Web Monitoring 2.0: Crossing Streams to Satisfy Complex Data Needs

Haggai Roitman 1, Avigdor Gal 2, Louiqa Raschid 3

Technion - Israel Institute of Technology and the University of Maryland, College Park

1haggair@tx.technion.ac.il
2avigal@ie.technion.ac.il
3louiqa@umiacs.umd.edu

Abstract—Web Monitoring 2.0 supports the complex information needs of clients who probe information and generate mashups by integrating across multiple volatile streams. A proxy that aims at capturing multiple client profiles that are customized to each client will face a scalability challenge in trying to maximize the number of clients served while at the same time fully satisfying complex client needs. In this paper, we introduce an abstraction of complex execution intervals, a combination of time intervals and information streams, to capture complex client needs. Given some budgetary constraints (e.g., bandwidth), we present online algorithmic solutions for the problem of maximizing completeness. We introduce three heuristics for this problem and use an offline approximation as a baseline comparing the performance of the heuristics. We use an extensive set of experiments on real traces and synthetic data to show that heuristics that exploit knowledge of the complexity level of profiles dominate across multiple settings.

I. INTRODUCTION

Web Monitoring 2.0 extends Web 2.0 to create a personalized proxy based platform where users can satisfy their complex information monitoring and aggregation/mashup needs by polling multiple information-rich and volatile Web 2.0 data sources. Such a platform responds to the personalization needs of Web 2.0. Instead of simply filtering data streams as they are pushed from servers, a proxy actively decides when it needs to cross these streams using pull-based technology, to satisfy client customization. Example platforms include personalization portals that are available to users worldwide, e.g., iGoogle1 and MyYahoo2. They provide a single point of access for personalized information. Portals provide services for continuously refreshing user profiles and for integration via a mashup3 of data extracted from multiple heterogeneous data sources.

To illustrate the problem of monitoring complex user needs, consider the problem of a financial analyst who looks for arbitrage opportunities, as suggested in [1]: “Arbitrage is the practice of taking advantage of a price differential between two or more markets: a combination of matching deals are struck that capitalize upon the imbalance, the profit being the difference between the market prices.”4 A simple arbitrage monitoring example is illustrated in Figure 1, showing the change in the price of one stock in two different markets. To identify arbitrage opportunities, financial data should be collected from multiple markets. This data is volatile, changing frequently with changes to market prices. Also, the analyst’s data needs require that data from both markets will be available, with overlapping time reference. That means that both streams need to be monitored (crossed) almost simultaneously. A complex profile in this case cannot be satisfied by the monitoring of a single market price. Rather, some (simple or complex) combination of monitoring tasks are needed.

This example illustrates the requirements of Web 2.0 Monitoring. Proxies cross multiple streams of events and probing must be performed in a timely manner to satisfy both the characteristics of the servers, e.g., intensity of updates, as well as the customization needs of clients. Supporting complex information needs requires a proxy to recognize dependencies among probes across resources. The proxy will face a scalability challenge when trying to satisfy the customized complex needs of millions of clients.

Push-based solutions to satisfy complex user needs exist yet data collection requires a high cost [2], especially if such push-based technology is not natural to the Web environment. Pull-based solutions have considered only simple monitoring solutions (e.g., [3]) that cannot satisfy the complex user needs.

---

1http://www.google.com/ig
2http://cm.my.yahoo.com/
3See http://www.programmableweb.com/ for mashup examples.
4This definition is taken from Wikipedia, http://en.wikipedia.org/wiki/Arbitrage
of Web 2.0 applications mashing up multiple continuously-updated streams. Web 2.0 monitoring is also different from continuous query processing or adaptive processing on streams where the objective is to support complex query processing, e.g., aggregation queries on data streams. To the best of our knowledge, our research is the first to support a new generation of solutions to address complex Web 2.0 monitoring in (primarily) pull-only settings. In prior work [1], we summarized the complexity of the Web 2.0 monitoring problem but did not provide solutions.

Complex user needs are expressed as a complex execution interval (CEI) [1] comprising of multiple simple execution intervals [4]. A proxy has to probe the corresponding resources during these intervals in order to satisfy the profile. Figure 1 shows 5 such labeled CEIs (ovals), each of which includes 2 simple intervals (numbered 1 through 5), one on each of the stock market servers. The analyst is satisfied only if the proxy probes both servers and captures both intervals of each CEI.

Our novel solution defines three levels of heuristic solutions to the monitoring problem. One is a simple solution (S-EDF) and the other two (M-EDF and MRSF) take into account the complexity of a CEI; this is expressed by the rank or the number of intervals that must be captured to satisfy a CEI. We study the properties of these heuristics and compare them both analytically and empirically to a baseline of an offline approximate solution.

We use extensive simulation on real and synthetic data to study the performance of the online solutions. Our analysis shows that M-EDF and MRSF both utilize the monitoring budget well. They dominate the offline approximation and the best known simple policies. The online policies are shown to be scalable and their performance deteriorates gracefully with an increase of update intensity, an increase of the number of profiles, or with the introduction of noise. Experiments with a variety of settings indicate that the heuristics perform well for typical Web 2.0 scenarios, e.g., where there is a skew in preference towards more popular Web sources.

The rest of the paper is organized as follows. We start with a detailed motivating example in Section II. In Section III we present our model for complex profile monitoring and formally define the problem, followed by the monitoring solution in Section IV. We present experiments in Section V and in Section VI we describe the related work. Section VII concludes and provides directions for future work.

II. Motivation

We illustrate a personalized portal in Figure 2. A business analyst identifies data sources that fit her needs, e.g., the CNN Breaking News 5 website, CNN Money.com,6 and Mish’s Global Economic Trend Analysis 7 blog. The data sources can be specified as URLs of online Web feeds (e.g., RSS, Atom) as shown by the RSS symbols on the right of Figure 2. Alternatively, Web 2.0 Web Scraping8 technology allows her to delineate content that is to be extracted, as seen on the left of the figure. Most Web feeds are available via pull-only access protocols, e.g., via HTTP GET requests. However, some feeds may be pushed, with the appropriate registration, using proprietary technology (e.g., Google Alerts service9).

