On the Challenges in Event Delivery

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

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

1 Introduction and motivating examples

Complex Event Processing systems are dependent upon the collection of events from distributed systems. Without a dependent mechanism for event delivery, the inferencing of complex events is hampered and the reliability of the system quickly deteriorates. Determining the right method for collecting events may have a significant impact on system performance and resource utilization and therefore the design of event delivery mechanism should be done with care.

As an example, consider the growing use of RSS feeds. A client can customize the rate of monitoring RSS feeds. The RSS 2.0 specification ([3]) also allows servers to specify some server capabilities, such as Time-To-Live (TTL) (specified by the <ttl> XML tag), indicating when an RSS feed is expected to be refreshed by the server. The RSS 2.0 specification also allows servers to use XML tags such as <skipHours> that provides instructions for clients to skip probing the server during specified hours of the day, *e.g.,* whenever a server is expected to be in maintenance, and therefore, a client probe would yield an old version of the RSS file from the server's cache. The RSS 2.0 specification further supports a mechanism named RSS Clouds (specified by the XML tag <cloud>). Using the cloud mechanism a client can register a callback service method that the server can use in order to notify the client when the RSS feed gets updated. Upon notification, the client is still required to pull the RSS feed data from the server and filter data from the RSS feed. With the lack of a service level agreement (SLA) between the client and the server, there is no way the server can guarantee an upper bound on the delay of its notifications to the client.

With the Google Alerts[1] service a client can register to Web resources, *e.g.*, RSS feeds and HTML files. The client can request to be notified as soon as an event occurs, yet there are no guarantees regarding the delay in notification. A more common way nowadays for accessing RSS feeds is using a feed aggregation client application that pulls the feeds from servers, *e.g.,* Google Reader.[2] Such a client allows the definition of more complex profiles, *e.g.,* defining the monitoring rate of each feed by the feed aggregator or specify a filiter for the RSS feed contents. Such clients are required to collect the RSS feed snapshots directly from the servers, using a pull mechanism.

As another example, we consider AJAX [1], an asynchronous hybrid push-pull communication protocol implemented with XML and JavaScript. In the BAYEUX protocol [4] implementation of AJAX, for example, once a client opens a connection to the server (pull) the connection is kept open for a specified, agreed upon, time window. During this window, the server pushes updates to the client and the client is required to reconnect to the server (pull) to keep the connection (and the possibility for server push) alive. This protocol supports only simple specifications. Also, a server supporting AJAX cannot specify its event delivery capabilities. For example, it cannot specify an upper bound on the delay in notifying the client.

With such a diversity of approaches to event delivery, we next propose a set of basic features of event delivery and offer a classification of delivery mechanisms (Section 2). Then, we briefly outline in Section 3 a set of challenges related with event collection.

2 Classification of Server Capabilities and Event Delivery Patterns

As illustrated above, there is a need for a framework where client *needs* and server event delivery *capabilities* are specified. Client and server specifications should be *independent* and *transparent* to the current event delivery solution. Within this framework, a proxy for the client matches client needs with relevant server capabilities. The proxy must determine if the server capability "covers" the client needs. Further, the proxy must generate an event delivery schedule which meets the client's needs and exploits the available server capabilities. The proxy may augment server

[1] http://www.google.com/alerts.

[2] http://www.google.com/reader.

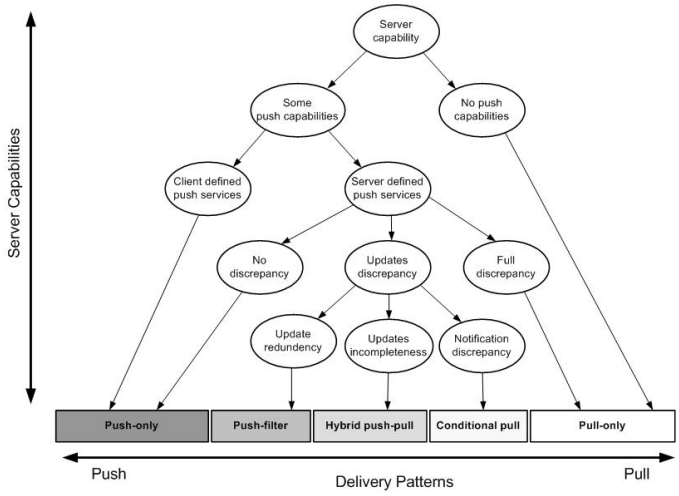

Figure 1. Classification of server capabilities and event delivery patterns

push with pull actions to meet client needs.

