The Internet contains Thousands of Poorly Explored FUTS Data Sources

Pedro Bizarro, Paulo Marques, Rafael Marmelo

CISUC/DEI – Polo II
University of Coimbra
3030-290 Coimbra, Portugal
{bizarro, pmarques}@dei.uc.pt, rjmm@student.dei.uc.pt

Abstract. The Internet contains thousands of Frequently Updated, Time-stamped, Structured (FUTS) data sources. This type of information represents a different class of information that is not properly handled by existing data management systems such as databases, data warehouses, search engines, pub-sub, event processing, or information retrieval systems. In this position paper we describe 9ticks, a system we are designing to collect, parse, store, query, and disseminate FUTS information. 9ticks is helping us understand that all those steps raise new challenges but also bring new opportunities. In this paper we summarize the challenges identified and present our vision of an end-to-end FUTS management system.

Keywords: Internet, database, events, event processing, web crawler, extract web data, manage web data

1. Introduction

The Internet contains thousands of Frequently Updated, Time-stamped, Structured (FUTS) data sources. Unlike semi-structured personal web pages, news sites or blogs, many of those FUTS sources have a very regular structure. Some of those frequently updated data sources are web pages or portions of web pages that, as if they were sensor streams, represent states and updates of real-world things. Examples of such pages include sports scores, stock and exchange information, real-time flight details, weather reports, auction values, traffic reports, monitoring tools (such as Ganglia's cluster monitoring tool [10]), product prices and rankings, DHL and FedEx tracking sites, and many millions of tables with structured information [5, 6]. Similar to a database record, much of this information is composed of a regular, fixed schema of easily inferred data types such as dates, strings, numbers, or unique identifiers. Many of these events – as they are sometimes called – could be used to detect interesting patterns or make important decisions. For example, is road traffic delay much higher today? did my DHL package arrive? is my flight delayed? was there a price drop on my favorite vacation package? what was the average price between a British Airways NY-London flight last year? Currently, users either discover new updates to those sources using simple push mechanisms (e.g., site-by-site alerts or RSS feeds), simple pull mechanisms (e.g., browser or email refresh), or simply not at all!

Although there are many systems that crawl, parse, store, index, and query the web, none is able to capture FUTS data sources and, thus, their values are lost forever. In fact, search engines such as Google don't give much freedom on querying the web as of a point in the past. Sites like the Internet Archive [12] display glimpses of the past, but not detailed enough to, say, determine the average price of a NY-London flight. Zoetrope [1] allows the user to see the past but only for a small subset of pre-selected pages.

We believe that FUTS data sources are not being properly handled by current systems and that there are interesting challenges and opportunities in building a new type of system. In Section 2 we describe our vision of an end-to-end FUTS Data Management System that is able to collect, parse, store, query, and disseminate generic FUTS information while scaling to thousands of sources and millions of users. Next, in Section 3, we identify some of the challenges of building such a system. In Section 4 we describe 9ticks, a prototype FUTS Data Managing System that we are building at the University of Coimbra. Finally, we summarize and conclude in Section 5.

2. A vision for a FUTS Data Management System

A FUTS Data Management System (FDMS) is a system that regularly collects information from millions of frequently updated, timestamped, Internet sources. Some sources will be well-known, commonly requested, previously indexed sources such as stock, weather or flight tracking information which might even take advantage of special protocols and adapters to obtain information before it reaches the web. Other sources will be user-specified sources.

There are hundreds of applications to obtain information from *single*, *well-known sources*. These applications target those commonly requested sources, and run stand-alone in personal computers or smartphones or, as widgets or gadgets as they are commonly called, included in personal dashboard web pages as provided by services such as Alerts.com, iGoogle, NetVibes, PageFlakes, My Yahoo! or Webwag.

These services handle the commonly requested data sources but: i) cannot track user-specific needs, and ii) force the user to install many tens of similar applications or widgets. For example, although there are many applications to track the English Premier League football, there is no similar application to track the Portuguese Second Division football results even though the results are made available in real-time on the web.

The challenge, then, is to build a generic system that can treat any information on the Internet, such as a user-defined *portion* of a web page, as a data source and send it in a timely fashion to specific users. For example, assume someone wants to track how many references are there in Google for "9ticks". In our vision, that user searches Google for 9ticks. Then, using, e.g., a browser plug-in, she clicks the Item Capture option of the browser plug-in (Fig. 1) and next she selects the total on the search results page (Fig. 2).