An example (Example 2) of a specific complex information need is as follows: she constructs a wide perspective by integrating data from multiple business and market data sources. She is interested in crossing the stream of CNN Breaking News and CNN Money.com when a periodic pull from Mish’s Global Economic Trend Analysis blog detects that a new post in the blog contains the word %oil%. This will be translated into the following three queries:

\[
q_1: \text{SELECT item AS F1} \\
\text{FROM feed(MishBlog)} \\
\text{WHEN EVERY 10 MINUTES AS T1} \\
\text{WITHIN T1+2 MINUTES}
\]

\[
q_2: \text{SELECT item AS F2} \\
\text{FROM feed(CNNBreakingNews)} \\
\text{WHEN F1 CONTAINS %oil%} \\
\text{WITHIN T1+10 MINUTES}
\]

\[
q_3: \text{SELECT item AS F3} \\
\text{FROM feed(CNNMoney.com)} \\
\text{WHEN F1 CONTAINS %oil%} \\
\text{WITHIN T1+10 MINUTES}
\]

The analyst is willing to accept a delay of up to two minutes in probing MishBlog and a delay of 10 minutes for the other two feeds. Further, she may get partial utility even when she only receives updates from CNN Breaking News. An example (Example 3) where the complex data need has to be completely satisfied is as follows:

\[
q_1: \text{SELECT item AS F1} \\
\text{FROM feed(StockExchange)} \\
\text{WHEN ON PUSH AS T1}
\]

\[
q_2: \text{SELECT item AS F2} \\
\text{FROM feed(FuturesExchange)} \\
\text{WITHIN T1+1 SECONDS}
\]

\[
q_3: \text{SELECT item AS F3} \\
\text{FROM feed(CurrencyExchange)} \\
\text{WITHIN T1+1 SECONDS}
\]

According to a study on Web feeds [5], 55% of Web feeds are updated hourly. Further, due to heavy workloads that may be imposed by client probes (especially on popular Web feed providers such as CNN), about 80% of the feeds have an average size smaller than 10 KB, suggesting that items are promptly removed from the feeds. These statistics on refresh frequency and volatility illustrate the challenge faced by a proxy in satisfying millions of complex profiles.

We note that we do not attempt to present a language to express complex user monitoring needs. While we use pseudo

---

5http://edition.cnn.com/
6http://money.cnn.com/
7http://globaleconomicanalysis.blogspot.com/
9www.google.com/alerts
continuous queries to illustrate, we expect that the Web 2.0 environment will generate many tools, e.g., Web scraping, to provide intuitive interfaces to clients.

III. Model and Problem Definition

Servers and clients share data through proxies. A server manages resources and can be probed (pull-based) by the proxy on behalf of its clients. Occasionally a server may push either an update or a notification of an update to the proxy. We assume that a proxy implements Web 2.0 monitoring by crossing streams of data; using a pull protocol (e.g., HTTP GET queries). Our focus is on the scheduling of proxy pull tasks to satisfy complex client information needs. We discuss the three building blocks of our model, namely client profiles, execution intervals, and schedules.

A. Profiles and complex execution intervals

The complex information needs of a client is specified as a profile stored at the proxy. A client profile is associated with a set of resources and complex execution intervals (CEI) [1]. An execution interval (EI) [4] defines periods of time during which the profile must be synchronized with the state of the resource at the server. Crossing multiple data streams is represented by combining individual EIs to construct CEIs, possibly over a set of resources. Each EI in a CEI should be monitored once for the CEI to be satisfied (or “captured”). We assume that the EIs of a CEI can be captured in any order and we consider AND semantics, i.e., all EIs of a CEI must be captured. This is equivalent to the conjunction joining operator in the Amit situation manager [6].

Formally, let \( R = \{r_1, r_2, ..., r_n\} \) be a set of \( n \) resources and let \( T = (T_1, T_2, ..., T_K) \) be an epoch with \( K \) chronons.\(^{10}\) We assume the proxy manages a set of client profiles \( P = \{p_1, p_2, ..., p_m\} \). A client profile \( p = \{\eta | \eta = \{I_1, I_2, ..., I_t\}\} \) is a collection of CEIs. A CEI \( \eta \) contains several EIs, where each EI \( I \) is associated with a resource \( r \in R \) and \( I \) contains a start and finish chronon \( I = [T_s, T_f]; T_s, T_f \in T; T_s \leq T_f \). We further denote by \(|I|\) the number of chronons in EI \( I \).

Profiles, CEIs, and EIs construct a hierarchy (as illustrated in Figure 3), in which a profile is a parent of its CEIs, and a CEI is a parent of its EIs. Two CEIs within the same profile are siblings, and two EIs within the same CEI are also siblings. We use the number of EIs in a CEI to model profile complexity. Therefore, we denote by \( rank(p) \) the maximal number of execution intervals in any CEI \( \eta \in p \) (\( rank(p) = \max_{\eta \in P} \{|\eta|\} \)), where \(|\eta|\) is the number of execution intervals in \( \eta \). The definition is easily extended to a set of profiles \( P \) as follows: \( rank(P) = \max_{P \in P} \{rank(p)\} \).

The beginning of an interval is determined by an update event at a resource or a temporal event (e.g., every ten minutes). In the case that the server will push the update event, or for a temporal event, the beginning of the interval

\[^{10}\] A chronon is an indivisible unit of time.
is deterministic. A proxy may need to predict an update event using an update model and stochastic modeling [7] and pull the update event. In this case, we also consider the effect of noise. The window (length) of the interval is determined with respect to the stream of update events, (e.g., update = overwrite), or as a temporal event (e.g., within five minutes of the beginning of the interval). For example, a profile for Web scraping over Web feeds requires that published items be collected before the server overwrites them; this is a stochastic event.

The CEIIs for Example 1 are illustrated in Figure 1. We use the continuous queries of Example 2 to illustrate how the CEIIs are generated in Figure 4. Probing the MishBlog feed every 10 minutes, with a possible slack of 2 minutes, is represented by the first set of CEIIs, labeled T1. We note that by registering to a Pub/Sub system, the proxy may be informed of updates to Mish’s blog. However, the proxy still has to cross the stream to get the updated blog. For item’s on Mish’s blog that contain the keyword oil, EIs will be scheduled to probe the other 2 resources, labeled T2 and T3, respectively. In this example, some CEIIs will only have a rank of 1 (when updates on Mish’s blog do not include oil) while others will have a rank of 3.

EIs of the same or different profiles may overlap in time. Suppose we consider a slightly modified profile of Example 2 where the MishBlog feed is probed every 3 minutes. As before, when keyword oil is found, the other two streams are probed. Figure 5 shows the EIs.

Overlapping intervals are interesting for two reasons. When intervals of different resources overlap (inter-resource overlap) they are all candidates for being simultaneously probed by the proxy. This can lead to congestion when the available probing budget is low. When the execution intervals associated with an identical resource overlap (intra-resource overlap), there is the potential to exploit this overlap in building a more efficient schedule. While delaying a probe of a resource could lead to more efficient schedules, delaying a probe could also result in eventually not capturing an EI, e.g., there is congestion based competition from an EI on another resource. This is discussed later when we present our solutions. The special case of no intra-resource overlap is of theoretical interest.

B. Schedules

A data delivery schedule \( S = \{s_{i,j}\}_{i=1,\ldots,n; j=1,\ldots,K} \) (\( n \) resources and \( K \) chronons) assigns \( s_{i,j} = 1 \) if resource \( r_i \in R \) should be monitored (probed) by the proxy at chronon \( T_j \in T \), else \( s_{i,j} = 0 \). We denote by \( S \) the set of all possible schedules.

We use an indicator \( \mathbb{I}(I, S) \) to indicate whether a schedule \( S \) successfully probes and captures (the state of some resource \( r_i \) during) the EI \( I \). Given a profile \( p \), a CEI \( \eta \in p \), and an EI \( I \in \eta \) that refers to resource \( r_i \in R \), we have the following:

\[
\mathbb{I}(I, S) = \begin{cases} 
1, & \exists T_j \in I : s_{i,j} = 1 \\
0, & \text{otherwise}
\end{cases}
\]

We extend the indicator to \( \mathbb{I}(I, S) \) to describe capturing a CEI as follows: Given a profile \( p \) and a CEI \( \eta \in p \), we say that \( \eta \) is captured by schedule \( S \in \mathbb{S} \) if \( \mathbb{I}(\eta, S) = \prod_{I \in \eta} \mathbb{I}(I, S) = 1 \).

