Enabling Live Exploration on The Graph of Things [⋆]

Danh Le-Phuoc, Hoan Nguyen Mau Quoc, Quoc Hung Ngo, Tuan Tran Nhat, and Manfred Hauswirth

INSIGHT Centre for Data Analytics
National University of Ireland, Galway.

Abstract. The Internet of Things(IoT) with billions of connected devices has been generating enormous amount data of data every hour. Connecting every data item generated by IoT to the rest of the digital world to turn this data into meaningful actions will create new capabilities, richer experiences, and unprecedented economic opportunity for businesses, individuals, and countries. However, providing an integrated view for exploring and querying such data at real-time is extremely challenging due to its Big Data natures: big volume, fast real-time update and messy data sources. To address this challenge we provides a unified integrated and live view for heterogeneous IoT data sources using Linked Data, , called the Graph Of Things(GoT). GoT is backed by a scalable and elastic software stack to deal with billion records of historical and static data sets in conjunction with millions of triples being fetched and enriched to connect GoT per hour at realtime. GoT makes hundreds of thousand of IoT stream data sources available as a SPARQL endpoint and continuous query channel via the web socket protocol that enables us to create a live explorer of GoT at `http://graphofthings.org/` with just HTML and Javascript.

Keywords: Internet of things, Graph of things, Linked Stream Data

1 Introduction

IDC reports that 90 per cent of all the data in the world has been generated over the last two years. In fact, IDC projects that by 2020 the digital universe will reach 40 zettabytes (ZB), which is 40 trillion GB of data or 5,200 GB of data for every person on Earth. Majority of this data will be contributed by billions of devices connected to the Internet of Things (IoT). Connecting every data items generated by IoT to the rest of the digital world to turn this data into meaningful actions will create new capabilities, richer experiences, and

[⋆] This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289 and by Irish Research Council under Grant No. GOIPD/2013/104 and European Commission under Grant No. FP7-287305 (OpenIoT) and Grant No. FP7-287661 (GAMBAS) and Grant No. FP7-ICT-608662 (VITAL) unprecedented economic opportunity for businesses, individuals, and countries. However, deriving trends, patterns, outliers, and unanticipated relationships in such enormous amount of dynamic data with unprecedented speed and adaptability is extremely challenging. Because as on the Web, access to and integration of information from large numbers of heterogeneous IoT streaming sources under diverse ownership and control is a resource-intensive and cumbersome task without proper support. Such streaming data sources generated from data acquisition infrastructures of Smart Cities, Social network application, medical sensors, etc are still dominated by static data silos, large warehousing systems that are often unmanageable and rudimentary user interaction interfaces which trade intuitiveness for completeness. None of these streams of data can be immediately applied into an application as they first need to be cleaned and made ready for processing.

To answer this need, Linked Stream Data [3] employs Linked Data model to provide graph as the basic representation for stream data together with static data. The effective exploitation of Linked Stream Data from multiple sources requires an infrastructure that supports the intense effort of enrichment, linkage, and correlation of data stream with very large static data collections, e.g, LinkedGeoData and DBpedia, while at the same time combining the result into increasingly complex data objects representative of realistic models of the world. Therefore, in this paper, we present an scalable and elastic approach for exploring and querying billions of dynamic IoT data points in conjunction with static data sets of Linked Data Cloud. This approach provides an integrated architecture to collect and curate useful RDF facts from IoT raw data create a graph that plays the role as a unified and live view of data objects about "Things", called the Graph of Things (GoT). Our back-end data management system supports the ingestion of million data points per second while it is still able to query live data while being indexed to the persistent distributed storage which stores billions-triple datasets of historical data as well as static datasets. The system exposes a web friendly interface to query GoT via HTTP and web socket using SPARQL-like query language. As a proof of concept, we also demonstrate how to build an web application to explore and visualise a pilot GoT dataset at near-realtime using such interface with Javascript. This pilot GoT dataset contains billion triples and millions of triples being updated live every second from millions of sensor data sources (see bellow).

