The Graph of Things: A Step Towards The Live Knowledge Graph of Connected Things

Danh Le-Phuoc[a,*], Hoan Nguyen Mau Quoc[a], Hung Ngo Quoc[a], Tuan Tran Nhat[a], Manfred Hauswirth[b]

[a]*Insight Centre for Data Analytics, National University of Ireland, Galway*
[b]*Institut für Telekommunikationssysteme, Technische Universität Berlin, Berlin, Germany*

Abstract

The Internet of Things (IoT) with billions of connected devices has been generating an enormous amount 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 provide 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 datasets in conjunction with millions of triples being fetched and enriched to connect the GoT per hour at realtime. The GoT makes approximately a half of million stream data sources queryable via a SPARQL endpoint and a continuous query channel using 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, Real-time Search Engine

1. Introduction

International Data Corporation (IDC) reports that the digital universe has been grown by a factor of 300 from 2005 to 2020. Specifically, 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 [1]. The 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 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. Besides, such distinct streaming data sources are generated from distributed data acquisition infrastructures of Smart Cities, Social network applications, medical sensors, etc. Hence, traditionally, to cross-correlate and analyze them into higher level data products, they need to be transformed, cleaned and consolidated into a large static data warehouse and then made ready for certain types of rudimentary query patterns, e.g., OLAP queries. However, these streaming sources operates on longer time scales on which a wide range of dynamic data feeds are continuously arriving from disparate and uncontrolled sources [9], thus, it is much more difficult to maintain a fresh and consistent integrated view for unlimited discovery and exploration of enormous ever growing IoT data.

Motivated by such challenges and inspired by Knowledge Vault [6], we aim to create a Live Knowledge Graph to pace the way towards building a "realtime search engine for Internet of Things", called the Graph of Things(GoT). Similar to the Knowledge Graph used in search engines like Google [2] and Bing [3], deep understanding of the world around us, GoT aims to enable deep understanding of data generated by connected things of the world around us, called The Graph of Things (GoT). To address aforementioned challenges, GoT is represented as Linked Stream Data [23] which employs the Linked Data model to provide a graph as the basic representation for stream data together with static data. This graph enables a smarter way to discover and explore IoT data under meaningful facts and their relationships.

Along this line, the effective exploitation of Linked Stream Data from multiple sources requires an infrastructure that supports the intense effort of enrichment, linking, and correlation of data streams with very large static data collections, e.g, LinkedGeoData[4] and DBpedia[5]. Moreover, at the same time,

*Corresponding author at: Insight Centre for Data Analytics (formerly DERI), National University of Ireland, Galway, University Road, Galway, Ireland. E-mail addresses: lpdanh@gmail.com

[1] http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf

[2] http://www.google.ie/insidesearch/features/search/knowledge.html

[3] https://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/

[4] http://datahub.io/dataset/linkedgeodata

[5] http://dbpedia.org

this infrastructure needs to support sophisticated queries, discovery and exploration on increasingly complex data objects representing realistic models of the world. Therefore, in this article, we present a scalable and elastic solution for ingesting, storing, exploring and querying billions of dynamic IoT data points in conjunction with static data sets of Linked Data Cloud. Our solution provides an integrated architecture to collect and curate useful RDF-based facts from IoT raw data to create a graph that plays the role as a unified and live view of data objects about "Things". This solution aims to provide a comprehensive software stack with easy-to-use toolkits to filter, aggregate, enrich, and analyze a high throughput of data from multiple disparate live data sources and in any data format to identify simple and complex patterns to visualize business in real-time, detect urgent situations, and automate immediate actions. For dealing with heterogeneity of dynamic data sources, Linked Stream Middleware [19] of this software stack will transform the data in a variety of data formats and data sources to make it ready to any further processing and high-level analytical operations. Its parallel processing layer of the stack will be able to elastically and dynamically distribute processing load to the cluster as well as cope with a large amount of queries and incoming streams as well in a huge volume of data. 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 remainder of the article is structured as follows. The Section 2 will present the process of building the Graph of Things from both physical world and virtual world. In the next section, we will share our design choice of building the system and infrastructure to store and query the GoT. The Section 4 will show some demonstrations of our live system and share some lessons learnt during the course of 10 months of deployment. Finally, the Section 6 will conclude the article and reveal our future plan for GoT.

