9ticks – The Web as a Stream

Rafael Marmelo, Pedro Bizarro, Paulo Marques

CISUC/DEI – Polo II
University of Coimbra
3030-290 Coimbra, Portugal
rjmm@student.dei.uc.pt, {bizarro, pmarques}@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 [14]), product prices and rankings, DHL and FedEx tracking sites, and many millions of tables with structured information [7, 8]. 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 personal decisions. For example: Is road traffic delay much higher today? Did my DHL package arrive? 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 such as the Internet Archive [16] 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. On the other hand, approaches such as Semantic Web [5, 13], that aim to make web content comprehensible to computers, are complementary to our work but are not as concerned with high scale, high throughput, low response time, as our system is in addressing those issues. Also, unlike Semantic Web approaches, we focus our work on simpler schemas and are not as concerned with matching semantic meaning across schemas.

In short, we believe that FUTS data sources are not being properly handled by current systems. Specifically, there are two major concerns we have. First, FUTS data is not stored anywhere and is lost forever. Second, because FUTS data is not easily available, no alarms are raised or comparisons are made when new data is available.

In the rest of the paper we list some interesting challenges and opportunities in building a new type of system to handle FUTS data properly. 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).

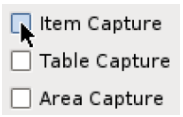

Fig. 1. Select capture mode

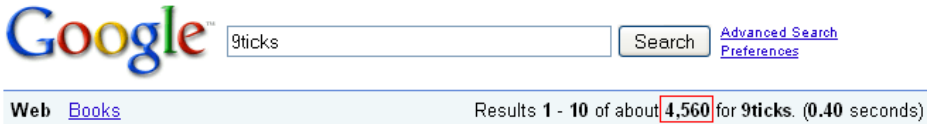

Fig. 2. User selects the item to capture by drawing a (red) box with the computer mouse

The plug-in then parses the page to obtain the XPath to the selected HTML element (normalized to XHTML). 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.

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. Furthermore, in most cases, users shouldn't have to go through this marking process. Once one user has marked a data source, it should become available to any other user looking for the same information. As Google provides a search engine for web pages, a proper FDMS provides search capability for finding previously marked sources.

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).

If the sources are user-defined, defining the frequency of revisits is even harder. Different users might define different revisit frequencies for the same source or might require frequencies which are non-optimal (e.g., 1-second revisit frequency for flight information). The system should detect that a source is changing much slower than the revisit frequency and should automatically increase the revisit frequency. Regarding storage, the system should be able to detect when a source has not changed to avoid storing repeated values.

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.

Note, however, that the user is not expected to mark, e.g., every game of a football team. Instead, the user should mark a place in a web page where the latest results of her team are displayed. For example, the latest Manchester United score will be the first score of its BBC's My Club results page[1].

In addition, many sources, such as the Manchester United latest scores, might be of interest for multiple users. After one user identifies a (portion of a) page as a source, other users can search, find and get events from that source. We expect that, as in Wikipedia, details of each source (e.g., fields to parse, meaning of fields, frequency of revisits) can be discussed, agreed upon, and changed by the interested or higher ranked users.

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 [10] and HBase [3], the ones used by distributed programming tools MapReduce [11] 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 [18]. 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. When to forget and how to forget

A FDMS has to deal with data at a web-size scale. It is, by design, thought to index and store the rapidly changing structured web information. Storing raw data will represent a large number of terabytes. However, a combination of forgetting old enough data with keeping summaries at ever increasing coarser levels will reduce storage requirements. For example, while data recently collected may have a very high granularity (e.g., at a minute or even second scale), older data will be aggregated and summarized (e.g., at a day or month scale). This is natural if we think that, for instance, when doing stock trading people are interested in the current fine changes in price, while looking into the past people only want to send the trends that took place. Also, is some cases a FDMS should *forget* sufficiently old data. For example, some events are only interesting until a specific point in time (e.g., a package that is due to arrive). After that, they rapidly lose interest.

[1] http://news.bbc.co.uk/sport2/hi/football/teams/m/man_utd/results/default.stm

3.5. 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., although planned, Hadoop does not currently 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 [15] (an open source, high performance, scalable database, alternative to HBase), produces running aggregations automatically and sends results to web clients.

Currently 9ticks is deployed in a Service Oriented Architecture [12] as shown in Fig. 4.

