

# Satisfying Complex Data Needs using Pull-Based Online Monitoring of Volatile Data Sources

Haggai Roitman <sup>#1</sup>, Avigdor Gal <sup>#2</sup>, Louiqa Raschid <sup>\*3</sup>

<sup>#</sup>*Technion–Israel Institute of Technology*  
Haifa 32000 Israel

<sup>1</sup>haggair@tx.technion.ac.il

<sup>2</sup>avigal@ie.technion.ac.il

<sup>\*</sup>*University of Maryland*  
College Park MD 20742

<sup>3</sup>louiqa@umiacs.umd.edu

**Abstract**—Emerging applications on the Web require better management of volatile data in pull-based environments. In a pull based setting, data may be periodically removed from the server. Data may also become obsolete, no longer serving client needs. In both cases, we consider such data to be volatile. To model such constraints on data usability, and support complex user needs we define profiles to specify *which* data sources are to be monitored and *when*. Using a novel abstraction of execution intervals we model complex profiles that access simultaneously several servers to gain from the used data. Given some budgetary constraints (e.g., bandwidth), the paper formalizes the problem of maximizing completeness.

## I. INTRODUCTION

In this work we discuss complex data delivery of volatile data sources in a pull-based setting. Volatile data is associated with an expiration time. In a pull-based setting, expiration times may stem from the inability of a server to store all history (as is the case with sensors with flash memory or news feeds) and also from the limited usefulness of data to clients. In particular, clients may have different tolerance levels towards delayed monitoring of data. As a result, data that reach a client may no longer be useful (e.g., auction and stock information). Volatile data is accessed by many contemporary applications, which include Web crawlers [1] and Web monitors [2].

To model user data needs and personalize pull-based data delivery, we define a *user profile* to specify *which* data sources are to be monitored and *when*. Profiles may be complex in that they need to access simultaneously several servers to benefit from the monitored data (hence our reference to *complex data delivery*). We use an abstraction of *execution intervals* to model profiles. An execution interval defines a period of time in which a server can be monitored to provide a client with useful information. A profile contains a set of execution intervals, possibly of different sources.

As a concrete example, consider a financial analyst that looks for arbitrage opportunities: “*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*

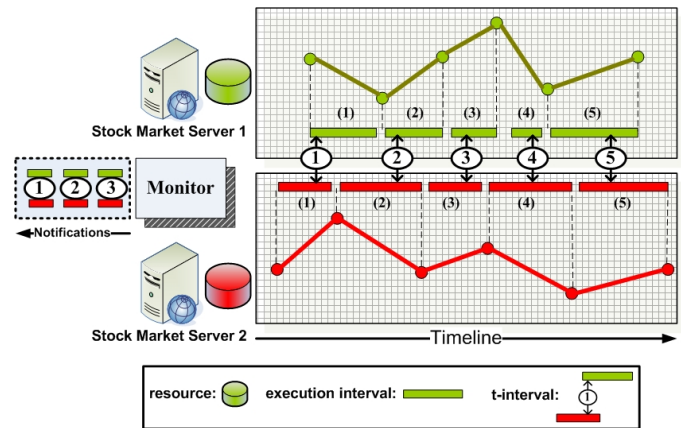


Fig. 1. Example arbitrage monitoring

*between the market prices.*<sup>1</sup> 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. Therefore, a user profile for arbitrage contains pairs of execution intervals (marked as rectangles connected with a numbered oval in Figure 1), a single execution interval for each market. These execution intervals overlap to ensure the validity of arbitrage opportunities (otherwise, prices may refer to different times, invalidating the arbitrage opportunity). The analyst gains some benefit only if both servers were monitored in their respective execution intervals. Therefore, 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. Similar profile examples can be found in accessing multiple auctions.

<sup>1</sup>This definition is taken from Wikipedia, <http://en.wikipedia.org/wiki/Arbitrage>

Given multiple profiles and multiple data sources, we aim at capturing as many of the sets of execution intervals (*e.g.*, price pairs in the arbitrage example) in a profile as possible, given some budgetary constraints (*e.g.*, bandwidth). In this short paper we provide a framework for evaluating complex profiles, serving an array of contemporary applications. This framework enables the extension of some applications to benefit from the presence of multiple clients. We formally present the problem of maximizing completeness, measured in terms of captured execution interval **sets** and discuss available solutions in the literature.

## II. MODEL

*Servers* and *clients* share data in our model through *proxies*. A server manages resources and can be queried by the proxy on behalf of the proxy clients. In this work we assume a hybrid approach, where the proxy probes servers for data via a pull protocol (*e.g.*, by *HTTP GET* queries) and delivers data to clients using a push protocol. Our focus is on the scheduling of proxy pull tasks.