2 The Graph of Things

The Graph of Things (GoT) is an on-going effort of consuming and curating stream data sources of IoT to provide simple interface to filter, aggregate, enrich, and analyze graph-based patterns to visualize business in real-time, detect urgent situations, and automate immediate actions. In current pilot version hosted at `http://graphofthings.org/`, GoT is composed from public stream data sources such as weather, traffic, flight , social media from all continents. These data sources are in a wide range data formats such as XML, HTML, JSON, text, binary. They are consumed and curated in LSM's wrappers to add meaningful links to GoT. The data is fetched or pushed into the system via several protocols such as HTTP, FTP, TCP/IP, web sockets, MQTT. Along with such stream data having been archived since July, 2014, we also transform historical NOAA's weather data[1] of over 23.000 sensors for last 100 years (7.5 billion raw data records in text format). For social media, we also extracted RDF-based named entities from Twitter stream channels and RSS feeds of well known news papers such as BBC, CNN, Yahoo! News. The summary of such stream data sources consumed until now are presented in Table 1.

Dataset	Size of Archive	Period of Archive	Update Frequency
NOAA	177B	Since 1901	3 hours
LSM [2]	2B	Since July, 2014	2 hours
Tweet streams	756M	Since July, 2014	1 second
RSS feeds	13M	Since July, 2014	1 hour
Flight Data	62M	Since July, 2014	5 minutes
Traffic Data	63M	Since July, 2014	1 second

Table 1: Summary of stream datasets until 19/09/2014

During the data curation phase, we use several data sets from Linked Data Cloud to create meaningful links to them. These links plays the pivotal roles for correlating stream data sources, for instances, find the flights departing the same city is currently in the same airspace of a country. For spatial context, we import LinkedGeoData (20 billion triples) and Geonames datasets(61 million triples). To enrich the named entities of social media stream, we also import YAGO2 (64 million triples) and DBpedia (580 million triples) datasets from which such entities are linked to.

3 Architectural Design

The architecture of managing GoT follows the layered architecture of our Linked Stream Middleware(LSM) [2] as shown in Figure 1. In the Data Acquisition Layer, we plug a wide range of wrappers to transform and curate stream data from variety of formats, protocols and device platforms to link streaming triples to the GoT layer which stores in distributed persistent partitions together with distributed in-memory storages of the processing cluster. The GoT layer provides interfaces for two query processing engines, i.e., SPARQL engine and CQELS engine, to enable the application developers to query data via SPARQL endpoint or Stream Subscribing Chanel in the Application layer.

In this architecture, the stream data of GoT is collected in the form of data streams, i.e. information is produced in real-time. However, traditionally, on the "consuming" side, the dominant processing paradigm is still batch-based, i.e., the complete data is first generated, then stored in a database and then the database is used for modelling and analysis, neglecting the dynamicity of the information and causing significant delays. To reduce the such delays, in our architecture, data will be generated on demand and immediately routed to where it will be needed and will be processed as soon as it becomes available as shown in Figure 2a. The processing flow of our architecture moves from a batch

