routing. In *selective dissemination of information* (SDI) or publish/subscribe systems [8, 18, 19] users subscribe to text documents by specifying the predicate that the document should satisfy. Early SDI systems were based on the vector or the boolean model, but recently Franklin and Altinel [1] described an SDI system using XML and XPath. In reliable multicasting of data over mesh-based overlay network [17], XML content routing is used to facilitate intelligent content pruning. In *Web services*, applications can exchange XML messages in a style resembling remote procedure calls, typically using the SOAP protocol [11, 12]. All these applications share a common framework: XML documents from a continuous stream are checked against a workload of XPath predicates, and then forwarded to other servers or clients, sometimes in a distributed fashion.

The main performance metric in XML content routing is the global throughput of the system, or the number of XML packets that the system can deliver in a unit time. This can be improved in two ways: by improving the query evaluation at each server, and by improving the network configuration. In this paper we address the first method.

XPath query evaluation has been studied recently for large query workloads [1, 4, 6, 10, 14]. The state of the art in evaluating large XPath query workloads is that the complexity is dominated by the time needed to parse the incoming XML document. The fastest parsers today [5] achieve a throughput of about 10MB/s, while advanced query evaluation techniques surpass this [2]. The only possible way to further improve the throughput is to develop evaluation methods that avoid parsing.

We have recently proposed such a method called *view*

[1]These are not hardware level routers, but are instead routers at the application level.

selection for stream processing [13]. The idea is the following: select a small set of linear XPath expressions (*views*), let the data producers pre-compute the views for each XML packet that they generate, and attach them to the packet in the form of a *header*. When a server receives an XML packet, it tries to evaluate its workload by inspecting only the values in the header. If it succeeds, then it doesn't need to parse that packet: we call this case a *hit*. If it fails, then it needs to parse the entire packet and evaluate the query in a normal way: we call this a *miss*.

The header can be thought of as a binary representation of a subset of the XML document. A complete replacement of the XML document with a binary representation would increase query performance significantly, but would be platform dependent, preventing the universal interoperation that the XML standard enables. Our solution consists of adding only a small binary header to the standard text representation of the XML packet, allowing applications that do not recognize this format to ignore it. In our prior work [13] we have checked the feasibility of such a technique. We found that a *hit* takes about 100 times less time than a *miss*, making the technique attractive even if the hit ratio is, say, only 80% (then it results in a speedup by a factor of 5).

In this paper, we address the following problem: select a view configuration that optimizes the entire network's throughput. We call this the *View Configuration* (VC) problem. We present a sequence of results that, together, lead to a practical and efficient solution to the view configuration problem.

First, we formalize the problem and describe it as an optimization problem. In its exact formulation the problem is unmanageable. Our first result is to show that (VC) is equivalent, under certain conditions, to a *Simplified View Configuration* (SVC) problem. The latter has a much cleaner formulation, and the fact that the two are equivalent is rather surprising.

In our second result we show that the *header minimization* variant of the (SVC) problem is equivalent to the Integer Cover (IC) problem, that has been intensively studied in the past [7, 15, 9, 16], while the dual *throughput maximization* problem is a variant of the integer packing (IP) problem. Both problems are well-known to be NP-complete, but these reductions are nevertheless very important in practice given the rich literature on the (IC) problem (the (IP) problem has been less studied). In particular, Dobson [7] describes a simple greedy algorithm for (IC) that gives a solution no worse than $\log(1+n)$ times the optimal. We adapt this greedy algorithm to the header minimization variant of the (SVC) problem. In the case of workloads with constant selectivities, n is the largest number of XPath expressions in any query in the workload. Since n can be around $10-20$ in practice, the algorithm may generate header sizes that are four times larger than needed, which is unacceptable in practice.

Our third contribution consists in proving that the same greedy algorithm performs much better on constant-selectivity workloads that are *hierarchical*. We give a formal definition of hierarchical workloads, argue that they are typical for XML packet routing applications, and prove that the greedy algorithm is optimal in this case. Next, we study the behavior of this algorithm on workloads that are small deviations from hierarchical workloads. We introduce a parameter called the hierarchy measure of a workload, Δ, s.t. hierarchical workloads have $\Delta = 0$ and arbitrary workloads have $\Delta \leq n$. We show that the greedy algorithm always finds a solution within a factor $\ln(1+\Delta)$ of the optimal. This result generalizes both Dobson's theorem (the case when $\Delta = n$) and the result for hierarchical workloads ($\Delta = 0$).

The rest of the paper is organized as follows. In Sec. 2, we develop the background necessary for this problem, then we define the problem formally in Sec. 3. We show how to simplify the view configuration problem to a manageable version in Sec. 4. Next we show that the simpler problem can be reduced to the integer cover problem, in Sec. 5, and describe a greedy approximation algorithm. In Sec. 6 we prove better bounds for this algorithm in special cases. Finally we conclude in Sec. 7.

2. BACKGROUND

We overview here the problem definition from [13]. An *XML Content Routing Network* consists of a network of servers that evaluate and route a stream of XML documents, called *packets*. Each server has a local set of queries which it evaluates on every incoming packet. Based on the result of these queries, it decides to which neighbor or set of neighbors to forward the packet. Our goal is to optimize the total throughput of the network, i.e. the number of packets it can process in a unit of time, by improving the query processing time at each server. We discuss next various aspects of our problem.

The network We denote with S the set of servers, and we assume that all XML packets originate outside of the network, at data producers that send the packets directly to some server. Given a server s, we denote with $t(s)$ the average time spent by s in evaluating its set of predicates on a single XML document. We also denote with g_s the average fraction of all XML packets produced that eventually reach the server s: since we assume that each server receives at least some packets, we have $0 < g_s \leq 1$. The numbers g_s, $s \in S$ either can be measured in the network, or can be computed from other statistics, e.g., the fraction of the packets that each server sends along each of its links. Let T be the inverse of the network's throughput. Then we have:

$$T = \max_{s \in S} g_s \times t(s)$$

Our goal is to minimize T.

