Profile-Based Online Data Delivery

Haggai Roitman, Avigdor Gal
Technion - IIT
Haifa 32000 Israel
{haggair@tx, avigal@ie}.technion.ac.il

Louiqa Raschid
University of Maryland
College Park MD 20742
louiqa@umiacs.umd.edu

Abstract

This research is aimed at providing *theoretically rigorous, flexible, efficient*, and *scalable* methodologies for intelligent delivery of data in a dynamic and resource constrained environment. Our proposed solution utilizes a uniform client and server profilization for data delivery and describe the challenges in developing optimized hybrid data delivery schedules. We also present an approach that aims at constructing automatic adaptive policies for data delivery to overcome various modeling errors utilizing feedback.

1 Introduction

Web enabled application servers have had to increase the sophistication of their server capabilities in order to keep up with the increasing demand for client customization. Typical applications include RSS feeds, stock prices and auctions on the commercial Internet, and increasingly, access to Grid computational resources. Despite these advances, there still remains a fundamental trade-off between the scalability of both performance and ease of implementation on the server side, with respect to the multitude and diversity of clients, and the required customization to deliver the right service/data to the client at the desired time.

We begin with two motivating examples, demonstrating the necessity of a framework for data delivery that goes beyond existing standards.

Example 1 (`RSS`): `RSS` ([17]) is growing in its popularity to deliver data on the Internet but currently faces scalability issues. `RSS 2.0` defines a mostly pull based protocol but supports a simple notification mechanism named "clouds" that can push notifications about `RSS` feed refreshment. Most `RSS` providers do not support the clouds mechanism since it requires bookkeeping about registered clients; this is not scalable for a popular `RSS` provider such as `CNN`. Current `RSS` consumers either poll the server periodically or pull according to authoring tags embedded within the `RSS` channel (*e.g.*, `<ttl>`). The pull based protocol of `RSS` also faces scalability issues; a recent survey in [9] of 100,000 `RSS` feeds from the popular `RSS` repository *syndic8.com* showed that almost 90% of pull actions by `RSS` consumers, typically using software such as `RSS` Readers or Aggregators, did not capture any useful changes to `RSS` feeds.

Example 2 (`Grid`): Consider a simplified scenario of a physicist client who wants to submit her application to a Linux Farm of several thousand nodes. The Linux Farm collects Node Status metadata for each node including attributes such as Machine Up Time, CPU Load, Memory Load, Disk Load, and Network Load; this data is updated every time a new job is submitted to the Linux Farm, and is periodically pushed to clients. She further determines that a remote dataset is needed and it can be retrieved from a number of replica sites. Network behavior metadata for these replica sites are available from either a GridFTP monitor or the Network Weather Service (NWS) monitor. Neither monitor pushes updates to clients. Current data delivery mechanisms provide few solutions to the physicist possibly resulting in excessive polling; this may have a performance impact on these servers.

Current data delivery solutions can be classified as either *push* or *pull* solutions, each suffering from different drawbacks. Push (*e.g.*, [8]) is not scalable, and reaching a large numbers of potentially transient clients is typically expensive in terms of resource consumption. In some cases, where there is a mismatch with client needs, pushing information may overwhelm the client with unsolicited information. Pull (*e.g.*, [7]), on the other hand, can increase network and server workload and often cannot meet client needs. Several hybrid push-pull solutions have also been presented in the past (*e.g,* [5]). Examples of related work in push-based technologies include BlackBerry [8] and JMS messaging, push-based policies for static Web content

**Proceedings of the VLDB2006 Ph.D. Workshop
Seoul, Rep of Korea, 2006**

(*e.g.*, [10]), and push-based consistency in the context of caching dynamic Web content (*e.g.*, [18]). Another related research area is that of publish/subscribe systems (*e.g.*, [1]). Pull-based freshness policies have been proposed in many contexts such as Web caching (*e.g.*, [7]) and synchronizing collections of objects, *e.g.*, Web crawlers (*e.g.*, [4]).