[1] http://www.ncdc.noaa.gov/

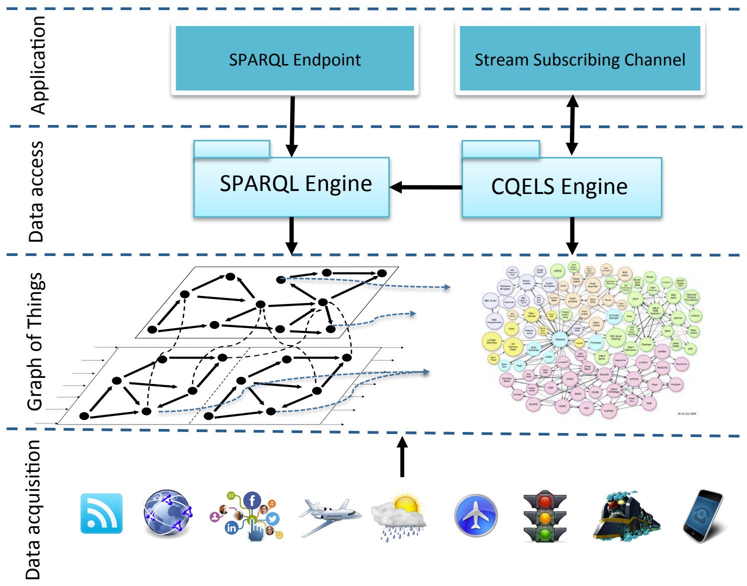


Fig. 1: Layered Architecture

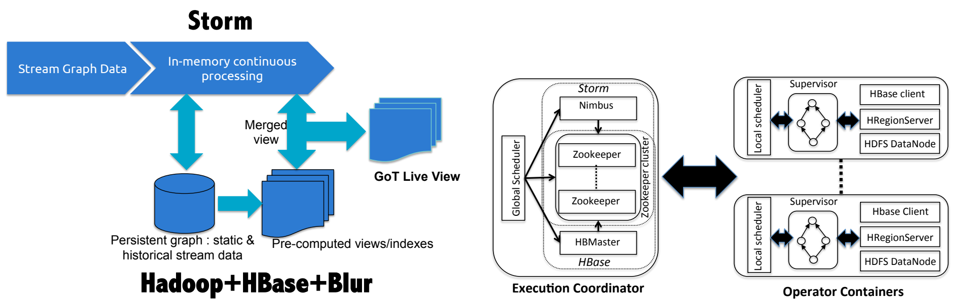
to a near-real-time processing paradigm using parallel processing infrastructures like Storm[2], Hadoop[3], HBase [4] and Blur [5]. For stream data ingestion we use Storm to process and filter them at realtime using declarative stream processing pipelines supported by CQELS Cloud [1, 5]. The newly updated data will be maintained in a live view to be merged with precomputed views or indexes stored in persistent storages. When the data in the live views reaches to certain threshold it will be partitioned and batched to store in corresponding partitions with indexes to be retrieved in parallel fashion. Such data is partitioned based on their characteristics and graph patterns. Free text data will be stored and partitioned by Blur. The time series data is splited and stored in our modified version of Open TSDB[6] backed by HBase. The static datasets is partitioned by the graph partitioning algorithm in [4].

The above processing flow is parallelized using CQELS Cloud parallel execution framework [1, 5] in Figure 2b. This framework is used to build highly parallel execution pipelines of SPARQL Engine and CQELS Engine. The execution of such pipelines is scheduled and coordinated by Storm and HBase's co-ordination services, thus, the elasticity of our system is powered by Storm and HBase.

4 System's Deployment

As GoT is a Linked Dataset, we make GoT accessible via SPARQL endpoint at http://graphofthings.org/sparql/. However, this SPARQL endpoint support

[2] `https://storm.incubator.apache.org/`
[3] `http://hadoop.apache.org/`
[4] `http://hbase.apache.org/`
[5] `https://incubator.apache.org/blur/`
[6] `http://opentsdb.net/`



(a) Data Consuming (b) Parallel Execution

Fig. 2: Scalable and Elastic Data Processing Framework

more powerful SPARQL query language than SPARQL 1.1. For spatial computation, it supports spatial extension for SPARQL query via Jena Spatial built-in function which are mapped to spatial computation functions provided by Blur. For query graph pattern associated with time series data, we also support temporal extension for SPARQL which is backed by our modified version of Open TSDB. On top of that, full text search is supported by fuzzy matching syntaxes of Lucence[7] which is processed by Blur. Following is an example of a SPARQL query over GoT using spatial, temporal and free-text search patterns.

```
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  text: <http://jena.apache.org/text#>
PREFIX  spatial: <http://jena.apache.org/spatial#>
PREFIX  temporal: <http://jena.apache.org/temporal#>
SELECT *
{?loc spatial:withinCircle (67.033 -178.917 10.0 'miles' 100).
?loc <http://www.w3.org/2003/01/geo/wgs84_pos#lat> ?lat.
?loc <http://www.w3.org/2003/01/geo/wgs84_pos#long> ?long.
?sensor <http://www.loa-cnr.it/ontologies/DUL.owl#hasLocation> ?loc.
?sensor temporal:sensors ('1971/01/01-03:00' '1971/01/01-09:00' '.
?sensor text:query (rdfs:label 'Birmingham').}
```

To support continuous queries over stream data of GoT, a Stream Subscribing Channel is given via web socket protocol at ws://graphofthings.org/cqels/. Via this channel, any client can pose continuous queries using CQELS query language [1] over stream data to get stream notification of interest. For example, a browser can use simple java script code to send a CQELS query (in text string) to get location updates of all air planes of an airline within a spatial boundary, e.g. Europe's airspace. This channel is especially useful for realtime web/mobile application that use Model-Controller-View (MVC) front-end programming frameworks and AngularJS, Backbone.js as the updates from streams of GoT can automatically triggered the relevant visualisation widgets of without having to interfere other parts of the applications (see more details in Section 5)

The underlying setup to serve the data described in Section 2 is a cluster of 7 servers running on share network backbone with 10Gbps bandwidth. Each server has following configuration: 2x E5-2609 V2 Intel Quad-Core Xeon 2.5GHz 10MB