2. Building The Live Knowledge Graph of Connected Things

To motivate the creation of the Graph of Things, we starts with a real-time search use case as follows. For example, John has just missed the connection flight at the Dublin airport, he has few hours to spare in the *sunny weather*. He intends to take the bus to his favourite Sushi restaurant in the city centre, but he finds out there is *no bus today* due to the current *strike of Dublin Bus*. To avoid current traffic situations in the city centre, he searches for some other Sushi restaurants that *can be reached from the airport in 10 minutes by taxi*. From the list of recommended restaurants, he finds out that there is a Sushi restaurant *next to the Dublin Ferry Port* where an old classmate from Liverpool will be *arriving 1 hour on a ferry*. After finishing the dinner, before coming back to the airport, he is notified that his connecting flight *will be delayed for 2 hours*, and he finds out there is *open-air music show just few blocks from the restaurant*. From *the live camera feed pointing to the show*, there is a big exciting crowd, then, he decides to take a walk to there.

In order to provides such live connected information to the user, we have to continuously fuse various IoT stream data sources such as flights, ships, traffic cameras and weather sensors into GoT as an integrated view. The GoT not only comprises some sensory data captured by sensors also involves the context, the meanings, relationships between data objects. In particular, the GoT is represented as a big connected RDF graph which enables applications, users and developers to traverse along graph edges (RDF triples) without the restriction of the database schema and fixed connections among data items, real world things. The search engine based on this graph can benefit greatly from direct access to these nature relationships to have real intrinsic view of real world events, phenomena. Therefore, the graph can provides smarter search results which lead to users' curiosity on new relevant topics. In above use case, returned data entries to users like flights, Sushi restaurants and events of interests are graph nodes which are connected to other nodes as potentially data of interest for the users. For instance, the restaurant and the music show events while the ferry has spatial relationships which trigger the interesting correlations according to to the users's contexts. Such correlations can be queried by using an extension of SPARQL 1.1 with spatial and temporal filtering conditions for expressing spatial-temporal context of a query. While we defers the technical details expressing and processing such queries, we will present how we model and integrate our IoT data sources summarised in Table 1 in following subsections.

2.1. Collecting facts of Physical Things

For the physical things equipped with sensors to "observe" facts about them, we use SSN Ontology [5] to capture the sensing context including sensor configurations, meaning of what to measure. To have a richer context, we correlate several data sets from Linked Data Cloud to create meaningful links to the sensors, properties, features of interest it measures. These links play the pivotal roles for correlating stream data generated by it, for instance, finding the flights departing from a same city is currently in the same airspace of a country. In fact, we extract relevant spatial data from LinkedGeoData, and Geonames dataset and other relevant known concepts or entities from DBpedia (which are intergrated in YAGO [10]). For instance, users can start with a string "weather in Dublin", the matched labels of weather sensors in Dublin and DBpedia entries of Dublin city, Ireland will trigger the smart data discovery process without being buried in tons of ambiguous results. Such contextual data of each sensor is added to the GoT as a subgraph.

An example illustrated in Figure 1 is sensor metadata of *a weather station that observes the temperature and humidity at Dublin Airport*. It captures the context in which the sensor readings are obtained to generate the dynamic, graph-based stream data from the time-varying sensor readings. These readings have links to their meanings, e.g. *:tempValue (18 Celsius) is the temperature of Dublin Airport at "21:32:52, 09/08/2011"*. Each readings plays the role as a stream subgraph attached to

the GoT when the sensor reading is fed into GoT from the original sensor source.