The interest of a client in updates to server’s resources are specified using profiles and stored at the proxy. Each client profile is associated with a set of resources, required by the client, and *execution intervals* [3]. An execution interval defines periods of time during which the client must be synchronized with the state of the resource in order to satisfy the client profile. The client can combine execution intervals to construct complex monitoring profiles over a set of resources. We refer the reader to [3] for details of a language to specify execution intervals and methods to generate execution intervals, possibly based on stochastic modeling (see [4]). The beginning of an execution interval is determined by either an update event to a resource or a temporal event (*e.g.*, every ten minutes). A condition for terminating an execution interval can be set relatively to the stream of update events (*e.g.*, update *overwrite*), or again as a temporal relative event (*e.g.*, five minutes after its beginning). For example, a profile defined over Web feeds may require to collect items published on the feed before the server overwrites them. According to a recent intensive study on Web feeds in [5], 55% of Web feeds are updated hourly. [5] further shows that due to heavy workloads that may be imposed by clients on servers (especially on popular Web feed providers such as CNN), about 80% of the feeds maintained by servers have an average size smaller than 10KB. Thus, we can expect servers of such feeds to keep each item available for a limited life period.

We assume that clients have varying needs. Therefore, each profile may either share or not share some of its execution intervals with other profiles. Given a set of client profiles, the proxy monitors resources, captures updates to resources during their specified execution intervals, and delivers client notifications about resource states, captured during these intervals.

We provide next a formal definition of three building blocks of our model, namely *client profiles*, *execution intervals*, and *schedules*.

### A. Profiles and $t$ -intervals

To formally represent the notion of a complex profile, we extend the notion of execution intervals to that of  $t$ -intervals. A  $t$ -interval consists of execution intervals, possibly of different resources. Each execution interval in a  $t$ -interval should be monitored at least once for the  $t$ -interval to be considered satisfied (or “captured”). A complex profile is simply a set of  $t$ -intervals, modeling the client needs.

Formally, let  $\mathcal{R} = \{r_1, r_2, \dots, r_n\}$  be a set of  $n$  resources and let  $\mathcal{T} = (T_1, T_2, \dots, T_K)$  be an epoch with  $K$  chronons.<sup>2</sup> We assume the proxy manages a set of client profiles  $\mathcal{P} = \{p_1, p_2, \dots, p_m\}$ . A client profile  $p = \{\eta \mid \eta = \langle I_1, I_2, \dots, I_t \rangle\}$  is a collection of  $t$ -intervals [6]. A  $t$ -interval  $\eta$  contains several *execution intervals*, where each execution interval (or EI in short)  $I$  is associated with a resource  $r \in \mathcal{R}$  and  $I$  contains a start and finish chronons  $I = [T_s, T_f]; T_s, T_f \in \mathcal{T}; T_s \leq T_f$ . Execution intervals are the most primitive building blocks, serving as a formal tool to capture the *volatile* property of a resource. Profiles,  $t$ -intervals, and execution intervals construct a *hierarchy*, in which a profile is a *parent* of its  $t$ -intervals, and a  $t$ -interval is a *parent* of its execution intervals. Two  $t$ -intervals within the same profile are *siblings*, and two execution intervals within the same  $t$ -interval are also *siblings*. We use the number of EIs in a  $t$ -interval to model *profile complexity*. Therefore, we denote by  $rank(p)$  the maximal number of execution intervals in any  $t$ -interval  $\eta \in p$  ( $rank(p) = \max_{\eta \in p} \{|\eta|\}$ ), where  $|\eta|$  is the number of execution intervals in  $\eta$ . The definition is easily extended to a set of profiles  $\mathcal{P}$  as follows:  $rank(\mathcal{P}) = \max_{p \in \mathcal{P}} \{rank(p)\}$ .

We now refer back to the arbitrage example in Figure 1 that provides a graphical illustration of  $t$ -intervals of a profile  $p$  with  $rank(p) = 2$ .  $p$  requires the monitoring of two different resources (each representing a different stock market). This profile requires to match an execution interval of one resource with an overlapping execution interval of the other resource.<sup>3</sup> The profile requires to deliver the state of each resource on every update before the next update (*overwrite* policy). The dots represent updates to a resource, and execution intervals are given as rectangles.

Execution intervals of different profiles may overlap in time. Further, execution intervals of the same profile may also overlap. For example, in Figure 1 EIs of the two servers overlap. 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. The special case of *no intra-resource overlap* (presented in Figure 1 for a single profile) is of theoretical interest.

<sup>2</sup>A chronon is an indivisible unit of time.

<sup>3</sup>A resource in this case is the stock price maintained by any of the two stock market servers.

## B. Schedules

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

To simplify our formal writing, we next define an indicator whose value depends on whether an execution interval is monitored. We next extend it to the capturing of a  $t$ -interval. Given a profile  $p$ , a  $t$ -interval  $\eta \in p$ , and an execution interval  $I \in \eta$  that refers to resource  $r_i \in \mathcal{R}$ , an indicator  $\mathbb{I}(I, S)$  indicates whether the schedule successfully captures resource  $r_i$  state during the required execution interval  $I$ ; Formally:

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

The definition is extended to  $t$ -intervals as follows. Given a profile  $p$  and a  $t$ -interval  $\eta \in p$ , we say that  $\eta$  is *captured* by schedule  $S \in \mathbb{S}$  if  $\mathbb{I}(\eta, S) = \prod_{I \in \eta} \mathbb{I}(I, S) = 1$ .