C. The monitoring problem

We assume that the proxy has a limited amount of resources that can be consumed for the monitoring task of client profiles. In this paper we consider a constraint similar to the one used in prior works of Web Monitoring [3] and Web Crawlers [8], where at each chronon \( T_j \in T \) the proxy can monitor up to \( C_j \) resources. This constraint is represented by a budget vector \( \vec{C} = (C_1, C_2, \ldots, C_K) \).

Given a set of client profiles \( P = \{p_1, p_2, \ldots, p_m\} \), the proxy objective is to maximize the gained completeness, that is, to maximize the number of CEIs from \( P \) that are captured given the budget \( \vec{C} \). A CEI is successfully captured once all of its execution intervals are captured. Every CEI \( \eta \in p \) that is successfully captured by the proxy schedule (indicated by \( \mathbb{I}(\eta, S) = 1 \)) increases the gained completeness.

Given a schedule \( S \in \mathbb{S} \), the gained completeness (denoted \( \mathcal{GC} \) in short) from monitoring \( P \) during \( T \) according to \( S \) is calculated as follows (where \( |p| \) denotes the number of CEIs in profile \( p \)):

\[
\mathcal{GC}(P, T, S) = \frac{\sum_{p \in P} \sum_{\eta \in p} \mathbb{I}(\eta, S)}{\sum_{p \in P} |p|}
\] (1)

Formally, the monitoring problem is defined by the following constrained optimization problem.
**Problem 1 (Complex Monitoring):** Given a set of profiles $\mathcal{P}$ and an epoch $T$:

$$\begin{align*}
\text{maximize} & \quad gC(\mathcal{P}, T, S) \\
\text{subject to} & \quad \sum_{i=1}^{n} s_{i,j} \leq C_j, \quad \forall j = 1, 2, \ldots, K 
\end{align*}$$

Problem 1 is challenging, as we shall demonstrate in Section IV. Nevertheless, it can be extended in several interesting directions. First, we do not consider the varying costs of probing resources. These variations may be due to computational costs, e.g., extracting a stock price may be cheaper than searching for a keyword in a blog, the bandwidth needed to download results of varying size, monetary charges at the servers, etc.

In this case, each probe will not consume the same budget. We defer these extensions to future work.

**IV. MONITORING SOLUTIONS**

We now present our solution to the complex monitoring problem. We first describe the online solution. A proxy has to make a probing decision without complete knowledge of future CEIs that may need to be captured. If we consider the information needs of Example 2, while the EIs for probing the MishBlog feed can be determined in advance, the need to cross the other two streams, (CNN Breaking News and CNN Money.com) depends on the contents of the first probe.

The online proxy decision when all CEIs are not known a priori is as follows: At every chronon $T_j$, the proxy may receive a set of new CEIs. The proxy then has to decide which resources in $R$ to probe, while considering the set of all candidate CEIs, including those submitted prior to $T_j$, which have not been completely captured yet, and the new set of CEIs. We denote the set of all candidate CEIs at chronon $T_j$ as $\text{cands}(\eta)$ and the union bag of all their EIs (termed candidate EIs) as $\text{cands}(I) = \bigcup_{\eta_j \in \text{cands}(\eta)} \eta_j$. The bag notation $(\bigcup)$ is used due to intra-resource overlaps.

**A. Policies**

To determine which candidate EIs in $\text{cands}(I)$ to choose the proxy uses policies. At chronon $T_j$, a policy $\Phi$ considers $\text{cands}(I)$ and the budget $C_j$, and returns up to $C_j$ EIs to probe. Such policies can be efficiently implemented.

Each of the policies we propose can be executed in either a non-preemptive or preemptive manner. Non-preemptive policies do not allow new candidate CEIs to be scheduled for monitoring at chronon $T_j$ if previously probed CEIs need to be probed at $T_j$. Therefore, a non-preemptive policy $\Phi$ first selects $I \in \text{cands}(I)$ that belongs to previously probed CEIs. Then, if there is any budget left, $\Phi$ selects EIs from the newly introduced CEIs. It is worth noting that even the non-preemptive policies do not guarantee a successful capture of a CEI that has been probed at least once. If the number of previously probed CEIs exceeds the bandwidth budget, some of these CEIs may be dropped.

The detailed implementation of these policies and execution time complexity analysis is in [9]. Policies can be classified according to the amount of information needed about the candidate CEIs and their execution intervals. We propose a three level classification as follows:

**Individual EI level:** An individual EI level policy utilizes only the local properties of a single EI without considering the parent CEI or sibling EIs.

As a representative of this level we suggest the Single Interval Early Deadline First (or S-EDF in short) policy, $\Phi_{S-EDF}$. This policy is modeled on the well known EDF policy [10], preferring EIs that have the earliest deadline. Given an execution interval $I$ and a chronon $T$, the deadline is calculated (in terms of number of remaining chronons) as follows:

$$S-EDF(I, T) = I.T_f - T + 1$$

The following proposition holds for this policy.

**Proposition 1:** Given $\mathcal{P}$ without intra-resource overlap, and $\text{rank}(\mathcal{P}) = 1$, the policy $\Phi_{S-EDF}$ is optimal.

**Proof:** [Sketch] Postponing probes of execution intervals with higher S-EDF value at some chronon $T_j$ provides more opportunities to captured additional EIs at the next chronon $T_{j+1}$, thus the $\Phi_{S-EDF}$ is optimal. \hfill ■

WIC [3], a well-known monitoring solution for the Web, can also be classified as an individual EI level policy. We note that WIC was designed to address a different optimization goal compared to our Problem 1). WIC’s solution balances completeness with timeliness, providing a bound of 2-competitiveness for its optimization goal. WIC defines a utility for each EI and picks those EIs with the maximum accumulated utility for probing in each chronon. Our experiments in Section V implemented WIC (details provided later) and compared the performance of WIC with the online policies from this section.