As demonstrated in Examples 1 and 2 the problem we face is that of providing *theoretically rigorous, flexible, and efficient* methodologies for intelligent delivery of data in a dynamic and resource constrained environment that can efficiently *match* client *requirements* with server *capabilities* and generate a *scalable hybrid push-pull solution*.

The rest of the paper is organized as follows. In Section 2 we introduce our proposed model. We then present in Section 3 the different challenges of the proposed research. Section 4 describe our current research progress, and we conclude in Section 5.

2 Model for Online Data Delivery

We begin by presenting the general architecture of our proposed framework. We then introduce our model for data delivery based on the abstraction of *execution intervals*, followed by profile specification.

2.1 Architecture

The proposed framework aims at providing a scalable online data delivery solution. We identify three types of entities, namely *servers*, *clients*, and *brokers*. A server is any entity that manages resources and can provide services for querying them by means of pull (*e.g.,* `HTTP GET` messages) or push (*e.g.,* registration to an alerting service in digital libraries [3]). Each server has a set of *capabilities* for data delivery (*e.g.,* periodical push of notifications). A client is any entity that has an interest in some resources and has *data delivery requirements* in the form of *client profiles*. Given client requirements and server capabilities, a broker is responsible to *match* the client with suitable servers, and provide the client with the desired information of interest specified in the client profile. To do so, the broker may register to servers and as needed augment server notifications with pull actions. Each broker can further act as both server or client of other brokers, formatting a brokerage network as illustrated in Figure 1. The profile specification proposed in Section 2.3 allows a broker to *efficiently aggregate* client profiles, thus achieving scalability.

We now present our model for profile based online data delivery, followed by the profile specification.

2.2 Model

Let $\mathcal{R} = \{R_1, R_2, ..., R_n\}$ be a set of n resources, where each resource R_i is described in some language (*e.g.,* RDF [13]) and has a set of properties, that can be

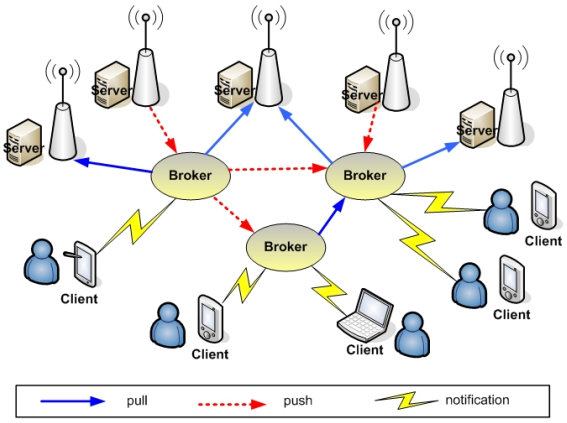

Figure 1: Framework architecture

queried. Utilizing a schema specification (*e.g.,* RDF Schema [14]) we further classify resources and describe properties in different granularity levels. We use $\mathbb{T}$ to denote a discrete timeline.

2.2.1 Execution intervals

Given a query Q which references resources from $\mathcal{R}$, we would like to formally define periods of time in $\mathbb{T}$ in which running query Q is useful for some client. Such periods of times are termed *execution intervals*. We further associate with each interval (denoted as I) the following properties.

Interval end points : Each interval has a start time $T_s \in \mathbb{T}$, and a finish time $T_f \in \mathbb{T}$, where $T_s \leq T_f$.

Query : A selection query over a subset of $\mathcal{R}$, denoted as Q, that should be executed at some time $T_j \in I$.

Utility : A scalar vector of size $|I|$ that defines the utility of executing Q at any time $T_j \in I$. For the client, the utility models the benefit of getting a notification of Q during I, while for the server, the utility models the benefit of pushing data during I.