The queries We assume queries to be conjunctive queries over the XML packet. For example, the following is such a query:

```
q = /news/agency/name/text() like "%Reuters%"
        and /news/company/text() = "IBM"
```

which checks whether the `agency`'s `name` contains the string `Reuters` and whether the `company` is `IBM`. To simplify the discussion we shall allow only very restricted XPath expressions, consisting only of tags and child navigation axes. This allows us to define a set $\mathcal{P} = \{p_1, \ldots, p_n\}$ of all XPath expressions occurring in all queries. We assume that any two path expressions, p_i and p_j, $i \neq j$ are independent, i.e. they refer to disjoint sets of XML nodes. A query q is a conjunction of atomic predicates, where an atomic predicate is of the form $p\ r\ c$, where $p \in \mathcal{P}$, r is a relational operator (e.g. $=, \leq, <, \geq, >$), and c is a constant.

The workload at a server s consists of a set of conjunctive queries, $Q_s = \{q_1, \ldots, q_{m_s}\}$, $m_s \geq 1$, and the server s must evaluate all queries in the workload on every incoming packet d. Based on prior work on XML stream processing [10] we make the simplifying assumption that $t(s)$ is constant, independent of the workload. The justification is that query processing time is dominated largely by the parsing time, and the latter depends only on the average document size, and not on the XPath workload. By choosing appropriate units for time, we set $t(s) = 1$, under the assumption that evaluation proceeds in a "standard" way, i.e., parsing followed by XPath evaluation: we will describe in a moment a technique under which $t(s) < 1$.

The views A *view*, v, is an XPath expression $v \in \mathcal{P}$ and a *view configuration* is an ordered set of views, $V \subseteq \mathcal{P}$. The method proposed in [13] to improve the network's throughput starts by choosing offline a *view configuration*, V, which is then made available to all servers in the network. During the network's normal operation, the data producers compute the views in V for each document d that they produce, and attach their results to d, in the form of a binary *header*, $H(d)$. The header contains byte offsets in the XML document for each XPath expression in the view, rather than the actual result, thus, if $k = |V|$, then the header size is $2k$ bytes[2]. When we need to evaluate a query containing $p \in V$, we fetch the byte offset for p from the header and read p's value from that offset in the packet.

Each server s that receives a packet d attempts to compute its workload by inspecting only the header $H(d)$, without parsing the packet. When it succeeds, we say it is a *hit*; when it is forced to parse the packet and revert to standard evaluation techniques, we call it a *miss*. We describe the evaluation, starting first with a single conjunctive query $q = (p_1\ r_1\ c_1) \wedge (p_2\ r_2\ c_2) \wedge \ldots$ and an XML packet d. The query is evaluated in "short-circuit", i.e. the evaluation stops once we find a conjunct $p_i\ r_i\ c_i$ which is false. First, we evaluate the conjuncts for which $p_i \in V$, and only then proceed to the other conjuncts if needed. If for some $p_i \in V$ the corresponding conjunct evaluates to false, or if the path expressions referred to by all conjuncts in the query are in V, then we have a hit; otherwise we have a miss. The entire workload Q_s is evaluated by evaluating each query separately: if all queries $q \in Q_s$ have a hit, then Q_s has a hit; if at least one query has a miss, then Q_s has a miss.

[2]We assume that packet size is limited to $2^{16} bytes$. So, at most 2 bytes are needed to represent an offset.

In prior work [13] we have discussed implementation techniques for the packets' headers, and have found experimentally that the processing time of a hit is about two orders of magnitude smaller than that of a miss. Since the latter is 1 (it is essentially the parsing time), the average processing time of a document $t(s)$ at a server s is given by the miss ratio at that server, denoted $M(V, s)$, i.e. the fraction of XML packets for which we have a miss.

To compute the miss ratio $M(V, s)$, we assume the following statistics to be known. For each XPath expression $p \in \mathcal{P}$ and each query q, $\sigma_{p,q} \in (0, 1]$ denotes the selectivity of the atomic predicate with path expression p in q. We may safely define $\sigma_{p,q} = 1$ when no atomic predicate in q refers to p. Furthermore, with each query q, we associate a set, which consists of the XPath expressions (neglecting the relational operators and the constants) that appear in that query. We will refer to this set using q, thus making $q \subseteq \mathcal{P}$, Whether q refers to this set of path expressions or to the query itself, will be clear from the context. Then, the miss ratio for a single query, $m(V, q)$, the miss ratio for the set of queries at server s, $M(V, s)$, the average time to process a packet at server s, and the inverse of the network's throughput are given by:

$$\begin{aligned}
m(V, q) &= \begin{cases} 0 & \text{if } q \subseteq V \\ \prod_{p \in V \cap q} \sigma_{p,q} & \text{otherwise} \end{cases} && (1) \\
M(V, s) &= 1 - \prod_{q \in Q_s} (1 - m(V, q)) && (2) \\
t(s) &= M(V, s) && (3) \\
T &= \max_{s \in S} g_s \times t(s) && (4)
\end{aligned}$$

The effectiveness Clearly, the larger the set V, the lower the miss ratio $m(V, q)$, and the lower the average processing time $t(s)$. But we cannot choose a large V because a large header increases the packet size, thus defeating the purpose of increasing the throughput. In some preliminary experiments, partially described in [13], we have found that even with a relatively small header size (10-15), optimally chosen views can result in miss ratios of about 5%-10%. This, in turn, results in large speedups over traditional processing (by factors of 10-20). Clearly, well designed view configuration can improve significantly the query processing speed at the severs in a network. The key problem that needs to be solved is how to choose that view configuration. This is the topic of our paper.

3. THE VIEW CONFIGURATION PROBLEM(VC)

We give here a formal definition of the view configuration problem.

3.1 Problem Definition

We model the view configuration problem abstractly as follows. We are given the following:

- A set of servers S.
- For each server s the number g_s representing the fraction of XML packets that reach the server s.
- For each server $s \in S$, a set Q_s. We denote $|Q_s| = m_s$, $\mathcal{Q} = \bigcup_s Q_s = \{q_1, q_2, \ldots, q_m\}$, and call the elements of $\mathcal{Q}$ *queries*.
- A set $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$, whose elements we call *path expressions*.
- A relation $\epsilon \subseteq \mathcal{P} \times \mathcal{Q}$. We write $p \in q$ whenever $(p, q) \in \epsilon$. When no confusion arises we write q for the set $\{p \mid p \in q\}$ and we write p for the set $\{q \mid p \in q\}$.
- For each $p \in q$ a number $\sigma_{p,q} \in (0, 1]$ called *selectivity*. When $p \notin q$, we set $\sigma_{p,q} = 1$.

Assuming these parameters to be given, we define two flavors of the view configuration problem:

Throughput Maximization (VC-T) Here the input is a number k, the header size, and we need to find a view configuration $V \subseteq \mathcal{P}$ s.t. $|V| \le k$ and T is minimal, where T is given by Equations (4), (1), (2), (3).