**Rank level:** A rank level policy bases its decision on profile complexity by considering the rank of the parent CEI. As a representative of this level we suggest the Minimal Residual Stub First (MRSF) policy, $\Phi_{MRSF}$. This policy prefers EIs that belong to parent CEIs with a minimal number of EIs left to be captured. The intuition behind this policy is that a CEI with less EIs remaining to probe has a higher probability of success. Formally, given an EI $I$, $I \in \eta$ and $\eta \in \mathcal{P}$, then the MRSF value is calculated as follows:

$$\text{MRSF}(I) = \text{rank}(p) - \sum_{I' \in \eta} \mathbb{I}(I', S)$$

where $I'$ iterates over all EIs in $\eta$. The following proposition provides a bound for the performance of this policy.
Proposition 2: Given $\mathcal{P}$ without intra-resource overlap, the $\Phi_{MRSF}$ policy is $l$-competitive, where:

$$l = \max_{\eta \in \mathcal{P}} \left( \sum_{I \in \eta} |I| \right)$$

Proof: [Sketch] Let $\eta$ be an arbitrary CEI that was captured by $\Phi_{MRSF}$ policy given $\mathcal{P}$. Given such CEI $\eta$, it is easy to build a new problem instance $\mathcal{P}'$ where for every EI $I \in \eta$ and for every chronon $T_j \in I$ we place an additional CEI $\eta'$ that competes with $\eta$ (i.e., $\Phi_{MRSF}$ would choose arbitrary between $\eta$ or $\eta'$, where we assume that it chooses $\eta$) and in which $\eta'$ will get probes by $\Phi_{MRSF}$ but will never get fully captured since it is blocked by $\eta$. Further, for every such chronon $T_j$ we add to $\mathcal{P}'$ another CEI $\eta''$ that also competes with both $\eta$ and the corresponding $\eta'$, where each first EI $I_1 \in \eta''$ has a size of one chronon and the rest of its EIs start after any of CEI $\eta''$'s EIs. Since $\Phi_{MRSF}$ chooses either EIs of CEIs $\eta$ or $\eta'$, every opportunity to capture an CEI $\eta''$ and increase the completeness by one actually gets lost in $\mathcal{P}'$. Therefore, given $\eta$, with total chronons of $\sum_{I \in \eta} |I|$, $\Phi_{MRSF}$ will gain only 1 in completeness where it could have actually gained $\sum_{I \in \eta} |I|$. In the worst case all CEIs captured by $\Phi_{MRSF}$ have similar total number of chronons of $l = \sum_{I \in \eta} |I|$. Therefore, the worst case competitive ratio of $\Phi_{MRSF}$ is $l = \max_{\eta \in \mathcal{P}} \left( \sum_{I \in \eta} |I| \right)$.

Multi-EIs level: A multi-EIs level policy utilizes all information about execution intervals of a parent CEI (including sibling information). As a representative of this level we suggest the Multi Interval EDF ($M$-EDF) policy, $\Phi_{M}$-EDF, which prefers execution intervals that have the minimal $M$-EDF value, calculated as follows (for $I \in \eta$):

$$M$-EDF$(I, T) = \sum_{I' \in \eta} (S$-EDF$(I', T) \cdot [1 - \| I', S \|])$$

The $M$-EDF value combines the EDF values of execution interval $I$ and its siblings that were not captured up to chronon $T$, where for each execution interval $I'$ (again, iterating over all EIs in the CEI) if the execution interval is not yet active (chronon $T < I'.T_a$), then the EDF value is calculated with $T = 0$. The intuition behind this policy is that a CEI with less total remaining chronons have less chance to collide with other CEIs, thus a higher probability for more gain in completeness.

The following proposition discusses the class $\mathcal{P}^{[1]}$ of profiles. We denote by $\mathcal{P}^{[1]}$ a set of profiles for which any EI $I$ of any CEI has a width of exactly one chronon.

Proposition 3: For problem instances with $\mathcal{P}^{[1]}$ profiles the $M$-EDF policy is equivalent to the MRSF policy.

Proof: With $\mathcal{P}^{[1]}$: $S$-EDF $(I', T) \equiv 1$ and we get:

$$M$-EDF$(I, T) = \sum_{I' \in \eta} [1 - \| I', S \|] = rank(p) - \sum_{I' \in \eta} \| I', S \|$$

Example 1: Figure 6 illustrates the value each policy assigns to a candidate CEI with four execution intervals. At chronon $T$, $S$-EDF counts the number of remaining chronons until the end of the execution interval (5). MRSF counts the number of remaining EIs in the CEI (4). Finally, $M$-EDF accumulates the number of chronons of all remaining EIs (22). The values each policy assigns are given at the bottom of the figure.

Example 2: Figure 7 illustrates the decision making of each of the policies, when faced with multiple complex execution intervals and a bandwidth constraint. Here, we have two candidates, the first ($CEI_1$) with four required EIs (colored in black) and the second ($CEI_2$) with three EIs (colored in gray). We have used dashed lines to connect a parent complex execution interval to its children. Assume the current time is $T$ and only one EI can be selected ($C_T = 1$). At $T$ the first two EIs of $CEI_1$ were successfully captured (see the $\checkmark$ signs near the top of the figure) yet we allow preemption. A decision needs to be taken whether to probe $EI_1$ or $EI_2$.

$S$-EDF counts the number of remaining chronons until the end of the current EI, yielding 5 for $EI_1$ and 6 for $EI_2$. Therefore, the $S$-EDF policy will stick with $CEI_1$.

MRSF counts the number of remaining EIs. With 2 remaining for $CEI_1$ and 3 remaining for $CEI_2$. Therefore, MRSF will stick with $CEI_1$, choosing $EI_1$.

Finally, $M$-EDF accumulates the number of chronons in all the remaining EIs. We have 19 chronons for $CEI_1$ and 16 for $CEI_2$. Therefore, $CEI_1$ will be preempted, since $M$-EDF favors $EI_2$.

B. Offline Solutions: A Baseline for Online Policies

So far, we have introduced our classification of policies and suggested individual policies for each class. While we were able to show some theoretical bounds on the policies performance, it is limited to special cases. We now provide a brief analysis of the parallel offline solution to this problem. We see two motivations for this analysis. First, offline solutions can serve as a baseline and provide upper bounds to the optimal online performance. Secondly, by analyzing the complexity of an offline solution we can gain some understanding of the difficulty of designing solutions to the online case.

In an offline setting, the proxy is provided with all CEIs in $\mathcal{P}$ for $K$ chronons in advance and has to determine the
Given schedule \( S \) of probing resources in \( R \). Such a scenario cannot be achieved in practice in most cases. Recall that for all but very simple cases new requests for complex execution intervals arrive on the fly. We first discuss the complexity of a full enumeration of feasible schedules, yielding an optimal yet costly solution. Then, we describe an approximation algorithm for the offline case and discuss its properties.

1) Schedule Enumeration: A feasible solution to Problem 1 is a schedule that satisfies the problem constraints. We denote by \( S(C) \subseteq S \) the set of all feasible schedules, given \( C \). Lemma 4 provides the cost of solving Problem 1 using full enumeration.

Proposition 4: Given \( n \) resources, \( K \) chronons, and a constraint \( C \) on the number of probes per chronon, the cost of finding an optimal solution to Problem 1 by enumerating all feasible schedules is \( O(Kn^{KC_{\text{max}}+1}) \) time, where \( C_{\text{max}} = \max_{j=1,2,...,K} \{C_j\} \).

Proof: [Sketch] The number of feasible schedules is given by

\[
|S(C)| = \prod_{j=1}^{K} \sum_{l=1}^{C_j} \binom{n}{l}
\]

(2)

Since \( \sum_{l=1}^{C_j} \binom{n}{l} = O(n^{C_j}) \), we get that \( |S(C)| = O(n^{KC_{\text{max}}} \) \), where \( C_{\text{max}} = \max_{j=1,2,...,K} \{C_j\} \). The computation of a schedule cost is based on a constant number of accesses to each entry in an \( n \times K \) matrix, which sums to \( O(Kn) \). Therefore, the overall cost of finding an optimal solution by enumerating all feasible solutions is \( O(Kn^{KC_{\text{max}}+1}) \).