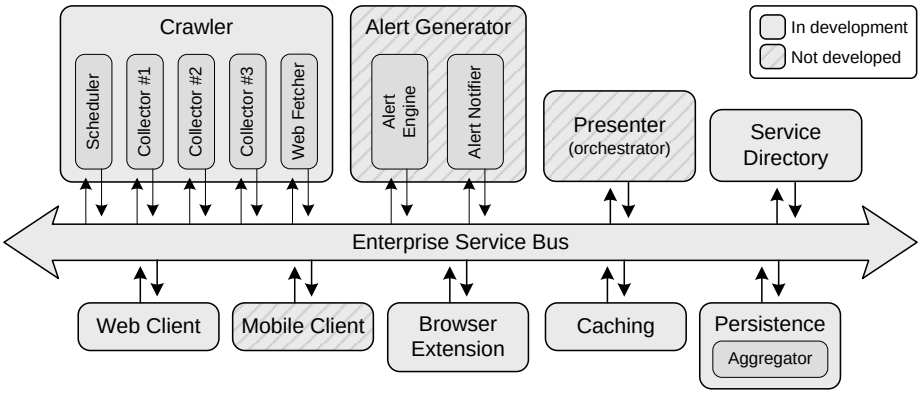

Fig. 4. 9ticks current architecture

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 XPath 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 *Scheduler*, *Collector*, and *Web Fetcher*. The Scheduler assigns tasks (e.g., data sources to poll) to Collectors. Collectors perform the polling using Web Fetchers to convert data from the sources.

The *Alert Generator* module (not implemented yet) will be 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.

Currently, the *Caching* module simple caches the last gathered event. The *Mobile Client* module is not implemented yet. The *Presenter* module will implement 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 improving the Browser Extension, Collector, and Adaptor modules to allow more sophisticated user-defined sources, types, and queries [9], and to make the scrapping process more robust to web page changes. At this point, we support simple HTML and active JavaScript-based pages. Flash-based web pages are not supported although screen-capture followed by OCR could help capture simple flash-based web pages.

Regarding early results, we have been collecting and storing information from about 40 web FUTS sources (weather reports, CPU usage, stocks, eBay prices, football scores, Google searches, digg statistics, and more) for about 2 months. The results from these sources are being continuously collected and summarized at different aggregation levels (every minute, hour, day, month, and year) within Hypertable. The total space consumption is currently just a couple of Gigabytes. The different aggregated values are then used to build smooth graphs with past data. Fig. 5 shows a screenshot of 9ticks: clockwise, starting on the top-left corner, it shows the temperature in London in a 2-week period, the current CPU utilization breakdown of the student accounts server in our department, the high and low values from GE stock, the weekly highest box office US movie, the number of reported hits for a “Swine flu” Google search in the past month, and the current most popular article at digg.

Also shown in Fig. 5 is a drop-down list showing a partial list of sources currently stored at 9ticks. Note that in terms of query language, the user simply searches sources by keyword. Then, the source is added to her dashboard web page and can

then be (re-)configured to display results in other forms (e.g., current value, graph of past values, trends).

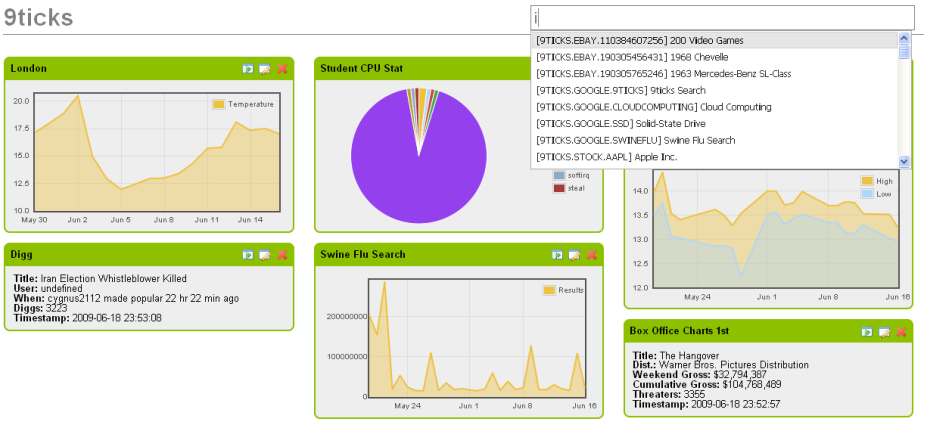

Fig. 5. 9ticks screenshot

The sources, field names, and (aggregated) values are stored in Hypertable using a vertical format (see Fig. 6). Normal relational databases store tuples in a horizontal format, which is not space efficient for sparse datasets. Although an interpreted format is both more efficient than a horizontal or vertical tuple format [6], for simplicity, and because our global schema (of all sources) is a sparse schema, we current store data in a vertical format only. We have a (vertical tuple-format) table for raw data, and extra tables for values aggregated by minute, hour, day, month, and year.