[7] http://lucene.apache.org/core/

Cache, Hard Drive 3x 2TB Enterprise Class SAS2 6Gb/s 7200RPM - 3.5" on RAID 0, Memory 32GB 1600MHz DDR3 ECC Reg w/Parity DIMM Dual Rank. One serves is dedicated as front-end server and coordinating the cluster, other 6 servers are used to stored data as and run as processing slaves. Our current deployment uses Zookeeper 3.4.5-cdh4.2, Storm 0.9.2, Blur 0.2, OpenTSDB 2.0 and HBase 0.98.4. A cluster includes 1 master node which has Nimbus, Zookeeper and Blur and HBase master installed. The other 6 nodes are within the same administrative domain play as Blur and HBase slaves.

5 Demonstration Features

We demonstrate the capability of managing big volume of data as well high updating throughput by walking through the process of building the live explorer of GoT using HTML and Javascript at `http://graphofthings.org/`. The GoT Explorer starts with a Live View that summarises "what's been happening in last X minutes", default value of X is 60 minutes, X can be changed by the slider on the left hand side as illustrated in Figure 3. The HTML page will call SPARQL queries corresponding to the map area and the time range of interest to fetch back ground information, i.e, locations and types and updating summaries of stream data sources that have readings in X minutes, to render information on the map. To keep the HTML page updated with the data streamed from relevant stream data sources, a Javascript agent of the HTML page register respective CQELS queries to update the summary of live information in the Live Update Dash board overlayered on the bottom of the map.

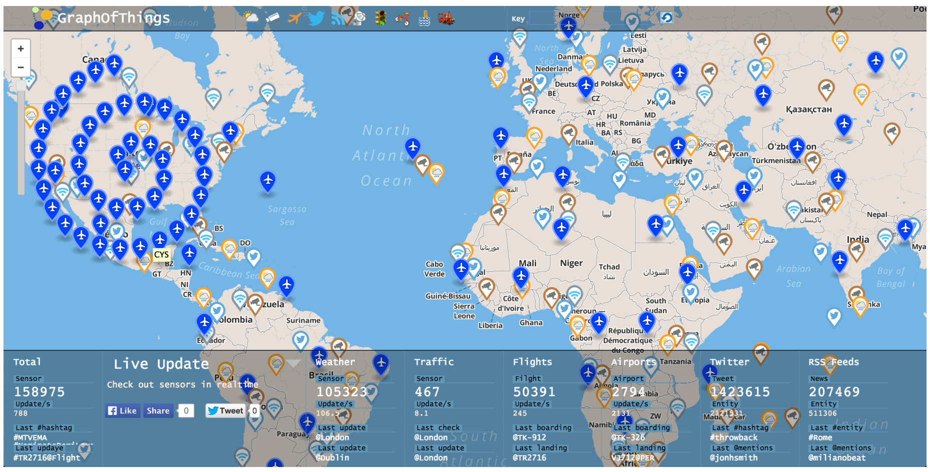


Fig. 3: Live View

To have a quick comparison on historical data of a selected set of stream data sources, e.g, sensor measurements, the GoT explorer provides a 3D layout of live thumbnails as shown in Figure 4. A live thumbnail is live a snapshot of the stream data sources that are fetched via SPARQL Endpoint. The 3D layout

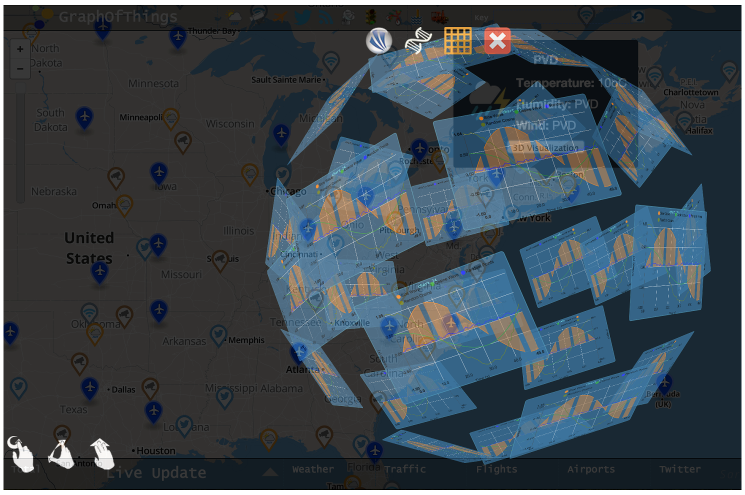


Fig. 4: 3D layout of live thumbnails

can display much more information that the usual 2D one. For example, the sphere layout in the Figure 4 can render much more thumbnail charts. This is very useful when exploring and correlating millions of stream data sources.