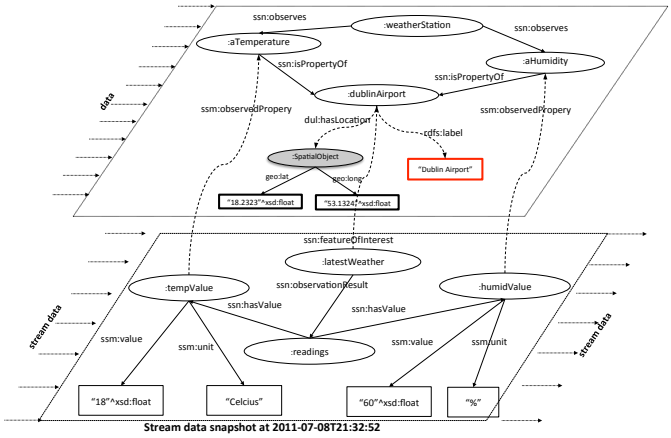


Figure 1: A stream subgraph based on SSN ontotology

Table 1 lists the dominated sensor data sources in our catalog. The "sensing objects" columns present the number of physical objects (places, aircrafts, ships, etc) that the sensing sources observes. These sensing data sources are fetched to GoT in different update rates corresponding to their nature of value, frequent of updates and the distribution of data sources. We has been archiving all fetched data since June, 2014. Among them, The NOAA's Climatic Data Center[6] (NCDC) provides 100 years of meteorological data sources from 20,000 weather stations[7]. However, we decide to limit the queryable archived window back to 1971 due to storage and processing capability of our cluster. Addition to NOAA, LSM sensor data sources [19] offers weather data of 60,000 places around the worlds via Web-based APIs from Web-based weather providers. For aircrafts and ships, they both use transponders to identify them by broadcasting or exchanging their information (identities, coordinates, speed, etc) with radars or other receivers. Such radars or receivers then relay received information to a gateway on which we subscribe or fetch data from. We also built a spider to scrap webpages that provide live camera feeds. As a result, we currently have 45,000 camera feeds, however, due to the size of data and legal issues, we only provide live feeds and do not archive any data. On top of that, thanks for government open and public data projects, we have several small set of data sources that cover small areas (a city or a country). For example, several smart cities projects such as Dublin, London and New York publish a wide range of sensor data sources such as train, bike and bus information.

2.2. *Expand the knowledge graph to Social Things*

Similar to Citizen Sensing [27, 32] and Social Sensing [25, 2], we consider social media such as Twitter, Facebook, RSS feeds as "Social Things" which "sense" events, information

Sources	Sensing objects	Updates/hour	Archived window
NOAA	21k	21k	since 1/1971
LSM	60k	28k	since 6/2014
Camera	45k	live	since 8/2014
Flight	317k	81.6k	since 8/2014
Ship	20k	240k	since 2/2015
Twitter	–	67k	since 8/2014
Others	5,4k	12k	since 8/2014

Table 1: IoT stream data sources of The Graph of Things

from Web citizens. Therefore, we extend the SSN ontology [5] to model a social media extractor as a Sensor. Instead of extracting only the Tweet text, we use natural language tools to extract named entities and then enrich them with meaningful concepts and relationships as RDF triples. The Figure 2 illustrates a snapshot of Tweet data that is extracted as RDF-based triples. In this diagram, the extracted RDF subgraph is driven SSN ontology, NERD ontology[8] and also spatial context in the Tweets.

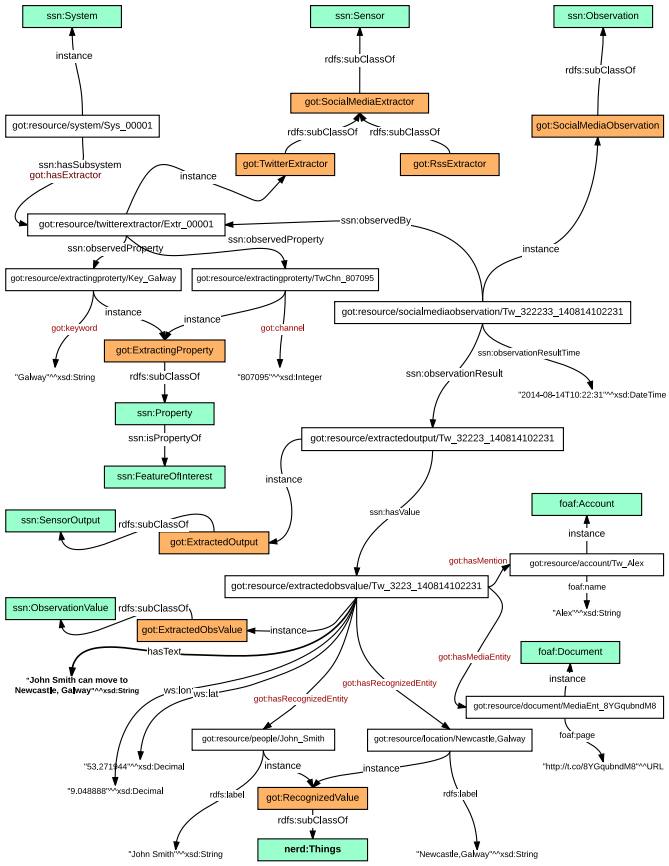


Figure 2: A social sensor reading snapshots of Tweet stream

Currently, we choose Twitter for social network streams, and popular RSS news channels, such as BBC, CNN, Yahoo! News for RSS feed streams. To provide useful relationships