We now present a hierarchical classification of server capabilities and the event delivery patterns. Figure 1 graphically illustrates these relationships.

No push services: A server allows client pull, possibly under politeness constraints [2], yet does not provide push services. This scenario is common in the public Web. A proxy needs to poll a server in this scenario, resulting in Pull-only event delivery pattern (see right-most part of Figure 1).

Server-defined push services: A server offers a (limited) set of push services for client subscription. These services may not match a client's actual needs. On the one extreme, there is no discrepancy and the server capabilities can perfectly satisfy the client needs, resulting in a Push-only event delivery pattern (see the path that terminates in the left-most delivery pattern). On the other extreme, the server capabilities cannot satisfy any of the client needs, and therefore, resulting in a Pull-only event delivery pattern.

In between, there are three possible types of discrepancies. The first type is when the server is capable to fully satisfy the client needs, but delivers also some redundant updates that may overwhelm the client. Therefore, some of the push events need to be filtered out by the proxy (Push-filter event delivery pattern, second from left in Figure 1). For example, eBay pushes information to a client who made a bid, whenever the client was outbid. A client that is interested in notifications only when she was outbid by a certain threshold is required to filter out some of these notification.

The second discrepancy type involves partial satisfaction of client needs. For example, a server may push the client periodic notifications, while the client requires to get notifications after each update. In this case, some combination of push-pull is required to satisfy the client profile, augmenting server push by monitoring tasks (Hybrid-push-pull event delivery pattern. middle of Figure 1).

Finally, a discrepancy is possible whenever the timing of server notifications yields no real value to clients. For example, a server may push to a client an update within 10 minutes, while the client needs it in 5 minutes. In this case, the server push is augmented with a conditional pull action in case the notification fails to arrive on time (Conditional-pull event delivery pattern, second from right). Such conditional pull may be also required in cases where a client has to actively pull the server to retrieve the actual content upon receiving a notification of such update occurrence from the server (*e.g.,* using the RSS cloud mechanism).

Client-defined push services: A server in this category receives client specifications and delivers events accordingly, resulting in a Push-only event delivery pattern.

3 Challenges

The main event delivery challenge is the ability to utilize resources efficiently and be flexible in determining when to use push capabilities of a server and when to pull. In some cases, parphrasing on Shakespeare's Hamlet, "to pull or not to pull, this is the question." In other cases, given the costs of server push capabilities, there is a need to balance the utility of push services with those of pull abilities.

Given server capabilities and client needs, some of the specific challenges to be addressed involve the determination of whether a schedule of *push* actions can satisfy client needs; Efficient (*e.g.*, by minimizing the number of *pull* actions and maximizing the usage of server push) augmentation of *push* scheduled actions with *pull* actions so as to satisfy client event needs. Another challenge involves the generationg of a schedule that minimizes the delay in event delivery while minimizing the number of missed events.

References

[1] E. Bozdag, A. Mesbah, and A. van Deursen. A comparison of push and pull techniques for AJAX. *Report TUD-SERG-2007-016a*, 2007.

[2] J. Eckstein, A. Gal, and S. Reiner. Monitoring an information source under a politeness constraint. *INFORMS Journal on Computing*, 2007. forthcoming.

[3] RSS. http://www.rss-specifications.com.

[4] A. Russell, G. Wilkins, and D. Davis. Bayeux - a JSON protocol for publish/subscribe event delivery protocol. *0.1draft3,http://svn.xantus.org/shortbus/trunk/bayeux/bayeux.html*, 2007.