Label Function : Given the set of resources referenced by query Q, we define a labelling function $\mathcal{L}_I : Q \rightarrow \{\texttt{push}, \texttt{pull}, \texttt{none}\}$. The meaning of the possible labels for each resource is as follows. `push` means that the state (or values) of resource R that is part of Q is pushed during I by some server that stores R. `pull` means that the state of resource R is pulled during I from some server that stores R. `none` if no action is taken during I.

Execution intervals are the basic elements of our model. We shall discuss their usage in Section 2.3.

"Return the title and description of items published on the `CNN` *Top Stories* `RSS` *feed, once three new updates to the feed channel have occurred. Notifications received within ten minutes after new three update events have occurred will have a utility value of* 1*. Notification outside this window has a value of* 0*. Notifications should take place in the two months following April 24th 2006, 10:00:00 GMT."*

Figure 2: Example client profile for `RSS` domain

2.2.2 Events, eventlines and timelines

Events in the context of data delivery identify possible changes to states of resources of interest. We define event e of resource R (denoted by $e(R)$) to be any update event or temporal (time based periodical) event. We denote by T_{e_k} the occurrence time of the k-th event on the eventline. An event occurrence time can be further defined to be the start or end time of an execution interval, denoting that the execution of query Q is conditioned on the occurrence of some event (*e.g.,* an `RSS` channel was refreshed three times). We translate eventlines into timelines and use update models that rely on resource update histories observed from the data delivery process. Such models are usually given in stochastic terms and may suffer from different modelling errors. In Section 3.2 we present an adaptive scheme that aims at providing efficient data delivery despite such errors.

2.3 Profile language specification

We have a particular interest in semantic richness of profiles. Such richness is given in the profile expressive power to allow reasoning about temporal evolution of resources, where profiles can be modified to add and remove resources of interest dynamically to express evolving interests. We aim at unique profile specification that can be used uniformly to define client requirements, and server capabilities in supporting such requirements. Profiles basically state four important aspects of data delivery, namely, *what are the resources of interest*, *when such interests exist and notifications should take place*, *what are the conditions upon notifications*, and *what is the value (utility) of each notification*. An example for a possible client profile for the `RSS` domain is given in Figure 2.

2.3.1 Profile languages

Let $\mathbf{P}$ denote a profile language (or profile) class and let p be a profile. Let $\mathcal{R} = \{R_1, R_2, ..., R_n\}$ be a set of resources. Class $\mathbf{P}$ is a triplet $\mathbf{P} = \langle \Omega, \mathcal{N}, \mathcal{T} \rangle$, where $\Omega \subseteq \mathcal{R}$ is called *profile domain*, $\mathcal{N} = \{\eta_1, \eta_2, ..., \eta_m\}$ is a set of rules termed *notification rules*, and $\mathcal{T} = \{T_1, T_2, ..., T_N\}, \mathcal{T} \subset \mathbb{T}$ is a set of N chronons termed *profile epoch*. Given an epoch $\mathcal{T}$ we define $\mathcal{I}_\mathcal{T} = \{[T_j, T_k] \ | T_j \leq T_k\}$. $\mathcal{I}_\mathcal{T}$ contains all possible intervals in $\mathcal{T}$. Each notification rule η is a quadruple $\eta = \langle Q, \Gamma, \mathbf{T}, U \rangle$, where $Q \subseteq \Omega$ is termed *notification query*, $\Gamma \subseteq \mathcal{I}_\mathcal{T}$ is a set of *execution intervals*, $\mathbf{T} \subseteq \mathcal{T}$ is *notification scope*, and $U : \Gamma \rightarrow \mathbb{R}^m; m \in \mathbb{N}$ is a *utility function*. This general class of profiles can be further classified, for example, to the class of languages that allow to refer to events on the eventline (using T_{e_k}) or those which use deterministic timeline (*e.g.,* periodic time). As an example, Figure 3 contains the specifications of `RSS` client profile.