Header Size Minimization (VC-H) Here the input is a number T_0, the inverse of the required throughput, and we need to find a view configuration $V \subseteq \mathcal{P}$ s.t. $T \le T_0$ and $|V|$ is minimal, where T is given by Equations (4), (1), (2), (3).

There are obvious polynomial-time reductions between the two problems, based on binary search.

4. SIMPLIFYING THE PROBLEM

The view configuration problem defined in Sec. 3 is dauntingly complex. We show here how it can be reduced to a manageable problem, with minimal loss in precision. We define the *simplified view configuration* problem (SVC) by the following constraints:

$$m(V, q) = \prod_{p \in V \cap q} \sigma_{p,q} \quad (5)$$

$$M(V, s) = \max_{q \in Q_s} m(V, q) \quad (6)$$

$$t(s) = M(V, s)$$

$$T = \max_{s \in S} g_s \times t(s)$$

Notice that the last two equations have not changed: we have included them for readability. The only change is in the definition of $m(V, q)$ and $M(V, s)$. As before, we consider the two variants, the throughput optimization problem and the header size optimization problem.

Simplified Throughput Maximization (SVC-T) Here the input is a number k, the header size, and we need to find a view configuration $V \subseteq \mathcal{P}$ s.t. $|V| \le k$ and T is minimal, where T is given by Equations (4), (5),(6), (3).

Simplified Header Minimization (SVC-H) Here the input is a number T_0, the inverse of the required throughput, and we need to find a view configuration $V \subseteq \mathcal{P}$ s.t. $T \le T_0$ and $|V|$ is minimal, where T is given by Equations (4), (5), (6), (3).

We justify next why a good solution to (SVC) is also a good solution to (VC). Consider first Eq. (5) versus Eq. (1). The two equations for $m(V, q)$ are actually identical when $q \not\subseteq V$. We argue that this simplification is harmless for two reasons. The first is that often a view configuration will not cover completely query q, and then no error is introduced at all: the header is typically small, and, thus, V is expected to cover completely only very few queries. Second, even if some query is completely covered, the error thus introduced is $\prod_{p \in V \cap q} \sigma_{p,q}$, which is small when the selectivities $\sigma_{p,q}$ are small. While it is possible to quantify this error precisely, we will simplify our discussion in the sequel and assume that $q \not\subseteq V$ for every query q, hence the migration from Eq.(1) to (5) introduces no error.

The second simplification replaces Eq. (2) with Eq. (6), and seems much more ad hoc and mysterious. Surprisingly, we can show that it incurs no loss in precision, under certain assumptions. Given a VC problem, define its *gap*, G, to be the following:

$$\begin{aligned} G = \min\{ & \frac{g_s \times m(V, q)}{(g_{s'} \times m(V', q'))} \mid \\ & V, V' \text{ are view configurations} \\ & s, s' \in S, q \in Q_s, q' \in Q_{s'} \\ & \text{and } \frac{g_s \times m(V, q)}{(g_{s'} \times m(V', q'))} > 1\} \end{aligned}$$

The gap is obtained by enumerating all possible values of $g_s \times m(V, q)$, for all combinations of servers s, queries $q \in Q_s$ and view configurations V, computing all ratios between two such values, and retaining the smallest such ratio larger than 1. The gap is always well defined because the number of view configurations is finite. In general, the gap becomes very large when the selectivities $\sigma_{p,q}$ are small. For example, in the case of a VC problem with *constant query selectivities*, where there exists some number σ s.t. for all p, q either $\sigma_{p,q} = 1$ or $\sigma_{p,q} = \sigma$, and *constant server selectivities*, where there exists some constant g s.t. $g_s = g \,\forall\, s \in S$, then $G = 1/\sigma$. Indeed, each value $g_s \times m(V, q)$ is in this case equal to $g \times \sigma^k$, for some $k \ge 0$, hence all ratios greater than 1 are of the form $1/\sigma^n$, and the gap is $1/\sigma$.

We can prove that if the gap is large, then (VC) and (SVC) are equivalent. Recall that $m_s = |Q_s|$ is the number of queries at server s.

THEOREM 4.1. *Suppose that the gap satisfies $G > m_s$, for every server s, and any optimal solution V to the (SVC-T) satisfies $q \not\subseteq V$ for every query $q \in \mathcal{Q}$. Then there exists an optimal solution V to the (SVC-T) which is also an optimal solution to the (VC-T).*

PROOF. For a view configuration V we denote $T_{VC}(V)$ and $T_{SVC}(V)$ the values of T to the (VC) and (SVC) problems respectively. Similarly, $M_{VC}(V,s)$ and $M_{SVC}(V,s)$ denote the values of M under the two definitions, Eq. (2) and (6). We have the following:

LEMMA 4.2. *Give two servers s, s', and two view configurations V, V', such that $g_s \times M_{SVC}(V,s) < g_{s'} \times M_{SVC}(V',s')$, then $g_s \times M_{VC}(V,s) < g_{s'} \times M_{VC}(V',s')$.*

PROOF. Let $q_0 \in Q_s$ be the query that maximizes $m(V,q)$; in other words, $M_{SVC}(V,s) = \max_{q \in Q_s} m(V,q) = m(V,q_0)$. Similarly, let $q_0' \in S_{s'}$ be the query that maximizes $m(V',q')$. Since $g_s \times m(V,q_0) < g_{s'} \times m(V,q_0')$ we have $g_s \times m(V,q_0) \leq \frac{1}{G} \times g_{s'} \times m(V',q_0')$, where G is the gap. Then the following holds, where $m_s = |Q_s|$:

$$\begin{aligned}
& g_s \times M_{VC}(V,s) \\
&= g_s \times (1 - \prod_{q \in Q_s}(1 - m(V,q))) \\
&\leq g_s \times (1 - \prod_{q \in Q_s}(1 - m(V,q_0))) \\
&= g_s \times (1 - (1 - m(V,q_0))^{m_s}) \\
&\leq g_s \times (1 - 1 + m_s \times m(V,q_0)) \\
&= g_s \times m_s \times m(V,q_0) \\
&\leq g_{s'} \times m_s \times \frac{1}{G} \times m(V',q_0') \\
&< g_{s'} \times m(V',q_0') \\
&\leq g_{s'} \times M_{VC}(V',s')
\end{aligned}$$