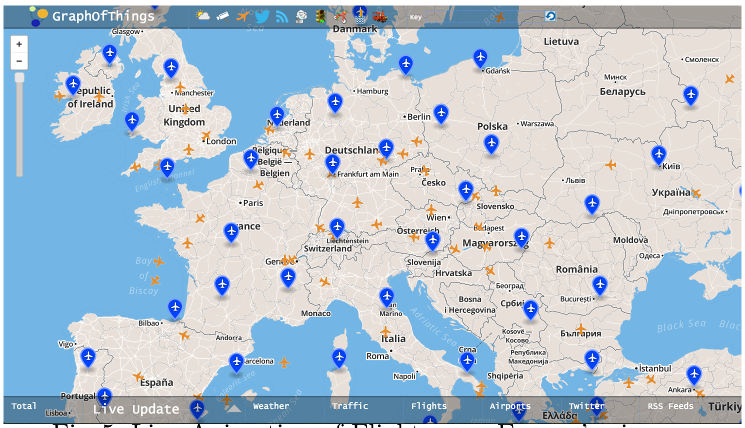


Fig. 5: Live Animation of Flights over Europe's airspace

By using temporal SPARQL query patterns, we can create animated visualisations of from time series data of interest. For instance, the Figure 5 plays live animation of flights over Europe's airspace.

References

1. D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *Proceedings of 10th International Semantic Web Conference*, pages 370–388, 2011.
2. D. Le-Phuoc, H. Q. Nguyen-Mau, J. X. Parreira, and M. Hauswirth. A middleware framework for scalable management of linked streams. *Web Semantics: Science, Services and Agents on the World Wide Web*, 0(0), 2012.

3. D. Le-Phuoc, J. Xavier Parreira, and M. Hauswirth. Linked stream data processing. In T. Eiter and T. Krennwallner, editors, *Reasoning Web. Semantic Technologies for Advanced Query Answering*, volume 7487 of *Lecture Notes in Computer Science*, pages 245–289. Springer Berlin Heidelberg, 2012.
4. K. Lee and L. Liu. Scaling queries over big rdf graphs with semantic hash partitioning. *Proc. VLDB Endow.*, 6(14):1894–1905, Sept. 2013.
5. D. L. Phuoc, H. N. M. Quoc, C. L. Van, and M. Hauswirth. Elastic and scalable processing of linked stream data in the cloud. In *The Semantic Web - ISWC 2013*, pages 280–297, 2013.

Appendix: Meeting the The Requirements of Big Data Track

Minimal requirements

√ **Data Volume.** Until 19/09/2014, GoT dataset contains more than 200 billion triples. Among them stream data contributes 180 billion triples and 21 billon triples of static datasets are loaded.
√ **Data Variety.** The original data sources of GoT are in various formats, e.g, XML, HTML, JSON, binary. The data is has some certain structures to transform and enrich to RDF but RDF-based entities are extracted unstructured Tweet streams and RSS feeds using natural language processing tools. The pluggable wrapper mechanism of our Linked Stream middleware provides easy mechanism for plugging any stream data sources. Current version of GoT feeds data coming from more than 120.000 live stream sources under control of more than 70 providers.
√ **Data Velocity.** The stream part of GoT has millions of updates per hour. The system is incrementally indexing 1.2 billion triples per month to store in the persistent storage of the cluster. The back-end system can consume millions of triples per second on a cloud infrastructure, it can adapt to the fluctuate stream speed. Its scalability and elasticity is tested in [1, 5].

Additional Desirable Features

√ **Usability.** The system provides a SPARQL Endpoint and a Stream Subscribing Chanel which can be used posed complicated SPARQL queries as well as continuous queries in CQELS language. The query languages support spatial, temporal and free text search features over GOT data. The Endpoint and the Chanel supports JSON data format to make it easier to parse on Web browsers or mobile clients. Their usability is demonstrated via the live GoT Explorer in Section 5.
√ **Scalability.** The system is elastically scalable on the cluster of 7 servers. Its underlying processing system, CQELS Cloud [1, 5], is tested to scale perfectly on 32 processing nodes.
√ **Near-real time processing capability.** The system consumes, indexes incoming stream data at near-real time. The data can be queried as soon as the data transformed and stream to processing bus.