$$
\begin{aligned}
&\Omega(p) \leftarrow \\
&\{\texttt{http://rss.cnn.com/services/rss/cnn\_topstories.rss}\} \\
&\mathcal{N}(p) := \{\eta\} \\
&Q(\eta) := \sigma_{\texttt{title,description}}(\texttt{item}) \\
&e(\texttt{cnn\_topstories.rss}) := \text{“}\texttt{UPDATE TO channel}\text{”} \\
&\Gamma(\eta) := \{[T_{e_{3*k}}, T_{e_{3*k}} + 10\ \texttt{minute}]; k \in \mathbb{N}\} \\
&\mathbf{T}(\eta) := \text{“}\texttt{4/24/2006 10:00:00 GMT}\text{”} + 2\ \texttt{month} \\
&\mathbf{U}(I) = \begin{cases} [1, 1, ..., 1] \ \textit{if } I \in \Gamma(\eta) \\ [0, 0, ..., 0] \ \textit{otherwise} \end{cases}
\end{aligned}
\tag{1}
$$

Figure 3: Formal specification of the example client profile

3 Challenges

We now introduce two main challenges. The first involves the provision of an optimized hybrid scheduling solution to the data delivery problem. The second challenge involves the provision of a mechanism for data delivery that can adaptively reason about events at servers despite update modelling errors, exploiting feedback gathered from the data delivery process. We now describe each challenge in details.

3.1 Optimized hybrid scheduling

Utilizing the unique profile specification for both client requirements for data delivery and server capabilities to support such needs, *we wish to construct efficient, flexible, and scalable hybrid data delivery schedules to satisfy a set of client profiles by exploiting server push as much as possible while augmenting with pull activities when needed.*

Profile satisfaction ([15]) means that clients are *guaranteed* to get notifications according to their profiles. Given a set of either client or server profiles $\mathcal{P} = \{p_1, p_2, ..., p_m\}$, we translate each profile into a set of client/server execution intervals respectively. For each client execution interval I and each resource R referenced by the interval associated query Q, a schedule S assigns a pair of values $\langle \mathcal{L}_I(R), T \rangle$ which denotes the type of action to be taken during I for resource R and the actual time $T \in I$ that such an action should take place. For example, if schedule S assigns

$\langle \texttt{pull}, T \rangle$ than resource R should be pulled at time T. We denote by $\mathcal{S}$ the set of all possible schedules.

Any schedule that aims at satisfying a given client profile *must* assign either a `push` or `pull` action to each resource referenced by each client execution interval query. Using client profile *coverage*, formally defined in [16], we wish to *efficiently* find a *minimal* set of server execution intervals that *maximizes* the number of client execution intervals that can be covered by pushing data during the client execution intervals. Whenever we identify a set of candidate server intervals that can cover the same client execution interval, we shall further *rank* this set according to metrics such as server availability, commitment to push, and others.

We now describe three important aspects of data delivery that this research considers, namely, *constraint hybrid data delivery*, *scalability*, and *tradeoffs*.

3.1.1 Constraint hybrid data delivery

Data delivery becomes a constrained optimization problem when system resource usage is limited. Generally speaking, given a set of profiles $\mathcal{P}$ we define a target function $F(\mathcal{P})$ to be optimized, subject to a set of constraints on the data delivery schedule, $C(\mathcal{P}, \mathcal{S})$. Examples of possible constraints over pull actions are an upper bound on the total bandwidth, CPU, and memory that can be allocated per chronon (*e.g.,* [11]) or upper bound on the total number of probes allowed per the whole epoch, termed *politeness* in [6]. As for push related costs, each push may have an associated cost set by the server providing the push, thus a further upper bound on total payments can be set. Examples for target functions for the data delivery process are total gained utility, profile satisfaction, system resources used to process $\mathcal{P}$, etc.

3.1.2 Scalability

Current pull and push based solutions face scalability problems. Using our suggested framework for data delivery, we shall accomplish better scalability. Client profiles will be *aggregated* and *optimized*, offering scalable profile processing. In the proposed system architecture, each broker will be responsible to track a set of servers and manage client profiles for several clients, and can further act as either a client or server of other brokers. The hybrid approach can reduce server workloads and save network resources.