[6] http://www.ncdc.noaa.gov/
[7] http://www.ncdc.noaa.gov/data-access/quick-links
[8] http://nerd.eurecom.fr/ontology

to Physical Things, we prioritise the semantic facts relevance to RDF entities generated from connected things, such as location-based information, events, traffic situations. In particular, for Twitter streams, the system crawls Tweets related to physical sensors in our database judging by their locations and other relevant metadata. However, there are only roughly 4% of tweets that explicitly have geo-tags (longtitude and latitude values). To increase this rate, we use the enriching processing in Figure 3 by using natural language processing techniques on the text of the Tweets. Firstly, raw tweets can be crawled and stored as json files by using TwitterAPI[9]. Next, raw text of tweets are tagged with entities of interest by popular natural language processing toolkits such as StanfordNER[10] [15] and IllinoisNER[11] [30]. These toolkits recognise named entities with three types (including person, location, and organization) from plain text of social media messages, such as Twitter, Facebook, or RSS Feeds [8]. The tagged entities are selected as candidate ones to go through the next step by its frequency of appearance in the crawled Tweets in the recent window of times, e.g, 1 hours. The tagged entities will then be verified by querying YAGO ontology to determine whether they appear in the ontology or not. If we find matched items in YAGO for an tagged entity, we then extracts relevant geographical information as well as other meta data, e.g relevant Wikipedia entries from the YAGO ontology.

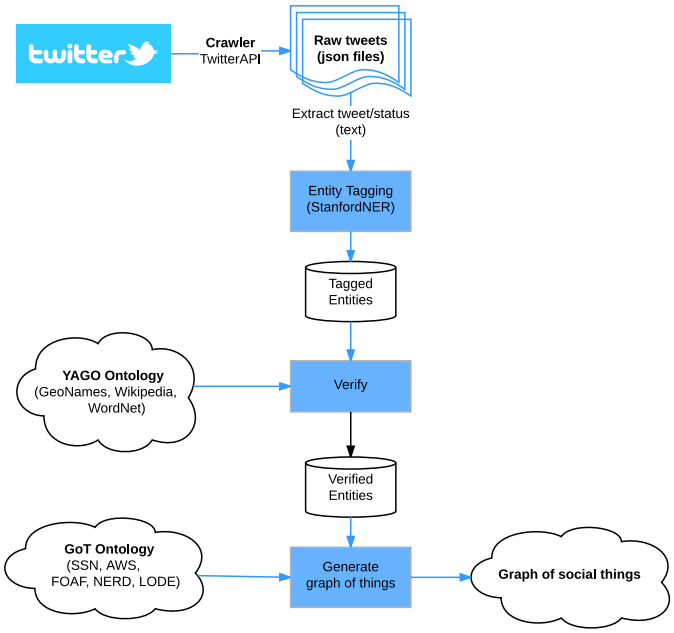


Figure 3: Subgraph Extracting Process of Social Things

Similar to Tweet processing, RSS feeds are also annotated by language processing toolkits to connect to YAGO ontology. However, RSS feeds do not include the location data. The mentioned locations of RSS feeds are only extracted by linking with GeoNames entities in YAGO. These linked entities will have spatial properties like "http://www.w3.org/2003/01/geo/wgs84_pos#lat" and "http://www.w3.org/2003/01/geo/wgs84_pos#long".

3. System Design

From the description of the Graph of Things above, this section will design the scalable and elastic architecture, storage and infrastructure that enable users/developers to continuously add and query IoT data at near-realtime.