It is worth noting that \( C_{\text{max}} \) and \( K \) are known yet arbitrary constants and therefore the problem is polynomial in the number of resources. This serves as little consolation whenever \( C_{\text{max}} \) or \( K \) are large (e.g., \( K = 100 \)). We assume that \( C_{\text{max}} \neq O(n) \), otherwise scheduling would be easy since there will be sufficient budget to probe most resources at each chronon. Nevertheless, \( C_{\text{max}} \) may still be of substantial size.

Clearly, enumerating all possible solutions may not be the most efficient way of solving this problem. However, to date we are unaware of any low-polynomial algorithm for solving Problem 1.

2) Offline Approximation: We now shortly discuss how to achieve the best offline approximation for Problem 1. Proposition 5 establishes the relationship between a solution to problems with input \( \mathcal{P}^{[1]} \) to problems with a general set of profiles of the same complexity.

Proposition 5: Let \( \mathcal{P} \) be an arbitrary set of profiles with \( \text{rank}(\mathcal{P}) = k \). Let \( \mathcal{A} \) be an algorithm that provides \( \alpha(k) \)-approximation given an arbitrary \( \mathcal{P}^{[1]} \) with \( \text{rank}(\mathcal{P}^{[1]}) = k \), then \( \mathcal{A} \) can provide \( \alpha(k+1) \)-approximation given \( \mathcal{P} \).

Proof: [Sketch] Given \( \mathcal{P} \) we first transform it into a \( \mathcal{P}^{[1]} \) set of profiles as follows. Let \( \eta = \{I_1, I_2, \ldots, I_k\} \) be a CEI with \( k \) execution intervals, where \( \exists p \in \mathcal{P} : \eta \in p \). Further let \( n_q \) denote the number of chronons of \( I_q \in \eta \). We generate \( \prod_{q=1}^{k} n_q \) new CEIs \( \eta_j \) each has \( k+1 \) execution intervals of width of exactly 1 chronon (see Figure 8 for illustration). We position the execution intervals of each new CEI according to combination formed from the EIs chronons of the original CEI \( \eta \). For example, the first new CEI \( \eta_1 \) has each execution interval positioned in a way that it collides with the corresponding EI of the original CEI \( \eta \) in the first chronon of that EI; the second new CEI \( \eta_2 \) has each EI positioned in a way that it collides with the corresponding EI of the original CEI in the first chronon of that EI, except for the first EI which collides with the first EI of the original CEI in the second chronon of its first EI; and so on. Obviously the output of the transformation is a problem with profiles \( \mathcal{P}^{[1]} \), which further has \( \text{rank}(\mathcal{P}^{[1]}) = k+1 \). Thus, \( \mathcal{A} \) can provide \( \alpha(k+1) \)-approximation to the new transformed problem. It is easy to show that any solution produced by \( \mathcal{A} \) to \( \mathcal{P}^{[1]} \) is a solution to the original problem \( \mathcal{P} \). Therefore, \( \mathcal{A} \) provides \( \alpha(k+1) \)-approximation given \( \mathcal{P} \).

Bar-Yehuda et al. have proposed in [11] an offline approximation to the problem of scheduling \( t \)-intervals, also termed split intervals in [11]. In their problem, \( t \)-intervals are composed of segments of arbitrary length\(^{11} \) and a segment represents an interval in which a resource is needed. Note that \( t \)-intervals are slightly different than our CEIs since we require that a resource should be probed only in a single chronon of each execution interval.

Equipped with Proposition 5, we utilize the Local Ratio scheme [11], which has the best approximation ratio given an arbitrary \( \mathcal{P}^{[1]} \) with \( \text{rank}(\mathcal{P}^{[1]}) = k \), and provides \( 2k \)-approximation for \( C_{\text{max}} = 1 \) and \( (2k+1) \)-approximation for \( C_{\text{max}} > 1 \). Thus, for an arbitrary \( \mathcal{P} \) with the same rank, according to Proposition 5, applying the Local Ratio scheme of [11] we can achieve \( (2k+2) \)-approximation when \( C_{\text{max}} = 1 \), and \( (2k+3) \)-approximation for \( C_{\text{max}} > 1 \). For example, assuming that we have as an input profiles of \( \mathcal{P}^{[1]} \) with \( \text{rank}(\mathcal{P}^{[1]}) = 2 \) (each CEI has at most 2 segments), in the case of \( C = 1 \), we can produce a feasible solution to Problem 1 that guarantees at least \( 1/2 \cdot 2 = 25\% \) of the optimal gained completeness.

The Local Ratio scheme of [11] adds additional constraints to Problem 1 and transforms it into a problem of finding

\(^{11}\)It is worth noting that segments in [11] correspond to execution intervals, and the segments of each \( t \)-interval refer to a single resource.
a maximal independent set of $t$-intervals in a **split interval graph**. Using this scheme we can provide the best offline approximation ratio to Problem 1. This solution is, however, limited in more than one way. First, it is not suitable in all but very simple cases to the Web monitoring problem. Secondly, its theoretical bounds are guaranteed only for a set of profiles with no intra-resource overlaps. Finally, this solution does not scale well for real world problem instances (as we show in Section V-D). For all of these reasons, we intend to use it next as a baseline for comparison against our proposed online policies, which can handle Problem 1 efficiently.

V. EXPERIMENTS

A. Datasets and Experiment Setup

1) Datasets: We used both **real-world** and **synthetic** traces. A real-world trace of 732 eBay 3-day auctions with a total of 11150 bids for Intel, IBM, and Dell laptop computers was obtained from an RSS feed for a search query on eBay. A second real-world trace of 130 different RSS feeds with about 68000 news events that were gathered during a period of two months from Aug. to Oct. 2007 was also used. We also used a synthetic data stream that was generated using a Poisson based update model; the parameter $\lambda$ controls the update intensity of each resource. The FPN update model [3] was used to introduce noise to the real-world trace.

2) Profiles and CEIs: We used a profile template to specify complex user needs and to generate multiple profile instances. “AuctionWatch($k$)” is a sample template that monitors the prices of $k$ auctions and notifies the user after a new bid is posted in all $k$ auctions. The start of an EI to monitor an auction will be predicted from an update model constructed from the traces. The length of each EI can be specified as **overwrite** or **window($w$)**. The **overwrite** requests every new bid to be delivered before the next update occurs and overwrites the last published bid. The **window($w$)** requests every new bid to be delivered within a window of $w$ chronons from the time the bid was posted.

We generated up to $m$ profile instances from a template using a 2-stage process and 2 Zipf distributions. Recall that $rank(P) = k$ corresponds to $k$ EIs in a CEI. We determine the rank of each profile instance according to a Zipf($\beta, k$) distribution, where $k$ is specified in the template. $\beta = 0$ implies a random selection or a uniform distribution $U[1, k]$, while a positive $\beta$ value produces more profiles whose CEIs include less ($< k$) EIs. This stage models a variance of the rank (complexity) of a profile.