The last inequality follows from the fact that $m(V',q) \leq M_{VC}(V',s')$, whenever $q \in Q_{s'}$. □

We now prove the theorem. Let V be an optimal solution to the (SVC), and V' be an optimal solution to the (VC). If $T_{SVC}(V) \geq T_{SVC}(V')$ then we are done, because V' is also an optimal solution to (SVC). So suppose $T_{SVC}(V) < T_{SVC}(V')$ and we will show that it leads to a contradiction. Let $s_0 \in S$ be the server that maximizes $g_s \times M_{VC}(V,s)$; it follows that $T_{VC}(V) = g_{s_0} \times M_{VC}(V,s_0)$. We show now that there exists some query $q_0 \in Q_{s_0}$ for which $g_{s_0} \times m(V,q_0) = T_{SVC}(V)$. Indeed, by definition, there exists some $s \in S$ and $q \in Q_s$ s.t. $T_{SVC}(V) = g_s \times M_{SVC}(V,s) = g_s \times m(V,q)$. Suppose now that $\forall q_0 \in Q_{s_0}$, $g_{s_0} \times m(V,q_0) < g_s \times m(V,q)$: this implies that $g_{s_0} \times M_{SVC}(V,s_0) < g_s \times M_{SVC}(V,s)$, and, by the lemma, that $g_{s_0} \times M_{VC}(V,s_0) < g_s \times M_{VC}(V,s)$, contradicting our definition of s_0. Hence $T_{SVC}(V) = g_{s_0} \times m(V,q_0)$. Now we turn to V', for which we assumed that $T_{SVC}(V) < T_{SVC}(V')$. This implies $g_{s_0} \times M_{SVC}(V,q_0) = T_{SVC}(V) < g_s \times M_{SVC}(V',s)$ for every sever s. Then, we apply the lemma and prove that $g_{s_0} \times M_{VC}(V,s_0) < g_s \times M_{VC}(V,s)$, for every server s. But this implies that $T_{VC}(V) < T_{VC}(V')$, contradicting the fact that V' is an optimal solution to (VC).

5. ALGORITHMS FOR SIMPLIFIED VIEW CONFIGURATION PROBLEM

5.1 The Simplified Header Size Minimization

We reduce (SVC-H) to the Integer Cover problem (IC), and then adapt Dobson's [7] greedy algorithm for (IC) to solve (SVC-H).

5.1.1 Reduction to Integer Cover

The Integer Cover (IC) problem is the following. Given an $m \times n$ matrix A and n-dimensional row vectors c, u and b, find $x \in \{0,1,2,\ldots,\}^n$ such that:

$$\begin{aligned}
Ax &\geq b \\
x &\leq u
\end{aligned}$$

and cx is maximized. (IC) has been extensively studied, see e.g. [7, 9, 15, 16]. We prove now that the header size minimization problem can be reduced to (IC).

THEOREM 5.1. *The problem (SVC-H) can be reduced to (IC) with $c = \vec{1}$ and $u = \vec{1}$.*

PROOF. Given a formulation of (SVC-H), let $n = |\mathcal{P}|$ and $m = |\mathcal{Q}| = \sum_{s \in S} |Q_s|$. Recall that we denoted the global workload by $\mathcal{Q} = \{q_1, \ldots, q_m\}$. Define g_i to be g_s where s is the site holding the query q_i (i.e. $q_i \in Q_s$). Identical queries belonging to two different sites are treated as two different queries. Then define:

$$\begin{array}{lcll}
a_{ij} & = & -\log \sigma_{p_j, q_i}, & i = 1, \ldots, m; \\
 & & & j = 1, \ldots, n \\
b_i & = & -\log T_0 + \log g_i & i = 1, \ldots, m \\
c_j & = & 1 & j = 1, \ldots, n \\
u_j & = & 1 & j = 1, \ldots, n
\end{array}$$

□

Intuitively, a_{ij} is the coverage that the path expression p_j can provide to query q_i, and b_i is the total coverage required by the query q_i. Setting c_j to 1 for each j, means that each path expression, if chosen to be a part of the view configuration, takes up same amount of space in the header. And setting u_j equal to 1 for each j means that no path expression can be chosen more than once. The view configuration selected is given by $\{p_j \mid x_j = 1\}$.

5.1.2 A Greedy Algorithm

Here we describe a greedy algorithm for (SVC-H). The greedy algorithm works by including, at each stage, the path expression $p_{j'}$ that provides maximum additional coverage. In the algorithm below, setting $x_{j'} = 1$ amounts to selecting the path expression $p_{j'}$ to be in the view configuration.

Algorithm 1 A Greedy Algorithm for Header Size Minimization

```
1:  for $j = 1 \rightarrow n$ do
2:      $x_j \leftarrow 0$
3:  end for
4:  for $i = 1 \rightarrow m$ do
5:      for $j = 1 \rightarrow n$ do
6:          $a_{ij} = -\log \sigma_{p_j,q_i}$
7:      end for
8:      $min_{a_i} \leftarrow \min_{1 \le j \le n, a_{ij} \neq 0} a_{ij}$
9:      $b_i = (-\log T_0 + \log g_i)/min_{a_i}$
10:     for $j = 1 \rightarrow n$ do
11:         $a_{ij} \leftarrow a_{ij}/min_{a_i}$
12:     end for
13:     if for each $j$, $a_{ij}$ is an integer then
14:         $b_i = \lceil b_i \rceil$
15:     end if
16: end for
17:
18: while $\mathbf{b} \neq \mathbf{0}$ do
19:     if for each $j$, $\sum_{i=1}^{m} a_{ij} < 1$ then
20:         for any row $i$ such that $a_{ij} \neq 0$ has not been
            scaled so far, scale row $i$ to $\delta$
21:     end if
22:     $j' \leftarrow j$ such that $\sum_{i=1}^{m} a_{ij}$ is maximum
23:     $x_{j'} \leftarrow 1$
24:     for $i = 1 \rightarrow m$ do
25:         $b_i \leftarrow b_i - a_{ij'}$
26:         for $j = 1 \rightarrow n$ do
27:             $a_{ij} \leftarrow \min(a_{ij}, b_i)$
28:         end for
29:         $a_{ij'} \leftarrow 0$
30:     end for
31: end while
```

Lines 6 and 9 perform the reduction from (SVC-H) to (IC). The rest is Dobson's algorithm [7], and we describe it next. The main loop is in lines 18-31. At each step it picks that $x_{j'}$ to set to 1 that maximizes the sum $\sum_{i=1}^{m} a_{ij}$. Then it "sets" $x_{j'}$ to 1, by decreasing the b_i's. Values of a_{ij} that are larger than b_i are reduced to b_i in line 27: this does not affect the algorithm's correctness, but is important for the proof of the theorem below.