Horizontal

Oid	A1	A2	A3	A4
1			v4	v2
2		v1		
3	v3		v1	
4	v1	v4		
5			v5	

Vertical

Oid	Attr	Value
3	A1	v3
4	A1	v1
2	A2	v1
4	A2	v4
1	A3	v4
3	A3	v1
5	A3	v5
1	A4	v2

Fig. 6. Horizontal vs Vertical tuple format[2]

[2] Image from Beckmann et al, "Extending RDBMSs To Support Sparse Datasets Using An Interpreted Attribute Storage Format", ICDE'06 [6].

5. Conclusions and Future Work

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, 16] and capture structured information from the web [7, 8, 9], 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.

Currently 9ticks already includes the major modules to add sources, collect, parse, aggregate, store, query and display results. The next steps are the completion of the Alert Generator module, the Presenter module, and scaling the system to thousands of sources and hundreds of clients. We also plan to provide APIs so that mashup editors such as Yahoo! Pipes [19] or Microsoft Popfly [17] can efficiently read data from 9ticks and insert data APIs such that other services or sites can add data onto 9ticks.

References

1. E. Adar, M. Dontcheva, J. Fogarty and D. Weld, “Zoetrope: Interacting With the Ephemeral Web”, in Proc. of the 21st Annual ACM Symposium on User Interface Software and Technology (UIST'08), pg. 239-248, Monterey, CA, USA, 2008.
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 and Berthier Ribeiro-Neto, “Modern Information Retrieval”, ISBN 020139829X. Addison Wesley, May 1999.
5. R. Baumgartner, O. Frölich, G. Gottlob, “The Lixto Systems Applications in Business Intelligence and Semantic Web”, in Proc. of the 4th European Semantic Web Conference (ESWC 2007), pg. 16-26, Innsbruck, Austria, June 3-7, 2007.
6. J. L. Beckmann, A. Halverson, R. Krishnamurthy, J. F. Naughton, “Extending RDBMSs To Support Sparse Datasets Using An Interpreted Attribute Storage Format”, in Proc. of the 22nd International Conference on Data Engineering (ICDE'06), 3-8 April, 2006.
7. M J. Cafarella, A.Y. Halevy, Y. Zhang, D Z. Wang and E. Wu, “Uncovering the Relational Web”. In Proc. of the 11th International Workshop on Web and Databases (WebDB'2008), Vancouver, Canada, June 13, 2008.
8. M J. Cafarella, J. Madhavan and A Y. Halevy, "Web-Scale Extraction of Structured Data", in SIGMOD Record, Vol. 37(4), pg. 55-61, December 2008.
9. M J. Cafarella, “Extracting and Querying a Comprehensive Web Database”, in Proc. of the 4th Biennial Conference on Innovative Data Systems Research (CIDR'2009), Asilomar, CA, USA, January 2009.
10. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber., “Bigtable: A Distributed Storage System for Structured Data”, in Proc. of the 7th Symposium on Operating System Design and Implementation (OSDI'06), pg. 205-218, Seattle, WA, USA, November 2006.
11. J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, in Proc. of the 6th Symposium on Operating System Design and Implementation (OSDI'04), pg. 137-150, San Francisco, CA, USA, December 2004.

12. T. Erl, “Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services”, ISBN 0131428985, Prentice Hall, 2004.
13. T. Berners-Lee, J. Hendler and O. Lassila, “The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities”, in Scientific American, May 2001.
14. M. Massie, B. Chun and D. Culler, “The Ganglia Distributed Monitoring System: Design, Implementation, and Experience”, in Parallel Computing, Vol. 30(7), pg. 817-840, Elsevier, July 2004.
15. Hypertable. http://hypertable.org/. Accessed May 1, 2009.
16. Internet Archive. http://www.archive.org. Accessed May 1, 2009.
17. Microsoft Popfly. http://www.popfly.com/. Accessed June 18, 2009.
18. A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, M. Stonebraker, “A Comparison of Approaches to Large-Scale Data Analysis”, in Proc. of the 2009 ACM SIGMOD International Conference (SIGMOD'2009), Providence, Rhode Island, USA, June 2009.
19. Yahoo! Pipes. http://pipes.yahoo.com/pipes/. Accessed June 18, 2009.