3.1. Architecture

To design the data management system for GoT, we decouple the data processing flow by following the the layered architecture of our Linked Stream Middleware(LSM) [20] as shown in Figure 4. The data consumption is handled in the Data Acquisition Layer which provides a wide range of plug-in wrappers. These wrappers transform and curate stream data from variety of formats, protocols and device platforms to link streaming triples to the Graph of Things layer (GoT layer). The GoT layer stores and indexes RDF-based data in distributed persistent partitions together with distributed in-memory storages of the processing cluster as presented in Section 3.2. The GoT layer provides data access interfaces for two query processing engines, i.e., SPARQL engine and CQELS engine [17, 22], to enable the application developers to query data via SPARQL endpoint or Stream Subscribing Chanel in the Application layer. While SPARQL Endpoint serves one-shot queries using an extension of SPARQL 1.1 query language with spatial, temporal and free text built-in functions via a customised SPARQL Engine, Stream Subscribing Chanel serves continuous queries using CQELS-QL query language [16] via websocket. CQELS Engine is a stream processing engine which supports continuous queries over RDF stream data. A continuous query is a long standing query which is continuously triggered when the new relevant data arrives. Therefore, via the websocket protocol, once a user subscribes a CQELS-QL, a stream of output will be fed back to the subscriber.

In the Data Acquisition layer, the data is fetched or pushed into the system via several protocols such as HTTP, FTP, TCP/IP, web sockets, MQTT and then is processed asynchronously in different processes on distributed processing nodes. These processes are grouped in a type of wrappers, e.g, wrapper for collecting data from xml-based weather service, or RDF-based wrapper for extracting Tweet streams. As the processing load of each wrapper type is quite different, some of them can run in a single machine but others have to be distributed across different machines. Whether the data sources are fetched in push-based or pull-based fashion, the output of the processing wrapper is pushed in the stream fashion to the shared bus in RDF format, called, the Stream Graph Data Bus. The stream data in this bus is serialised in RDF-Turtle or JSON-LD to be ingested to the next step.

The processes of feeding data to the Stream Graph Data Bus need to be fault-tolerant as the connection to external data sources are unstable, the data processing wrappers are error-prone. Moreover, the incoming stream throughput is fluctuated

[9] https://dev.twitter.com/overview/api
[10] http://nlp.stanford.edu/software/CRF-NER.shtml
[11] http://cogcomp.cs.illinois.edu/page/run_demo/NER

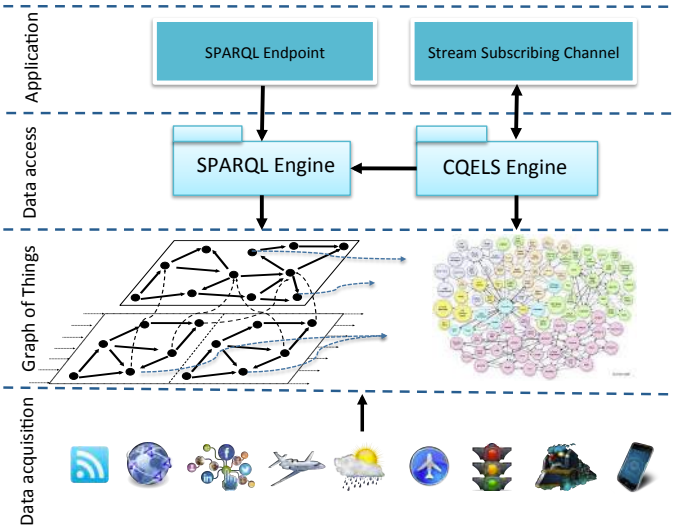


Figure 4: Layered Architecture

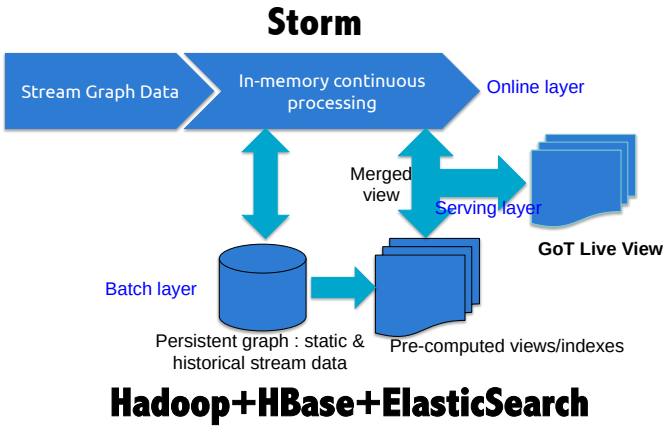


Figure 5: Lambda architecture for GoT data ingestion, publishing and querying

and busty, thus, the architecture of our data management has to be able to serve a wide range of workloads with very low-latency of reads and updates. Due to this requirement of robustness, we realise the Lamda Architecture [26] as the processing flow showed in Figure 5.