Lines 10 to 15 handle the case when A and b are not integral. First all entries in A are normalized to be ≥ 1 in lines 10-11. Then, line 19, does the following. When the sum in a row i becomes < 1 then one can prove that all coefficients in that row, including b_i are equal: they are all substituted with some small number $\delta < 1/m$. Values in these rows will not contribute to maximizing $\sum_i a_{ij}$ until all rows have had their coefficients adjusted to δ. Again, this is important for the theorem below. Based on Dobson's theorem, we derive the following for the (SVC-H) problem. Recall that the function H is defined as $H(d) = 1 + 1/2 + 1/3 + \ldots + 1/d \approx \ln d$.

THEOREM 5.2. *[7] Algorithm 1 returns a solution to (SVC-H) that is within* $\max_{1 \le j \le n}\{\log(\Sigma_{i=1}^{m} a_{ij}) + 1 + H(d_j)\}$ *of the optimal solution, where* d_j *is the number of queries in which the path expression* p_j *occurs.*

5.2 The Simplified Throughput Maximization Problem

5.2.1 A Greedy Algorithm

While the header minimization problem corresponds to integer cover, the throughput maximization problem corresponds to the integer packing problem, for which no good approximation results exist. We introduce here a new greedy algorithm, which we will show to be optimal in a special case. The algorithm starts off by placing every path expression into the view configuration, and then it eliminates them one by one, until it is left with a view configuration of size k.

DEFINITION 5.3. *Given an instance of (SVC-T), let* $V \subseteq \mathcal{P}$, *then we define a vector* $cov(\mathcal{Q}, V) \in \mathbf{R}^m$, *called the coverage vector. The* i^{th} *component of this vector is given by* $cov(q_i, V) = -\log g_i - \sum_{p \in q_i \cap V} \log \sigma_{p,q_i}$, *where* $g_i = g_s$ *and* s *is site holding the query* q_i.

DEFINITION 5.4. *Given two vectors* $u, v \in \mathbf{R}^m$, *we say that* u *is lexicographically greater than* v *if there exist* $1 \le i \le m$ *such that (i)* $u_j = v_j, \forall j < i$ *and (ii)* $u_i > v_i$. *Two vectors are lexicographically equal if they are equal in all their components.*

Algorithm 2 A Greedy Algorithm for Throughput Maximization

```
1: $V = \mathcal{P}$, the set of all path expressions
2: while $|V| \geq k$ do
3:     $j' \leftarrow j$ such that $sort(cov(\mathcal{Q}, V - \{p_j\}))$ is lexico-
       graphically maximal
4:     $V \leftarrow V - \{p_{j'}\}$
5: end while
```

Line 3 is the greedy decision-making step. The function *sort* takes a vector and returns a vector whose components are sorted in increasing order. Note that, starting with an empty view configuration and adding path expressions, one by one, using the same greedy criterion as in line 3 of Algorithm 2 can result in a solution whose approximation ratio can be arbitrarily bad. This is illustrated by the following example:

Example 5.5 There is a single server s with $g_s = 1$, and with workload consisting of two queries $q_1 = \{p_1, p_2\}$ and $q_2 = \{p_1, p_3\}$, with $\sigma_{p_1,q_1} = \sigma_{p_1,q_2} = \sigma_0$ and $\sigma_{p_2,q_1} = \sigma_{p_3,q_2} = \sigma_1$ (which is less than σ_0). The optimal view configuration of size 2 in this case is $V = \{p_2, p_3\}$, which is also the solution returned by Algorithm 2. However, if instead we start with an empty view configuration and add path expressions to it, one by one, using the same greedy criterion as the one in line 3 of Algorithm 2, then the solution is $V' = \{p_1, p_2\}$. The miss ratio for V is

σ_1, whereas for V', it is σ_0, which is off by a factor of σ_0/σ_1, which can be made arbitrarily large by choosing σ_0 and σ_1 appropriately.

6. SPECIAL CASES

In this section, we consider the view configuration problem in some special cases.

6.1 Constant Query Selectivities

Recall from Sec. 4 that a (VC) problem has *constant query selectivities* if there exists some constant σ s.t. for all p, q, either $\sigma_{p,q} = 1$, or $\sigma_{p,q} = \sigma$. In this case lines 8-15 of the Algorithm 1 will normalize the matrix A to contain only 0's and 1's and then round the vector b to an integer.

For this special case Dobson [7] proves a better bound than the one in Theorem 5.2. When applied to the (SVC-H) problem this gives:

COROLLARY 6.1. *When the view configuration problem has constant selectivities, then Algorithm 1 returns a solution to (SVC-H) that is within $H(\max_{1\leq j\leq n}|p_j|)$ of the optimal solution. Here $\mathcal{P} = \{p_1, \ldots, p_n\}$ is the set of all path expressions, and $|p_j|$ denotes the number of queries containing path expression p_j, $j = 1, \ldots, n$.*

6.2 Hierarchical Workloads

DEFINITION 6.2. *Let $\Delta \geq 0$ be an integer. We say that a set of sets $\mathcal{P}$ is Δ-hierarchical if for every $p, p' \in \mathcal{P}$, one of the following four conditions hold:*

- *$p \subseteq p'$, or*
- *$p' \subseteq p$, or*
- *$|p \cap p'| = 0$ or*
- *$|p| \leq \Delta$, $|p'| \leq \Delta$.*

When $\Delta = 0$, we call $\mathcal{P}$ a hierarchical set.

To see an example of a hierarchical workload consider:

```
q1:  /news/region="Europe" and
     /news/body/soccer/player="Zidane"
q2:  /news/region="Europe" and
     /news/body/soccer/player="Beckham"
q3:  /news/region="USA" and
     /news/body/baseball/inning/number="4"
```

All three queries start by inspecting the path expression `/news/region` and, depending on its value go on to inspect other paths. There are three path expressions, $p_1 =$ `/news/region`, $p_2 =$ `/news/body/soccer/player`, and $p_3 =$ `/news/body/baseball/inning/number`. Thus, $\mathcal{P} = \{p_1, p_2, p_3\}$, and the corresponding sets are: $p_1 = \{q_1, q_2, q_3\}$, $p_2 = \{q_1, q_2\}$, $p_3 = \{q_3\}$. Hence $\mathcal{P}$ is hierarchical by the definition above. Such workloads are typical in XML routing.