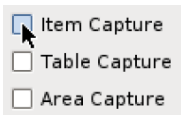
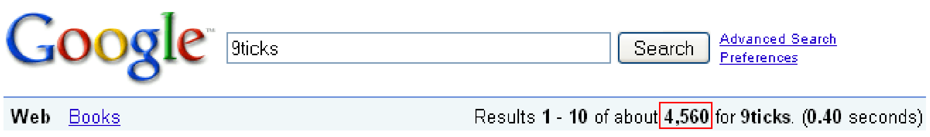


Fig. 1. Select capture mode



Results 1 - 10 of about 4,560 for 9ticks. (0.40 seconds)

Fig. 2. Select the item to capture drawing a box with the computer mouse

The plug-in then parses the page to obtain the path to the selected html element. Next, the system infers the type of the selected element based on the value and using a library of common type formats (some examples of possible inferred types include integer, real, date, time, currency, temperature, DHL tracking number, football scores, golf scores, or free text) or asks the user to provide type information. This information, as well as a refresh rate (user-defined or not), is then transmitted to the FDMS as a new data source to track. Periodically, the FDMS schedules jobs to collect and parse (potentially new) information from the data sources, stores them in persistent, distributed storage, and pushes new information to clients as needed. The user can then see the new source in her personal web page or mobile device together with all the other things she is tracking (Fig. 3). Different types of events will be represented with different graphics or colors depending on data types or user choices.

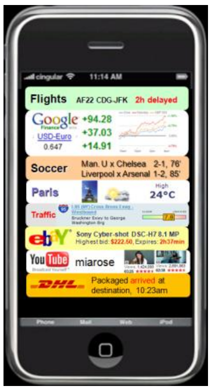

cingular
11:14 AM
Flights AF22 CDG-JFK 2h delayed
Google Finance
+94.28
+37.03
+14.91
USD-Euro
0.647
Soccer
Man. U x Chelsea 2-1, 76'
Liverpool x Arsenal 1-2, 85'
Paris
High
24°C
Traffic
Bruckner Expy to George Washington Brg
ebay
Sony Cyber-shot DSC-H7 8.1 MP
Highest bid: $222.50, Expires: 2h37min
YouTube
miarose
DHL
Packaged arrived at destination, 10:23am
Phone
Mail
Web
iPod

Fig. 3. A FDMS client showing multiple sources and running in a mobile device

In addition to seeing the most recent values from her data sources, a user should also be able to browse back in time and see past values, summaries, or trends.

We expect that in a FDMS such as the one described above the number of subscribers per source will follow a Zipf's distribution [4]. That is, some sources will have millions of subscribers and millions of sources have only a few or just one subscriber. Building a system with millions of sources and millions of users, where data is extracted from web pages, and where the structure of those pages, while mostly fixed, might slowly change over type, identifying multiple sources with equivalent data, optimizing the refresh and push mechanisms, and delivering data in a timely manner are big challenges that need to be overcome. In the next section we list a few of those challenges.

3. FDMS: Challenges Ahead

A FDMS needs to collect, parse, store, query, and disseminate FUTS information. Below, we detail challenges related to those activities.

3.1. Frequency of Revisits

Unlike a search engine, a FDMS has no set of crawlers, jumping from page to page, parsing pages and following links. Instead, the system will start with a number of pre-defined sources and will grow as users add their own preferred sources. While the number of indexed unique sources of a FDMS will be much smaller than the number of unique sources collected by a search engine, the frequency of revisits of the FDMS sources will be much higher than the frequency of revisits performed by a search engine crawler. For example, while Google crawlers revisit personal web pages on the scale of once every week or every month and crawls high-ranked sites such as the BBC several times a day, a FDMS might have to obtain fresh data once per minute (e.g., for football matches or stock updates). In addition, the optimal frequency of revisits of FUTS sources will vary with time (e.g., there are no stock updates during weekends or at night), might be irregular (e.g, only needs to get fresh football scores on game days) and might be knowable (e.g., the exact day and time of games is known before the game starts).

3.2. Collect and Parse

Although the information we want to collect has a regular fixed structure (e.g., a Manchester United football score has always two numbers for the home and away goals), the location of the information in the page might change (e.g., the score information might be in any row of a table with the week matches) or the structure of the web page itself may change. Thus, the correct place to fetch the data from might not exactly match the path stored upon the data source creation. The collect and parse process must be robust to those changes. Finding the location might imply a similarity match between the tree structures of the original and current web page versions.

3.3. Storage

Given the scale of the data to collect and store, a FDMS must have an appropriately scalable storage system. Some of the most scalable storage systems ever built are Bigtable [8] and HBase [3], the ones used by distributed programming tools MapReduce [9] and Hadoop [2], the tools that support the search engines of Google and Yahoo! Those storage systems however, are optimized for high throughput and for batch updates and are likely not appropriate for low response time, continuous inserts. Recent work shows that Hadoop has response times orders of magnitude higher than database management systems performing the same tasks on the same clusters [13]. We expect that developing a petabyte-scale system with millions of queries per day, with very low read response times and very high insert rates is the most challenging task of building an FDMS.

3.4. Query

Building a system that is simultaneously efficient for range queries (e.g., stock values between two points in time), window aggregations (e.g., computing 1h moving sums of the volumes per stock symbol), and continuous inserts using a distributed storage system will be challenging. In fact, the data management market is now segment into different products (databases, data warehouses, event processing systems, and distributed storage systems), each specialized for different types of operations. The specialization of those products is such that, e.g., Hadoop does not even allow the selection of all values between two timestamps.