## III. 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 [2] and Web Crawlers [1], where at each chronon  $T_j \in \mathcal{T}$  the proxy can monitor up to  $C_j$  resources. This constraint is represented by a budget vector  $\vec{C} = (C_1, C_2, \dots, C_K)$ .

Given a set of client profiles  $\mathcal{P} = \{p_1, p_2, \dots, p_m\}$ , the proxy objective is to *maximize the gained completeness*, that is, to maximize the number of  $t$ -intervals from  $\mathcal{P}$  that are captured given the budget  $\vec{C}$ . A  $t$ -interval is successfully captured once *all* of its execution intervals are captured. Every  $t$ -interval  $\eta \in p$  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 GC in short) from monitoring  $\mathcal{P}$  during  $\mathcal{T}$  according to  $S$  is calculated as follows (where  $|p|$  denotes the number of  $t$ -intervals in profile  $p$ ):

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

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  $\mathcal{T}$ :

$$\begin{aligned} & \text{maximize } GC(\mathcal{P}, \mathcal{T}, S) \\ & \text{s.t. } \sum_{i=1}^n s_{i,j} \leq C_j, \quad \forall j = 1, 2, \dots, K \end{aligned}$$

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 that involve volatile data. In a pull environment the data processing system is required to collect the data, *e.g.*, via periodically monitoring of resources. Such systems include, among others, query processing in sensor networks

(*e.g.*, [7], [8]), Continuous Queries (CQ) and Web Monitoring (*e.g.*, [9], [2]), and Grid query processing (*e.g.*, [10]). Current pull based solutions cannot handle complex data needs over multiple data sources. For example, current works in CQ and Web monitoring such as WIC [2] handle only simple single resource monitoring tasks that are further assumed to be independent one of each other. Other works in sensor networks further 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 [8].

In an offline setting, the proxy is provided with **all**  $t$ -intervals in  $\mathcal{P}$  for  $K$  chronons **in advance** and has to determine the schedule  $S$  of probing resources in  $\mathcal{R}$ . A *feasible solution* to Problem 1 is a schedule that satisfy the problem constraints. Lemma 1 provides the cost of solving Problem 1 using full enumeration.

*Lemma 1:* Given  $n$  resources,  $K$  chronons, and a constraint  $\vec{C}$  on the number of probes per chronon, all feasible schedules can be enumerated in  $O(n^{K C_{\max}})$  time, where  $C_{\max} = \max_{j=1,2,\dots,n} (C_j)$ .

It is worth noting that  $C_{\max}$  and  $K$  are known yet arbitrary constants and therefore the problem is polynomial in the number of resources. This serves as little consolation whenever  $C_{\max}$  or  $K$  are large (*e.g.*,  $K = 100$ ). We assume that  $C_{\max} \neq O(n)$ , otherwise scheduling would be easy since there will be sufficient budget to probe most resources at each chronon. Nevertheless,  $C_{\max}$  may still be of substantial size. To date, we are unaware of any low-polynomial algorithm for solving Problem 1.

## REFERENCES

- [1] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen, "Optimal crawling strategies for web search engines," in *Proceedings International WWW Conference*. Honolulu, Hawaii, USA: ACM Press, 2002, pp. 136–147.
- [2] S. Pandey, K. Dhamdhere, and C. Olston, "WIC: A general-purpose algorithm for monitoring web information sources," in *Proceedings of the International conference on very Large Data Bases (VLDB)*, Toronto, ON, Canada, Sept. 2004, pp. 360–371.
- [3] H. Roitman, "Profile-based online data delivery, model and algorithms, phd thesis proposal," <http://tx.technion.ac.il/~haggair/thesisProp.pdf>, 2006.
- [4] A. Gal and J. Eckstein, "Managing periodically updated data in relational databases: A stochastic modeling approach," *Journal of the ACM*, vol. 48, no. 6, pp. 1141–1183, 2001.
- [5] V. R. Hongzhou Liu and E. G. Sirer, "Client and feed characteristics of rss, a publish-subscribe system for web micronews."
- [6] R. Bar-Yehuda, M. M. Halldórsson, J. Naor, H. Shachnai, and I. Shapira, "Scheduling split intervals." *SIAM J. Comput.*, vol. 36, no. 1, pp. 1–15, 2006.
- [7] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks." in *VLDB*, 2004, pp. 588–599.
- [8] Y. Diao, D. Ganesan, G. Mathur, and P. Shenoy, "Rethinking data management for storage-centric sensor networks," in *CIDR 2007: Proceedings of the third beinnial conference on innovation data systems research*, 2007.
- [9] L. Liu, C. Pu, and W. Tang, "Webcq: Detecting and delivering information changes on the web." in *CIKM*, 2000, pp. 512–519.
- [10] K. Zhang, H. Andrade, L. Raschid, and A. Sussman, "Query planning for the grid: adapting to dynamic resource availability." in *CCGRID*, 2005, pp. 751–758.
- [11] "Online supplementary version," <http://tx.technion.ac.il/~haggair/cosmos.pdf>.