We associate the following DAG to a set of distinct sets $\mathcal{P}$: the nodes are the sets $p \in \mathcal{P}$ and the edges are (p, p') s.t. $p \subset p'$ and there is no other set "between" p and p'. When $\mathcal{P}$ contains duplicate sets, ties are broken arbitrarily. When $\mathcal{P}$ is hierarchical, this DAG is a forest, which we denote $forest(\mathcal{P})$. When $\mathcal{P}$ is Δ-hierarchical then this DAG has the following structure. Define $\mathcal{P}_a = \{p \mid p \in \mathcal{P}, |p| > \Delta\}$ and $\mathcal{P}_b = \mathcal{P} - \mathcal{P}_a$. Then $\mathcal{P}_a$ is hierarchical and its DAG is a forest, while all nodes in the DAG $\mathcal{P}_b$ are below those in $\mathcal{P}_a$.

The parameter Δ allows us to measure the degree by which a workload $\mathcal{P}$ of path expressions deviates from being hierarchical. In particular, if m is the total number of queries in a workload, then any $\mathcal{P}$ is Δ-hierarchical for $\Delta = m$.

DEFINITION 6.3. *Given a forest, we define a set of nodes to be a $prefix$ of the forest if whenever a node belongs to that set then its parent node (if any) also belongs to that set.*

Header Minimization We first analyze the greedy Algorithm 1 on the header minimization problem for hierarchical and Δ-hierarchical workloads.

THEOREM 6.4. *Let $\mathcal{P}$ denote the set of all expressions occurring in (SVC-H). If $\mathcal{P}$ is hierarchical, and if the queries have constant selectivities, then Algorithm 1 computes an optimal solution to (SVC-H).*

PROOF. We first need to prove the following two lemmas before we can proceed with the proof of the theorem:

LEMMA 6.5. *For a hierarchical workload, there is always an optimal solution to (SVC-H), which is a prefix of the $forest(\mathcal{P})$.*

PROOF. Let V^* denote an optimal solution to (SVC-H). Let us assume that V^* is not a prefix of $forest(\mathcal{P})$. From this solution, we will construct another solution V' to (SVC-H) that is a prefix of $forest(\mathcal{P})$, and is no worse than the optimal solution.

Since V^* is not a prefix of $forest(\mathcal{P})$, it means that there exists $p_j \in V^*$ such that $p_{j'} = parent(p_j) \notin V^*$. Notice that $V' = V^* \cup \{p_{j'}\} - \{p_j\}$ is also a solution to the same instance of (SVC-H) because $p_{j'}$ is a parent of p_j in $forest(\mathcal{P})$, which implies that $p_j \subseteq p_{j'}$. So, every query that contains the path expression p_j, also contains the path expression $p_{j'}$. Thus p_j can be replaced by $p_{j'}$ in the view configuration without increasing the miss ratio for any query. Also, notice that V' is no worse than V^* because $|V'| = |V^*|$. Continuing in this manner, we can construct a view configuration that is optimal and is also a prefix of $forest(\mathcal{P})$. $\square$

LEMMA 6.6. *For a hierarchical workload, the view configuration picked by the Algorithm 1 is always a prefix of $forest(\mathcal{P})$.*

PROOF. The (IC) portion of the algorithm runs on a matrix A whose entries are always 0 and 1. Given two path expressions p_{j_1} and p_{j_2} s.t. $p_{j_1} \supset p_{j_2}$, the inequality $\sum_{i=1}^{m} a_{ij_1} > \sum_{i=1}^{m} a_{ij_2}$ will always hold, until x_{j_1} is set to 1. Hence, the solutions picked by the greedy algorithm form a prefix in the forest. $\square$

Let V' be the prefix of $forest(\mathcal{P})$ constructed from an optimal solution, and let V'' be the prefix of $forest(\mathcal{P})$ returned by Algorithm 1. To prove the theorem, we need to show that the $|V''| \leq |V'|$. Let us assume the contrary, i.e. $|V''| > |V'|$.

Observe that in any tree of $forest(\mathcal{P})$, a leaf node corresponds to the set of queries, which contains exactly the path expressions, encountered from the root of the tree to the leaf. So, for any query $q \in \mathcal{Q}$, and a prefix V of $forest(\mathcal{P})$, the set of path expressions $q \cap V$ consists of the first $|q \cap V|$ nodes on the path from root to the leaf corresponding to q. This means that there exists a query $q_i \in \mathcal{Q}$ such that $|V_i''| > |V_i'|$ where $V_i'' = q_i \cap V''$ and $V_i' = q_i \cap V'$. Otherwise, we would have $|V''| \leq |V'|$.

Since, V_i' and V_i'' consist of the first $|V_i'|$ and $|V_i''|$ nodes on the path from the root to the leaf corresponding to q_i, and $|V_i''| > |V_i'|$, we have $V_i' \subset V_i''$, which in turn means that there exists p_j such that $p_j \in V_i''$, but $p_j \notin V_i'$. From Lemma 6.6, it follows that before the greedy algorithm will pick p_j, it will pick every node in V_i' because every node in V_i' is an ancestor of p_j. However, from the definition of V_i' it follows that, once the greedy has picked the nodes in V_i', the constraints for all the queries containing the path expression p_j are satisfied, thus implying that p_j is never picked by the greedy. This in turn implies that $V_i'' = V_i'$, thus contradicting the assumption that $|V''| > |V'|$.

Deviations from hierarchical workloads are probably likely in practice, i.e. we may have Δ-hierarchical workloads for small $\Delta > 0$. Interestingly, the greedy algorithm degrades gracefully with the amount of deviations, as we show next.

THEOREM 6.7. *Let $\mathcal{P}$ denote the set of all path expressions occurring in (SVC-H). If $\mathcal{P}$ is Δ-hierarchical, for some $\Delta \geq 0$, and if all queries have constant selectivities, then Algorithm 1 returns a solution that is within a factor of* $\ln(1+\Delta)$ *of the optimal.*

PROOF. We prove a small lemma before we proceed with the proof of the theorem.