3.1.3 Data delivery tradeoffs

Data delivery tradeoffs can be captured by means of multi-objective optimization that involves a set of target functions $\mathcal{F} = \{F_1(\mathcal{P}), F_2(\mathcal{P}), ..., F_m(\mathcal{P})\}$ to be optimized. For example, the tradeoff between *data completeness* and *delay*[1] can be defined as a bi-objective optimization problem as follows.

$$\begin{array}{l} \mathcal{F} = \{F_1(\mathcal{P}) = \text{“}completeness\text{”}, F_2(\mathcal{P}) = \text{“}delay\text{”}\} \\ maximize\ F_1(\mathcal{P}) \\ minimize\ F_2(\mathcal{P}) \\ s.t.\ C(\mathcal{P}, \mathcal{S}) \end{array} \quad (2)$$

It is obvious that such target functions can be dependent as in Example 2, thus optimizing one function can come at the expense of the other. Such dependencies capture “tradeoffs” in data delivery. In multi-objective optimization problems we would like to find a set of non dominated solutions, termed also as *Pareto Set* in the literature, representing optimal tradeoffs and offering different alternatives to clients or servers during the data delivery process. In the general case, finding a Pareto set is an NP-Hard problem and we shall further find an approximated set instead ([12]).

3.2 Adaptive policies for data delivery

Motivated with the work originated in [2] we shall investigate *adaptive policies* to overcome modeling errors and unexpected temporary changes (“bursts”) to existing valid update models. Modeling errors generally occur due to two basic error types. First, the underlying update model that is assumed in the static calculation is stochastic in nature and therefore updates deviate from the estimated update times. Second, it is possible that the underlying update model is incorrect, and the real data stream behaves differently than expected. Bursts are kind of unpredicted noise in the underline update model and are hard to predict. This research will devise an *adaptive algebra* to create (automatic) adaptive policies for data delivery. The basic idea for such algebra is as follows. Given a profile p and some time $T \in \mathcal{T}$, we can associate two sets of execution intervals with each $\eta \in \mathcal{N}(p)$, denoted as $\Gamma_{t \leq T}(\eta)$ and $\Gamma_{t > T}(\eta)$. Lets assume that all actual execution intervals of η up to time T *are known* (*e.g.,* gathered from previous probes feedback). Thus $\Gamma_{t \leq T}(\eta)$ holds (maybe a subset of) those intervals. $\Gamma_{t > T}(\eta)$ still holds execution intervals that their actual alignment is known only in expected or probabilistic terms. An adaptive policy $\mathcal{A}$ takes as input the pair of sets $\langle \Gamma_{t \leq T}(\eta), \Gamma_{t > T}(\eta) \rangle$ and returns a set of execution intervals $\Gamma^*_{t > T}(\eta)$ as output, reflecting the adaptive policy implemented by $\mathcal{A}$. The output set $\Gamma^*_{t > T}(\eta)$ can include corrections to interval end points, alignments, additional intervals, or corrections to label assignments for the hybrid schedule. It is worth noting that such adaptive policies can further take into consideration different constraints (*e.g.,* additional probes that the system can allocate to implement the policy), casting the adaptive task as an optimization problem.

[1] *Completeness* can be measured as the number of execution intervals correctly captured by any schedule, while *delay* can be measured as the time that elapse from the beginning of each such interval.

4 Current progress

Parts of the proposed research have already been accomplished. In [15] we have formally defined the notion of profile satisfiability and cast it as an optimization problem. We have developed an algorithm named `SUP` that guarantees to satisfy a set of profiles in expected terms, while minimizing the efforts of doing so. We also suggested an adaptive version for `SUP`, named `fbSUP` which is an example of an adaptive policy that uses feedback from previous probes and adds additional execution intervals as needed. Our study shows that we improved the data delivery accuracy with a reasonable increase in the amount of system resources that such adaptiveness requires.