All data entering the system is dispatched to both the batch layer and the online layer for processing. The batch layer is responsible for storing the master historical RDF streams (an immutable, append-only converted RDF triples) which are used to pre-compute the batch views specified by RDF Data Cube Vocabulary [12]. Then, in the serving layer these precomputed views are materialised and indexed to store in HBase [13] and ElasticSearch [14] so that they can be queried in low-latency, ad-hoc way. The online layer handled by Storm compensates for the high latency of updates to the serving layer and deals with recent data only. Any incoming query can be answered by merging results from batch views and real-time views using CQELS Cloud Engine [22] which coordinates HBase, ElasticSearch and Hadoop [15] and Storm [16] to handle the SPARQL and CQELS as mentioned in above Section. The architecture aims to be linearly scalable, and it should scale out rather than up, meaning that elasticity is handled by the number of machines added to the system.

3.2. *Query-aware Hybrid Storage*

The Graph of Things is exposed to the user as a single graph, however native triple stores like Virtuoso, Jena TDB could not scale to its number of RDF Triples and speed of updates [20]. According to research shown in [11, 24], the processing in Big RDF Graph could be parallelised efficiently by partitioning the graph into smaller subgraphs to store in multiple processing nodes. Moreover, in designing physical storage to store RDF triples, property tables could reduce subject-subject self-joins of the triples table. They are very good at speeding up queries that can be answered from a single property table, most queries require joins or unions to combine data from several tables [1]. However, using property tables introduces complexity by requiring property clustering to be carefully done to create property tables that are not too wide, while still being wide enough to answer most queries directly. Moreover, ubiquitous multi-valued attributes cause further complexity. Being aware of such advantage and drawbacks of property tables, we design the storage of subgraphs similar property tables by grouping properties by frequent query patterns and distribution of graph patterns, called Query-aware Hybrid Storage.

The sensor readings fed into the GoT have both spatial and temporal context and most of the queries on them contain spatial and temporal patterns. Therefore, we partition the GoT graph based on RDF predicates that imply the spatial and temporal context. We choose Hbase and ElasticSearch as underlying storage for such partitioned subgraphs. Their table structures has flexible data structure which enables us to store subgraphs which share a similar graph shape. This also solve the aforementioned issue of multi-valued properties that is problematic for both traditional property tables and relational tables. Such table structures can store a flexible number of attributes in the same table without having to use list, set, or bag attributes. Moreover, Hbase and ElasticSearch provide spatial and temporal indexing schema that lead to two types indexing mechanisms introduced in following subsections. Other triple patterns that are not able to indexed in these types will be stored in native triple storage. We currently use JenaTDB to store such generic subgraph like static data as they are less than 100 million triples which can be easily loaded into RAM of a standalone workstation for the sake of boosting performance. We envisage that a distributed solution like [11, 24] will be needed in the future when the static part of GoT grows.

The incoming data from Stream Graph Data Bus is routed

[12] http://www.w3.org/TR/vocab-data-cube/
[13] http://hbase.apache.org/
[14] https://www.elastic.co/products/elasticsearch
[15] https://hadoop.apache.org/
[16] https://storm.apache.org/

to destined storages via the data routing flow illustrated in Figure 6. The fetching operations are built as asynchronous tasks, scheduled in the fetching cluster. The GoT engine then evaluates the triple received and buffers the data streams for further processing. Data is routed to Triple Analyser to process based on Triple Patterns Recognition Rules which are predefined by user. The Triple Patterns Recognition Rules is a set of triple patterns defining the rule to extract the values from streaming data. Extracted values will be indexed correspondingly based on their characteristics and graph patterns. The stream subgraphs that have spatial context will be routed to ElasticSearch to index and the ones have time series of numeric values will be router to OpenTSDB cluster. The details of these two indexing mechanisms are presented in below. Otherwise, the stream subgraphs will be stored in the normal triple storage,i,e, JenaTDB.