LEMMA 6.8. *Let $\mathcal{P}$, the set of all path expressions occurring in (SVC-H), be Δ-hierarchical for some Δ. Let V be the view configuration picked by Algorithm 1. There exists an optimal solution V^*, such that $V^* \cap \mathcal{P}_a = V \cap \mathcal{P}_a$ where $\mathcal{P}_a = \{p \mid p \in \mathcal{P}, |p| > \Delta\}$.*

PROOF. Let $p_{i_1}, p_{i_2}, \ldots,$ be the order in which Algorithm 1 selects the nodes in $V \cap \mathcal{P}_a$. Let p_{i_j} be the first node in this ordering which is not in $V^* \cap \mathcal{P}_a$. Since, $p_{i_j} \in V$, there exists $q \in p_{i_j}$ such that $p_{i_1}, p_{i_2}, \ldots, p_{i_{j-1}}$ could not cover the query q. So, there must exist another node $p'_{i_j} \in V^*$ such that $q \in p'_{i_j}$. Since p_{i_j} and p'_{i_j} both cover q, so clearly $|p_{i_j} \cap p'_{i_j}| \neq 0$. Also, since $p_{i_j} \in \mathcal{P}_a$, we know that $|p_{i_j}| > \Delta$. And since the greedy algorithm always picks a node before it picks any of its children, $p_{i_j} \subseteq p'_{i_j}$ is also not possible. Since $\mathcal{P}$ is Δ-hierarchical, the only possibility that remains is $p'_{i_j} \subseteq p_{i_j}$, i.e. p_{i_j} covers every query that is covered by p'_{i_j} So, we can replace p'_{i_j} in the optimal solution by p_{i_j} to get another optimal solution. Continuing in this manner we can construct an optimal solution $V^{*\prime}$ where $V \cap \mathcal{P}_a = V^{*\prime} \cap \mathcal{P}_a$. $\square$

Now we proceed with the proof of the theorem. For a workload with constant selectivities, the greedy algorithm runs on integral instances of A and b, and at each step picks a column with a maximal sum. In Algorithm 1, let r denote the iteration of the while loop when for the first time a column j is picked for which the sum is $\leq \Delta$. Before this step all columns picked correspond to path expressions in $\mathcal{P}_a$, and from Lemma 6.8, we know that there exists an optimal solution that will contain all these path expressions. After this step, all the remaining columns in A have a sum which is $\leq \Delta$, and by Dobson's theorem on integral (IC), the solution returned by the algorithm for the sub-problem that remains after the $(r-1)^{st}$ iteration will be within a factor of $\ln(1+\Delta)$ of the optimal for the same sub-problem. Thus the overall solution produced by Algorithm 1 is also within a factor of $\ln(1+\Delta)$.

Notice that both Theorem 5.2 and Theorem 6.4 are special cases of Theorem 6.7. The first is a special case when $\Delta = n$: any workload is a Δ-hierarchical workload in this case. The second is a special case when $\Delta = 0$.

These theorems cannot be extended to queries without constant selectivities, as shown by the example in Fig. 1. This is a hierarchical workload, with $p_1 = p_2 = \{q_1, \ldots, q_n\}$, and $p_i = \{q_{i-2}\}$ for $i = 2, \ldots, n+2$. The optimal solution here is the vector $x = (1, 1, 0, 0, \ldots, 0)$. But the greedy algorithm will first set $x_3 = 1$, which decreases b_1 to 0 and reduces the first row in A to $0, \ldots, 0$. At the next iteration it sets $x_4 = 1$, etc. The solution returned is $(0, 0, 1, 1, \ldots, 1)$, a factor of $n/2$ worse than the optimal. This is consistent with Dobson's theorem 5.2, where the dominating therm is $\max_j \log \sum_i a_{ij} = \log(2^n - \epsilon) \approx n$.

Throughput Maximization Next we analyze the greedy

$$\overset{A}{\begin{bmatrix} 2^{n-1} & 2^{n-1} & 2^n - \epsilon & 0 & 0 & 0 \\ 2^{n-2} & 2^{n-2} & 0 & 2^{n-1} - \epsilon & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots \\ 1 & 1 & 0 & 0 & 0 & 2 - \epsilon \end{bmatrix}} \quad \overset{b}{\begin{bmatrix} 2^n - \epsilon \\ 2^{n-1} - \epsilon \\ \dots \\ 2 - \epsilon \end{bmatrix}}$$

Figure 1: An example of a hierarchical workload on which the greedy algorithm performs no better than in the general case

Algorithm 2, for throughput maximization, on hierarchical workloads.

THEOREM 6.9. *Algorithm 2 computes an optimal solution to (SVC-T) when the workload is hierarchical and the queries have constant selectivities.*

We need to prove the following two lemmas before we can proceed with the proof of the theorem:

LEMMA 6.10. *For a hierarchical workload, there is always an optimal solution to (SVC-T), which is a prefix of $forest(\mathcal{P})$.*

PROOF. The proof is similar to the proof of Lemma 6.5. Once again, let V^* denote an optimal solution to (SVC-T), and let us assume that V^* is not a prefix of $forest(\mathcal{P})$. Since V^* is not a prefix of $forest(\mathcal{P})$, there exists $p_j \in V^*$ such that $p_{j'} = parent(p_j) \notin V^*$. Consider the view configuration given by $V' = V^* \cup \{p_{j'}\} - \{p_j\}$. For the queries that do not contain $p_{j'}$, the miss ratio remains unchanged. For the queries that contain, $p_{j'}$, we have two cases: (1) For any query q that also contain p_j, the miss ratio remains unchanged. (2) For the queries q that do not contain p_j, $m(V', q) = \sigma_0 \times m(V^*, q) \leq m(V^*, q)$. Hence, p_j can be replaced by $p_{j'}$ in the view configuration without increasing the miss ratio for any query. Continuing in this manner, we can construct a view configuration that is optimal and is a prefix of $forest(\mathcal{P})$. $\square$

LEMMA 6.11. *For a hierarchical workload, the view configuration picked by the Algorithm 2 is always a prefix of $forest(\mathcal{P})$.*

PROOF. The basic idea of the proof is similar to that of Lemma 6.6. We will prove that the greedy algorithm never picks a node before it picks all its children. Once we have proved this, then the statement of the lemma trivially follows from it.