In [16] we have suggested a generic hybrid scheduling algorithm named `ProMo` which utilizes profile coverage and aims at maximizing server capabilities usage while providing efficient augmentation of pull actions when needed. In `ProMo` we have dealt with $1:1$ client-server relationship and provided a proxy based architecture. Future work with the brokerage architecture will use more general relationships focusing on providing expressive, yet scalable and efficient hybrid data delivery solution.

In our most recent work we devised an approximated solution to a bi-objective optimization problem similar to the example of Section 3.1.3, termed *Proxy Dilemma*, where a proxy manages a set of client profiles and aims at maximizing the number of execution intervals that can be served, while minimizing the delay in notification to clients, subject to the number of resources that can be probed per chronon.

5 Conclusions

In this paper we presented a novel framework for profile based online data delivery, including architecture, model, language, and algorithms for efficient, flexible, and scalable data delivery. We presented a unique profile specification based on execution intervals which can be used uniformly to describe both client requirements and server capabilities. We presented different challenges the research face in supporting efficient data delivery using the proposed model, and discussed automatic adaptive policies for online data delivery.

References

[1] S. Bittner and A. Hinze. A detailed investigation of memory requirements for publish/subscribe filtering algorithms. In *Proceedings of On the International Conference on Cooperative Information Systems (CoopIS'2005)*, pages 148–165, 2005.

[2] L. Bright, A. Gal, and L. Raschid. Adaptive pull-based policies for wide area data delivery. *ACM Transactions on Database Systems (TODS)*, 31(2):631–671, 2006.

[3] G. Buchanan and A. Hinze. A generic alerting service for digital libraries. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 131–140, New York, NY, USA, 2005. ACM Press.

[4] D. Carney, S. Lee, and S. Zdonik. Scalable application-aware data freshening. *Proceedings of the IEEE CS International Conference on Data Engineering*, pages 481–492, March 2003.

[5] P. Deolasee, A. Katkar, P. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: Deisseminating dunamic web data. In *Proceedings of the International World Wide Web Conference (WWW)*, 2001.

[6] J. Eckstein, A. Gal, and S. Reiner. Optimal information monitoring under a politeness constraint. Technical Report RRR 16-2005, RUTCOR, Rutgers University, 640 Bartholomew Road, Busch Campus Piscataway, NJ 08854 USA, May 2005.

[7] J. Gwertzman and M. Seltzer. World Wide Web cache consistency. In *Proceedings of the USENIX Annual Technical Conference*, pages 141–152, San Diego, January 1996.

[8] BlackBerry Wireless Handhelds. *http://www.blackberry.com*.

[9] Venugopalan Ramasubramanian Hongzhou Liu and Emin Gun Sirer. Client and feed characteristics of rss, a publish-subscribe system for web micronews.

[10] C. Liu and P. Cao. Maintaining strong cache consistency on the world wide web. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, Baltimore, MD, 1997.

[11] 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)*, pages 360–371, Toronto, ON, Canada, September 2004.

[12] C.H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS*, pages 86–92, 2000.

[13] RDF. http://www.w3.org/RDF.

[14] RDF schema. http://www.w3.org/TR/rdf-schema/.

[15] H. Roitman, A. Gal, L. Bright, and L. Raschid. A dual framework and algorithms for targeted data delivery. Technical report, University of Maryland, College Park, 2005. Available from http://hdl.handle.net/1903/3012.

[16] H. Roitman, A. Gal, and L. Raschid. Framework and algorithms for web resource monitoring and data delivery. Technical Report CS-TR-4806, UMIACS-TR-2006-26, University of Maryland, College Park, 2006.

[17] Rss. http://www.rss-specifications.com.

[18] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering server-driven consistency for large scale dynamic web services. *Proceedings of the International World Wide Web Conference (WWW)*, pages 45–57, May 2001.