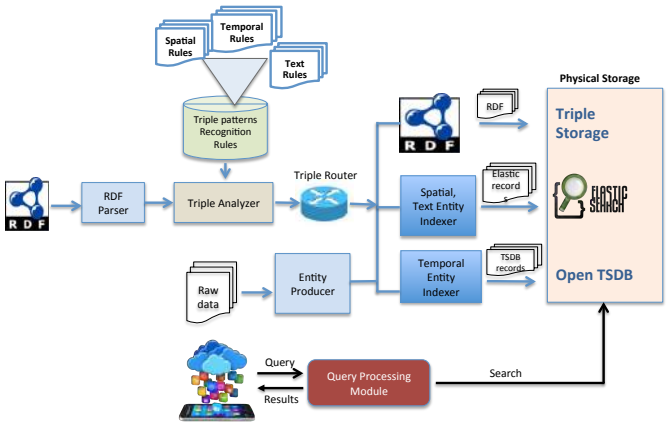


Figure 6: Data Routing Flow

3.2.1. *Spatial-driven Indexing*

To enable spatial computation on spatial properties attached to a subgraph, we store and index such subgraphs as a document in ElasticSearch storage as illustrated in Figure 7. Along with spatial attributes, ElasticSearch allows us to add date-time, string attributes to be indexed on which we can filter by the range condition. That means that a combination of spatial computation, free text search and temporal filter are supported in our SPARQL endpoint (see example queries in Section 4.1).

To build a document to insert to the ElasticSearch storage, the Spatial and Text Entity Indexer will extract RDF triples which comprises geographical information text label, and date-time value (Figure 6). Triples which is filtered by these rules then will be constructed to ElasticSearch record and to store into ElasticSearch storage. Both bulk and near real time (NRT) updates are possible with ElasticSearch. Bulk updates are accomplished using Hadoop Map/Reduce and NRT are performed through direct HTTP calls.

3.2.2. *Temporal-driven indexing*

A large amount sensor stream data of the GoT is fed as time series of numeric values such as temperature, humidity, wind speed. Therefore, we choose OpenTSDB (Open Time Series

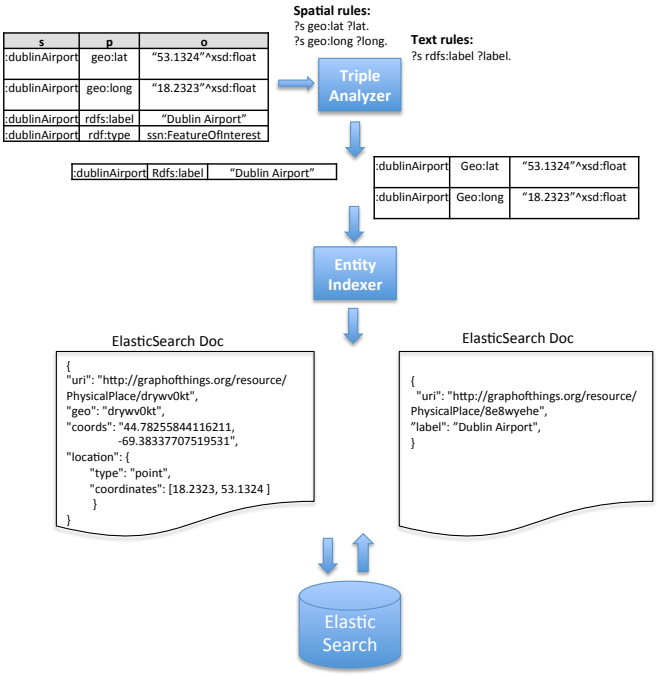


Figure 7: Store spatial subgraphs as ElasticSearch document

Database) which use HBase as underlines scalable database. OpenTSDB can ingest millions of time series data points per second. As shown in Figure 6, input triples which comprises numeric values and timestamps are analysed based on predefined temporal recognition rules in Triple Analyzer. The Entity Indexer extracts numeric measurements to construct time series rows to insert to HBase via parallel Time Series Daemons (TSD) running on different machines of our cluster.

Along with values and timestamps, some metadata can be added to each datapoint of OpenTSDB. Such metadata can be used to filter the values by the information encoded in the metadata. Therefore, we extract spatial context and sensing context of the numeric sensor readings as a temporal subgraph. This graph is used to construct necessary metadata to add to the data points that will be then inserted to OpenTSDB. The metadata is chosen by the frequency used as filtering parameters in the SPARQL queries. For example, the coordinate where the sensor measurement are captured will be used to covert to a geohash value. This geohash value is the used to encode as a filtering tag. Other properties such as type of sensor, type of reading are also encoded as filtering tags.