Let us assume the contrary i.e. the greedy algorithm indeed picks a node before it has picked all its children. Let $p_{j'}$ be the first such node, and let p_j be one of its children that has not yet been picked. This implies that the iteration in which $p_{j'}$ is picked, $sort(cov(\mathcal{Q}, V - \{p_{j'}\}))$ is lexicographically greater than $sort(cov(\mathcal{Q}, V - \{p_j\}))$. This implies that for some query $q_i \in \mathcal{Q}$, $cov(q_i, V - \{p_{j'}\}) > cov(q_i, V - \{p_j\})$, which in turn implies that $-\log \sigma_{p_{j'}, q_i} < -\log \sigma_{p_j, q_i}$, i.e. $\sigma_{p_{j'}, q_i} > \sigma_{p_j, q_i}$. However, since $p_{j'}$ is a parent of p_j, we know that for all queries $q \in \mathcal{Q}$, $\sigma_{p_{j'}, q} \leq \sigma_{p_j, q}$. Hence we have a contradiction. $\square$

Now, we go ahead with the proof of the theorem. Let V' be the prefix of $forest(\mathcal{P})$ constructed from an optimal solution, and let V'' be the prefix of $forest(\mathcal{P})$ returned by Algorithm 2. And let us assume that the throughput with V'' is less than the throughput with V'. Let $q'' \in \mathcal{Q}$ be the query for which $cov(q'', V'')$ is minimum. This coupled with the fact that V' is optimal implies that $cov(q'', V') \geq cov(q'', V'')$. If $cov(q'', V') = cov(q'', V'')$, then we are done. If however, $cov(q'', V') > cov(q'', V'')$, then it implies that $|q'' \cap V'| > |q'' \cap V''|$. So there must exist a $p'' \in (q'' \cap V') - (q'' \cap V'')$. Let p'' be the last such path expression selected by the greedy algorithm. Recall the fact that for any query $q \in \mathcal{Q}$, and a prefix V of $forest(\mathcal{P})$, the set of path expressions $q \cap V$ consist of the first $|q \cap V|$ nodes on the path from the root to the leaf corresponding to q. We also know that $|V'| = |V''| = k$, there must exist a query q' such that $|q' \cap V'| < |q' \cap V''|$, i.e. $cov(q', V') < cov(q', V'')$. Again there must exist a $p' \in (q' \cap V'') - (q' \cap V')$.

Now consider the iteration of Algorithm 2 in which p'' was selected. In the same iteration, p' was also a candidate for being selected because p' ends up being retained in V''. Let V denote the set of path expressions, still a part of the view configuration when this iteration began. Observe that $p' \notin V'$, the optimal solution. So, $\forall q \in p'$, $cov(q, V - \{p'\})$ is at least equal to $min_{q \in Q} cov(q, V')$, i.e. the value of the optimal solution. This is because $\forall q \in p'$, $q \cap V' \subseteq V - \{p'\}$.

Also, from the choice of p'', it follows that picking p'' to throw out leaves query q'' with coverage, $cov(q, V - \{p''\})$, which is equal to the value of the greedy solution, which is less than the value of the optimal solution (by assumption). So, greedy picks p' to throw out, contradicting the fact that the view configuration produced by greedy contains p'.

7. CONCLUSION

We have described an application of the View Configuration problem to a novel setting, where the views are path expressions and the data consists of XML packets. Although the exact formulation of the View Configuration problem is too complex for practical purposes, we have shown that it can be reduced to the Integer Cover problem, without loss in precision, under generous assumptions. A greedy approximation algorithm for the Integer Cover problem can be hence applied to the View Configuration problem, to produce a solution that is within a factor of $\log n$ of the optimal. Finally we have shown that the same greedy algorithm results in much better solutions in the case of special *hierarchical* workloads.

Acknowledgments

Suciu was partially supported by the NSF CAREER Grant IIS-0092955, NSF Grant IIS-0140493, a gift from Microsoft, and a Sloan Fellowship. Gupta was partially supported by the NSF CAREER Grant IIS-0092955. Halevy was supported by NSF CAREER Grant IIS-9985114, NSF ITR Grant IIS-0205635 and a gift from Microsoft Research.

8. REFERENCES

[1] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 53–64, 2000.

[2] I. Avila-Campillo, T. J. Green, A. Gupta, M. Onizuka, D. Raven, and D. Suciu. XMLTK: An XML toolkit for scalable XML stream processing. In *Proceedings of PLANX*, October 2002.

[3] A. Carzaniga and A. L. Wolf. Content-based networking: A new communication infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, 2001.

[4] C. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. In *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[5] J. Clark. XP - an XML parser in java. available from `http://www.jclark.com/xml/xp/index.html`.

[6] Y. Diao, P. Fischer, M. Franklin, and R. To. Yfilter: Efficient and scalable filtering of XML documents. In *Proceedings of the International Conference on Data Engineering*, San Jose, California, February 2002.

[7] G. Dobson. Worst-case analysis of greedy heuristics for integer programming with non-negative data. *Mathematics of Operational Research*, 7(4):515–531, 1982.

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

[9] T. Fujito. Approximation algorithm for submodular set cover with applications. *IEICE Trans. Inf. & Syst.*, E83-D(3):480–487, March 2000.

[10] T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata. In *Proceedings of ICDT*, 2003.

[11] M. Gudgin, M. Hadley, J. Moreau, and H. Nielsen. SOAP version 1.2 part 1: Messaging framework, 2001. available from the W3C, `http://www.w3.org/2000/xp/Group/`.

[12] M. Gudgin, M. Hadley, J. Moreau, and H. Nielsen. SOAP version 1.2 part 2: Adjuncts, 2001. available from the W3C, `http://www.w3.org/2000/xp/Group/`.

[13] A. Gupta, A. Halevy, and D. Suciu. View selection for XML stream processing. In *WebDB*, 2002.

[14] A. Gupta and D. Suciu. Stream processing of XPath queries with predicates. In *Proceeding of ACM SIGMOD Conference on Management of Data*, 2003. To appear.

[15] D. S. Hochbaum. *Approximation Algorithm for NP-hard Problems*. PWS Publishing Company, 1997.

[16] S. G. Kolliopoulos and N. E. Young. Tight approximation results for general covering integer programs. In *IEEE Symposium on Foundations of Computer Science*, pages 522–528, 2001.

[17] A. Snoeren, K. Conley, and D. Gifford. Mesh-based content routing using XML. In *Proceedings of the 18th Symposium on Operating Systems Principles*, pages 160–173, 2001.

[18] T. W. Yan and H. Garca-Molina. Index structures for selective dissemination of information under the boolean model. *ACM Transactions on Database Systems (TODS)*, 19(2):332–364, 1994.

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