In a second step, given some profile of rank $k$, we use a Zipf($\alpha, n$) distribution to select a set of resources. $\alpha = 0$ implies a random selection or a uniform distribution $U[1, n]$, while positive $\alpha$ value implies a preference towards “popular” resources. This stage models inter-user preferences and imitates the way popular resources are chosen by users, which in turn generates a skew in accessing resources. For Web feeds the value of $\alpha$ was estimated to be 1.37 [5].

---

### Table I: Controlled Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Range</th>
<th>Baseline value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega$ (chronons)</td>
<td>Max. El length</td>
<td>[0, 20]</td>
<td>10</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of Resources</td>
<td>[100, 2000]</td>
<td>1000</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of Profiles</td>
<td>[100, 2000]</td>
<td>100</td>
</tr>
<tr>
<td>$K$</td>
<td>Number of Chronons</td>
<td>10000</td>
<td>1000</td>
</tr>
<tr>
<td>$C$</td>
<td>Budget limitation</td>
<td>[1]</td>
<td>1</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Avg. updates intensity</td>
<td>[10, 50]</td>
<td>20</td>
</tr>
<tr>
<td>$rank(P)$</td>
<td>Max. profile rank</td>
<td>[1, 5]</td>
<td>0</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Inter preferences</td>
<td>[0, 1]</td>
<td>0.3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Intra preferences</td>
<td>[0, 2]</td>
<td>0</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>Policy</td>
<td>All</td>
<td>All</td>
</tr>
</tbody>
</table>

---

3) Experimental Setup: We implemented a simulation-based environment to test the different solutions. Given a profile template and an update event stream, we generate $m$ profile instances and their CEIs. For the offline approximation, we compute the CEIs for $K = 1000$ chronons. In the online setting, the proxy receives input at each chronon identifying the set of CEIs that overlap in that chronon. The online monitoring is executed on the same problem instances as the offline approximation. We repeated each execution (offline/online) 10 times and recorded the average performances. Table I summarizes the control parameters. We report on the following metrics:

- Completeness: as given in Eq. 1.
- Runtime costs: This is the execution time normalized over the total number of EIs that must be captured.

For comparison with the proposed online policies, we implemented WIC [3]. We used a straightforward implementation of the algorithm presented in [3], setting urgency to be uniform (that is, for each resource $r_j$ and chronon $T_{urgency_j}(T) = 1$), life to be either overwrite or time-window-append($w$) [3] (which corresponds to our window($w$)), and $p_{ij}$ set to 1 if resource $r_j$ has an update at chronon $T_j$, otherwise we set $p_{ij} = 0$. While we note that WIC does not handle complex CEIs, for certain parameter settings, i.e., with $rank(P) = 1$, $w = 0$, and with no intra-resource overlap, both WIC and S-EDF provide an optimal schedule [3]. Therefore both S-EDF and WIC are used as a baseline of simple EI policies for comparison.

The experiments were run on a Lenovo IBM Thinkpad T60 machine, with a 2.00GHz Intel Centrino Duo processor, and 2.00GB of RAM. The algorithms were implemented in Java, using JDK version 1.4.2. The JVM was initiated with a heap-memory size of 1.00GB.

B. Sensitivity of Policies

We have two versions of each of the online policies, namely with and without preemption. To compare them, we used the real-world trace and the profile template AuctionWatch(upto 3), i.e., each profile will monitor up to 3 auctions. We also used a window of $w=20$, i.e., the new bid must be reported within 20 chronons. Finally, we used a budget of $C = 2$ probes per chronon. We compared the completeness of each online policy using various parameter settings, with
and without pre-emption. Figure 9 reports on the results with 400 auction resources, 1590 CEIs and 3599 simple EIs.

Typically, MRSF and M-EDF perform better then \textit{S-EDF} over a range of experiment settings. This reflects that exploiting knowledge about the rank of the profile and the number of EIs to be captured has benefit. Our complete set of experiments with \textit{S-EDF} indicates that it performs better without pre-emption for \( C = 1 \), while for \( C > 1 \), the preemptive version is better. The other two policies, MR SF and M-EDF, almost always perform better with pre-emption. These results were consistent for most of the parameter settings that were tested, with a difference of up to 20\% in completeness between the preemptive and non-preemptive versions of each online policy. We label the policies “(P)” to denote the preemptive version and “(NP)” for non-preemptive.

C. Online Policies vs. the Baseline

We compared the performance of the online policies and WIC, with the offline approximation solution, for different parameter settings. We used the real-world auction trace and the AuctionWatch(\( k \)) profile template with \( \bar{w} = 0 \), i.e., the client requires an immediate probing of each EI as soon as a bid is posted. The \( \text{rank}(\mathcal{P}) \) varies from 1 to 5. For this experiment, if rank =3, then all CEIs that were generated for that problem instance has exactly 3 EIs. The value of \( C=1 \) and the number of CEIs and EIs varied with rank.

We first examine \( \mathcal{P}^{[1]} \) profiles. To generate these profiles and CEIs, we use \( C = 1 \) and avoid intra-resource overlap by ensuring that each EI of a CEI refers to a distinct resource. For this parameter setting, the offline approximation guarantees a 2\( k \)-approximation (for \( \text{rank}(\mathcal{P}^{[1]}) = k \)). This is the best possible offline approximation (see Section IV-B.2). For this setting, from Proposition 3, both \textit{MRSF} (\( \mathcal{P} \)) and \textit{M-EDF} (\( \mathcal{P} \)) policies perform the same and we only report on \textit{MRSF} (\( \mathcal{P} \)).

Figure 10 reports on the online policies and WIC and the offline approximation. The Y axis reports on the percentage completeness of each online policy calculated with respect to the worst case upper bound on the optimal completeness. To calculate this upper bound, for every \( \text{rank}(\mathcal{P}) \) level, we measure the completeness in terms of single EIs that are captured (i.e., assuming that \( \text{rank}(\mathcal{P}) = 1 \)).

Our first observation is that as the rank increases, all policies have to devote more probes to CEIs that may not be satisfied. Hence there is a general trend that the completeness decreases for all policies as the rank increases. Figure 10 and experiments with other settings show that both \textit{M-EDF} (\( \mathcal{P} \)) and \textit{MRSF} (\( \mathcal{P} \)) policies typically dominate the offline approximation, the \textit{S-EDF} policy (preemptive or not), and WIC. The \textit{MRSF} (\( \mathcal{P} \)) can outperform the offline approximation by up to 10\%. The \textit{S-EDF} policy does not dominate the offline approximation. Both the offline approximation and the \textit{S-EDF} policy dominate WIC. To summarize, both \textit{M-EDF} (\( \mathcal{P} \)) and \textit{MRSF} (\( \mathcal{P} \)) policies perform well, they are not dominated by the other policies, and as rank increases, they continue to reach a percentage completeness of over 75\%.

D. Runtime scalability

We next performed a scalability analysis, increasing the number of profiles and EIs, and measured the runtime of the offline approximation and the online policies aggregated over a run with \( K = 1000 \) chronons. The runtime is normalized over the total EIs. This experiment and the two following experiments use a synthetic trace since we needed to vary the update intensity.