4. 9ticks: an early prototype

At the University of Coimbra we are building a prototype FDMS code-named 9ticks. 9ticks already tracks pre-defined and user-defined sources, is able to detect simple data types from web pages (integers, doubles, temperatures), schedules revisits of web sources periodically, parses and extracts information from the pages, stores them on Hypertable (an open source, high performance, scalable database, alternative to HBase [11]), produces running aggregations automatically and sends results to web clients.

Currently 9ticks is deployed in a Service Oriented Architecture as shown in Fig. 4.

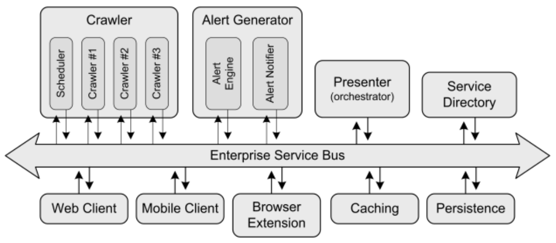


Fig. 4. 9ticks current architecture

Although a SOA is not the best design in terms of end-to-end response time, it will allow starting with an initial system and continuously re-design, replace and scale components as needed with minimum disruption to the other modules.

The *Browser Extension* module is a browser plug-in that lets the user select a piece of a web page as a user-defined data source. The plug-in captures the path to the element selected by the user, identifies the data types in question, proposes display modes and refresh rates, and then sends everything to the Crawler module.

The *Crawler* module is responsible to regularly poll data sources. This module is composed of several sub-modules with the roles of *Adapter*, *Collector*, and *Scheduler*. The Scheduler assigns tasks (e.g., data sources to poll) to Collectors. Collectors perform the polling using Adapters to convert data from the sources.

The *Alert* module is composed by an *Alert Engine* and an *Alert Notifier*. The Alert Engine continuously reads the new information collect by the Crawler module and checks which information needs to be sent to which users. The Alert Notifier then sends the information to the user using one of possible multiple channels (e.g., dashboard application, email, SMS). Currently, the Alert Notifier only sends information to the user *Web Client* dashboard.

The *Persistence* module stores all the information (users, data sources, current and past values, and meta-data) and is currently implemented on Hypertable. The Persistence module automatically computes and stores running averages and sums on some types of sources such as temperatures and stock prices. Those running aggregates are computed at several levels (currently every minute, hour, day, month, and year). Those running aggregations are then used to display past historical data. For example, if a user wants to see a graph of the previous month (day) of historical stock data, then the system will read the aggregated values from the day-level (hour-level) aggregation.

The *Caching* and *Mobile Client* modules, with obvious functions, are not implemented yet.

The *Presenter* module implements the presentation logic by abstracting the system to the Web Client and Mobile client modules. The *Server Directory* module is a well-known service that allows the other services to discover each other.

We are currently working on the Browser Extension, Collector, and Adaptor modules to allow more sophisticated user-defined sources, types, and queries [7], and to make the scrapping process more robust to web page changes.

5. Conclusions

To conclude, the Internet contains thousands of frequently updated, timestamped, structured data sources that are not being stored, parsed, aggregated, or queried. New data management systems with new user interfaces, parsers, storage engines and delivery mechanisms need to be developed to deal with this ephemeral, yet rich and very useful information. We are developing such a system, code-named 9ticks, at the University of Coimbra, Portugal. Unlike other similar systems that also store the past [1, 12] and capture structured information from the web [5, 6, 7], we are first and foremost interested in building a system with very high refresh rates over millions of user-defined data sources extracted from pieces of web pages.

References

1. Eytan Adar et al. Zoetrope: Interacting with the Ephemeral Web. UIST'08, Monterey, CA, October 19-22, 2008. Available at http://www.cond.org/zoetrope.html.
2. Apache Hadoop. http://hadoop.apache.org/. Accessed May 1, 2009.
3. Apache HBase. http://hadoop.apache.org/hbase/. Accessed May 1, 2009.
4. Ricardo Baeza-Yates, Berthier Ribeiro-Neto. Modern Information Retrieval. Addison Wesley, 1999.
5. Michael J. Cafarella, Alon Y. Halevy, Yang Zhang, Daisy Zhe Wang, Eugene Wu. Uncovering the Relational Web. *WebDB 2008*.
6. Michael J. Cafarella, Jayant Madhavan, Alon Y. Halevy. Web-scale extraction of structured data. *SIGMOD Record*, 37(4): 55-61 (2008).
7. Michael J. Cafarella. Extracting and Querying a Comprehensive Web Database. *CIDR 2009*.
8. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *OSDI 2006*.
9. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *OSDI 2004*, pages 10–10, 2004.
10. Ganglia Monitoring System. http://ganglia.info/. Accessed May 1, 2009.
11. Hypertable. http://hypertable.org/. Accessed May 1, 2009.
12. Internet Archive. http://www.archive.org. Accessed May 1, 2009.
13. Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis. *SIGMOD 2009*.