First we compared the offline approximation and online policies with small workloads. The update intensity of the synthetic trace was a Poisson with \( \lambda = 20 \), and the number of profiles ranged from 100 to 500. We observed that the offline approximation has several order of magnitude worse runtime than the online policies. For a setting of 500 profiles, rank=5, 1743 CEIs and 8715 EIs, the Offline execution time was 8.6 msec/EI. In contrast, the online policies were as follows: \textit{S-EDF}=0.06 msec/EI; \textit{MRSF}=0.07 msec/EI; \textit{M-EDF}=0.22 msec/EI. We omit the figure due to space limitations.

We further continued to investigate the scalability of the online policies and increased the workload using 2.5 times higher updates intensity and increased the number of profiles up to 2500. The results are in Figure 11; we do not show the time for the offline approximation since it is very high. The number of profiles and the update intensity that were used are given on top of the curves, and on the lower right side of the
We observe a linear trend in the policy's runtime behavior, suggesting that for real problem instances the online policies are scalable and more robust compared to the offline approximation.

E. Workload analysis

We next studied the effect of different workload settings on the completeness of the online policies. We do not report on the offline performance since it is not scalable. We can adjust two parameter settings, namely the average updates intensity per resource (given by $\lambda$), and the number of profiles ($m$), to adjust the workload. This experiment also used a synthetic trace so as to control $\lambda$. The budget is $C = 1$ and Rank=5.

Due to space limitations we only report on the results as we increase the update intensity in Figure 12. We observe that the MRSF($P$) and M-EDF($P$) policies have similar performance and are much better then S-EDF(NP) policy for all workload parameter settings that were used. We also observe that the M-EDF($P$) is slightly lower then the MRSF($P$) policy. For all policies, as the average update intensity increases, each profile requires to capture more CEIs and as a result the gained completeness decreases.

F. Effect of budgetary limitations

We now study the effect of budgetary limitations on the different policies. This experiment uses a synthetic trace. The rank of the profiles is 5. So far we have used a strict budgetary allocation of $C = 1$. We now show that impact of additional budget has on performance. The results are given in Figure 13. We observe that as the proxy budget increases, allowing it to probe more resources per chronon, a remarkable increase in performance is achieved. In particular, both MRSF($P$) and M-EDF($P$) policies utilize the budget much better then the S-EDF($P$) policy. In example, for $C = 1$ completeness of MRSF($P$) = 29% and S-EDF($P$) = 19% but with $C = 5$ completeness of MRSF($P$) has increased to 76% whereas completeness of S-EDF($P$) has only increased to 69%. We conclude that the aggregated view of MRSF($P$) and M-EDF($P$) policies helps to better utilize the budget.

G. Impact of profile rank variance and skew in accessing resources

We conducted experiments on the impact of user preferences on performance, as embedded in the user profiles. For this analysis, we used various $\alpha$ values (which control the skew in accessing resources according to user profiles), and $\beta$ values (which control the variance in profile ranks among the different user profiles). See Section V-A.2 for the definition of these two
parameters. We further used \( \text{rank}(P) \) \text{ upto} 5, which generated profiles \( p \in P \) that have a random rank \( \text{rank}(p) \in [1, 5] \) (using \( \text{Zipf}(\beta, 5) \)). Furthermore, each generated CEI \( \eta \in \mathcal{P} \) randomly refers to up to 5 different resources in \( \mathcal{R} \) (using \( \text{Zipf}(\alpha, 5) \)). We used a synthetic trace and budget \( C = 1 \). We compared the relative performance against the baseline values of \( \alpha = 0 \) and \( \beta = 0 \). For this parameter setting (where \( \text{rank}(P) = 5 \) and \( C = 1 \)) the baseline completeness for the \( M - EDF(P) \) and \( \text{MRSP} (P) \) is around 37\% and for \( S - EDF (NP) \) is around 26\%.

Due to space limitations, we only show the impact of the skew in accessing resources using various \( \alpha \) values in Figure 14. As \( \alpha \) increases, there is less random selection of resources in each profile, with more execution intervals coming from popular resources. The online policies gain more completeness due to more opportunities to capture intra-resource overlapping execution intervals of popular resources.

\( H. \) Sensitivity to update model noise

Finally, we analyze the effect of noise on the performance of the policies. As we discussed in Section III, an execution interval may need to capture some stochastic update event. Suppose we use an update model to generate an EI to capture that event. Then, if the update model is less than perfect and is noisy, there will be some deviation of the EIs from the real event stream. Thus, the EIs may no longer be able to capture the event, leading to lower completeness.

We follow [3] and use the \( \text{FPN}(Z) \) update model to represent a noisy update model for generating CEIs and EIs. The \( Z \) parameter is used to introduce different random noise levels (with probability \( Z \)) into the model. Thus, \( Z = 1 \) corresponds to an update model with no noise (a perfect model). The value \( Z = 0 \) corresponds to a totally noisy model where every EI has a deviation from the real event.

For these experiments, we used the \textit{real-world} auction and news traces. Figure 15 reports on the results of using the \( M - EDF(P) \) policy on the auction trace, for varying \( Z \). We do not show the other policies for lack of space and they show a similar trend. The rank was varied from 1 to 5, and the value of \( C = 1 \). We observe that if we consider a fixed profile complexity level, e.g., rank=3, the completeness decreases as we increase \( Z \) and increase noise. Similarly, for a fixed value of \( Z \), e.g., 0.6, the corresponding curve shows a decrease as we increase the rank. To summarize, completeness decreases with increasing noise and increasing profile complexity.

We observed similar results for the real-world news trace. In that experiment, we used an homogenous Poisson update model calculating \( \lambda \) as the average number of updates of each RSS news resource during a time period of two months to generate the EIs. We then validated the capture of events against the real event trace. For \( C = 1 \), as the rank varied from 1 to 5, the percentage completeness for the \( M - EDF(P) \) policy decreased from 62\% to 20\%.

\textbf{VI. RELATED WORK}

We now review works that involve satisfaction of either simple or complex data needs. While much focus has been given to efficient data processing methods that support complex data needs (expressed for example by queries or user profiles), less attention has been given to efficient data gathering methods in pull-based environments. Many contemporary applications, including Web crawlers [8], Web monitors [3], feed aggregators [12], [13], and lately new Web 2.0 mashup applications (e.g., [14], [15], [16], [17]), require access to multiple Web sources such as Web feeds. We further classify systems that require such data according to the way it is gathered, either by pull or push.

With push based systems, data is pushed to the system and the research focus is mainly on aspects of efficient data processing, where load shedding techniques [18], [19] can be applied in order to control what portions of the pushed data to process. Such systems include publish-subscribe (pub-sub) (e.g., [20], [21]), stream processing (e.g., [22], [23], [24], [2]), and complex event processing (CEP) (e.g., [25], [26], [27]). Pub-sub systems such as the ONYX system [21] allow the registration of complex requirements at servers and focus mainly on the trade-off between data processing efficiency and the expressiveness of the queries that can be processed by the system. Stream processing systems are also push-based in nature and focus mainly on smart filtering and load shedding techniques. Complex event processing (CEP) systems such as the Cayuga system [27] assume the pushing of a stream of raw events and focus mainly on efficient complex events and situations identification.

In this paper we assume a pull based solution. In a pull environment the data processing system is required to collect the data, e.g., via periodically monitoring of resources. Therefore, in the presence of stream data that is available via pull-only access (e.g., such as in the case with Web feeds), streams need to be crossed. Such systems include, among others, query processing in sensor networks (e.g., [28], [29]), Continuous Queries (CQ) and Web Monitoring (e.g., [30], [31], [3], [32]), Grid and Web services query processing (e.g., [33], [34]), and mashups of different Web sources (e.g., [14], [15]). Current pull based monitoring solutions are not fit to handle complex
data needs over multiple data sources. For example, current works in CQ and Web monitoring such as WIC [3] handle only single resource monitoring tasks that are assumed to be independent of each other. Works in sensor networks focus mainly on energy efficient data dissemination methods and thus data completeness requirements come in second, while now new opportunities with flash memory aided sensors require new techniques for data dissemination [29]. Query processing for Web Services has also been suggested [34], where the main focus is on minimizing the costs of query execution plans that access remote data using Web Services.

VII. Conclusions and Future Work

In this work we presented a framework for satisfaction of complex data needs in pull based environments that involve volatile data. We then presented offline solutions, and since those solutions fail to scale we suggested efficient online policies. Using intensive experiments we further analyzed the performance of these policies under different settings and showed that even under restrictive budget constraints they can perform well. We further showed that utilizing additional profile structures can assist to improve performance significantly. As future extension of this work we shall consider more general profile satisfaction constraints given as client profile utilities. Such utilities can further help to construct better prioritized policies. In this paper we assumed that all execution intervals of any CEI are required to be captured. We further intend to extend the notion of CEsIs to a more general construction which allow also alternatives (e.g., capture of a subset of execution intervals).

REFERENCES


Appendix

A. Algorithm details

We assume that the proxy decides which CEIs to probe according to a given policy Φ. The proxy uses the policy Φ to select at each chronon $T_j$ up to $C_j$ execution intervals from $cands(I)$ that belong to candidate CEIs $η_{ij} ∈ candsi(ηj)$.

We now describe a generic online monitoring algorithm. We assume that at the first time the algorithm is executed, $cands(ηj) = ∅$. We further initialize the algorithm with a policy Φ. The online monitoring algorithm receives as input at each chronon $T_j$ the set of new candidate CEIs $η(j)$, the chronon $T_j$, the budget $C_j$, and schedule $S$ that is assigned with the algorithm decisions which resources in $R$ to probe (if any) at chronon $T_j$. We further use two sets $cands^+(I)$ and $cands^-(I)$, the set of candidate EIs $I ∈ candsi(I)$ that belong to CEIs $η ∈ candsi(ηj)$ that have at least one execution interval $ι = j$ that was already captured prior to chronon $T_j$. In the same way, $cands^-(I)$ denotes the complementary set.

Algorithm 1 Online Complex Monitoring

```plaintext
1: input: $Φ, T_j, η(j), C_j, S$
2: $R_{ids} ← ∅$
3: $cands^+(I) ← ∅$
4: $cands(ηj) ← candsi(ηj) ∪ η(j)$
5: $cands(I) ← candsi(I) ∪ getEIs(cands(ηj), T_j)$
6: if $Φ$ is non-preemptive policy then
7:   for all $I ∈ candsi(I)$ do
8:     $η ← parent(I)$
9:     if $∃ I' ∈ η : |I' \cup S| = 1$ then
10:        $cands^+(I) ← cands^+(I) \cup \{ I \}$
11:   end if
12: end for
13: end if
14: $cands^-(I) ← candsi(I) \setminus candsi^+(I)$
15: if $cands^+(I) ≠ ∅$ then
16:   $count ← probeEIs(cands^+(I), Φ, R_{ids}, C_j, T_j, S)$
17: end if
18: probeEIs($cands^-(I), Φ, R_{ids}, C_j = count, T_j, S)$
19: $cands(I) ← \{ candsi^+(I) \cup candsi^-(I) \}$
20: for all $I ∈ candsi(I)$ do
21:   if $r(I) ∈ R_{ids} \setminus I.T_j = T_j$ then
22:     $cands(I) ← candsi(I) \setminus \{ I \}$
23:   end if
24: if $I.T_j = T_j$ then
25:     $cands(ηj) ← candsi(ηj) \setminus \{ parent(I) \}$
26: end if
27: end for
```

The algorithm further uses a procedure probeEIs that is used to decide which resources to probe according to the policy $Φ$. It is worth noting that the policy gets as an input also the current chronon $T_j$ in case the policy decisions depend also on time. The set of resource indexes $R_{ids}$ is used to avoid the waste of probes on resources that were already captured by probing prior overlapping execution intervals at chronon $T_j$. Finally, at the end of the algorithm, the algorithm checks for CEIs that cannot be captured anymore after chronon $T_j$ (since at least one of the execution intervals ends at chronon $T_j$ and has not been captured yet, thus has been “expired”), and removes them from $cands(ηj)$.

B. Complexity Analysis

We now analyze the runtime complexity of the online algorithm scheme for arbitrary policy execution.

Let $N_j = candsi_j(I)$ denote the size of the candidate EI set $cands(I)$ at chronon $T_j$. At chronon $T_j$ the size of $cands_j(I)$ is $\sum_{q=2}^{j} \left( \cup_{η \in η(j)} |η| - C_j - 1 \right) ≤ j \cdot M_j$, where $M_j = \max_{q=1,...,j} \left( \cup_{η \in η(j)} |η| \right)$. In the worst case at every chronon $T_j$ each profile may require to probe a new CEI that requires to probe $k = rank(Φ)$ execution intervals, and thus $M_j = O(m \cdot k)$, and we get that $N_j = O(k \cdot m \cdot j)$. Using an efficient data structure to rank the candidates in $cands_j(I)$ according to the policy $Φ$ (e.g., a heap) we get a worst case runtime of $O(τ(Φ) N_j \log(N_j))$, where $τ(Φ)$ denotes the time complexity of the chosen policy $Φ$. Thus, the runtime complexity is at the worst case $O(τ(Φ)(k \cdot m \cdot j) \log(k \cdot m \cdot j))$.

The values of the policies $S$-EDF, and MRSF can be calculated in $τ(Φ) = Θ(1)$ time since they require constant access to local variables either of a single candidate execution interval and/or its CEI (without accessing its execution intervals); where as the value of $M$-EDF can be calculated in $τ(Φ) = O(k)$ time since each value calculation may require at the worst case to consider $k = rank(Φ)$ execution intervals.

```plaintext
1: Procedure: probeEIs($cands(I), Φ, R_{ids}, C, T_j, S$)
2: count ← 0
3: while count < min($|cands(I)|, C$) do
4:   $I ← Φ(cands(I), T_j)$
5:   $cands(I) ← candsi(I) \setminus \{ I \}$
6:   $i = index(r(I))$
7:   if $r(I) ∉ R_{ids}$ then
8:     $s_{i,j} ← 1$
9:     $R_{ids} ← R_{ids} ∪ \{ r(I) \}$
10:    count ← count + 1
11: end if
12: end while
13: return count
```