4. Demonstrations and Lessons Learnt

4.1. *Demonstrations*

Considering GoT as an open linked dataset, we make GoT accessible via SPARQL endpoint at http://graphofthings.org/sparql/. However, this SPARQL endpoint supports 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 ElasticSearch. 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 which is processed by ElasticSearch.

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 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. European 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 trigger the relevant visualisation widgets of without having to interfere other parts of the applications. Until May, 2015, the online system at http://graphofthings/ serves 8.5 billion data entries stored in ElasticSearch and OpenTDB clusters. Among them, transportation data like ship, flight, bus train accounts for 370 million entries over 10 months. As we loaded the NOAA for nearly 40 years, there are nearly 8 billion of meteorological data entries. This data is approximately equivalent to 140 billon triples. The system is still continuously consuming millions of new data entries and we are adding more data sources per month. The underlying setup to serve the such data 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 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 server 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, ElasticSearch 1.5.2, OpenTSDB 2.0 and HBase 0.98.4. A cluster includes 1 master node which has Nimbus, Zookeeper and ElasticSearch and HBase master installed. The other 6 nodes are within the same administrative domain play as ElasticSearch and HBase slaves. All heavy processing pipelines are parallelized using CQELS Cloud parallel execution framework [22]. 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, ElasticSearch and HBase.

Powered this setup, 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 the world" as illustrated in Figure 8. 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 60 minutes, to render information on the map.

For instance, [a] is the heat map of aggregated from temperate readings in last 1 hour in corresponding map areas. The query can be easily expressed by following query.

Query 1. Heat map query

```
PREFIX temporal: <http://jena.apache.org/temporal#>
SELECT *
{ ?v temporal:avg ('1h-ago' 'u0q' 'temperature').
}
```

To retrieve a list of indicated sensor, a query to get a list on a certain type of sensors, e,g. temperature sensor within a bounding box can be expressed as following.

Query 2. Spatial query for sensor discovery

```
PREFIX  spatial: <http://jena.apache.org/spatial#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX dul: <http://www.loa-cnr.it/ontologies/DUL.owl#>
PREFIX was: <http://purl.oclc.org/NET/ssnx/meteo/aws#>
SELECT *
{?loc spatial:withinBox (dul:PhysicalPlace
        67.033 -178.917 67.24 -177.67).
 ?loc geo:lat ?lat.
 ?loc geo:long ?long.
 ?sensor dul:hasLocation ?loc.
 ?sensor a aws:TemperatureSensor;
}
```

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 part [b] of Figure 8. A live thumbnail is a latest snapshot of the stream data sources that are fetched via SPARQL Endpoint. The 3D layout can display much more information than the usual 2D one. For example, the sphere layout in the Figure 8 can render 64 thumbnail charts or even more. This is very useful when exploring and correlating millions of stream data sources. The data to feed into such charts can be retrieved from the SPARQL query similar to the query below. This query retrieve historical data from 64 temperature sensors within 200 miles from the coordinate where the mouse clicked (67.033 -178.917).

Query 3. Query using spatial and temporal search patterns

```
PREFIX spatial:<http://jena.apache.org/spatial#>
PREFIX temporal:<http://jena.apache.org/temporal#>
PREFIX dul:<http://www.loa-cnr.it/ontologies/DUL.owl#>
PREFIX ssn:<http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX cf:<http://purl.oclc.org/NET/ssnx/cf/cf-property#>

SELECT ?sensor ?obs ?value ?time
{
 ?loc spatial:withinCircle (dul:PhysicalPlace
        67.033 -178.917 20.0 'miles' 200).
 ?sensor dul:hasLocation ?loc.
 ?sensor ssn:observes cf:air_temperature.
 ?obs ssn:observedBy ?sensor.
 ?obs ssn:observationResult ?output.
 ?obs ssn:observationResultTime ?time.
 ?output  ssn:hasValue ?value.
 ?value temporal:values ('2015/01/01-03:00'
                         '2015/01/05-09:00').
}limit 64
```

When the user is interested in a detailed data of a snapshot, he/she can click the snapshot to see all relevant historical data

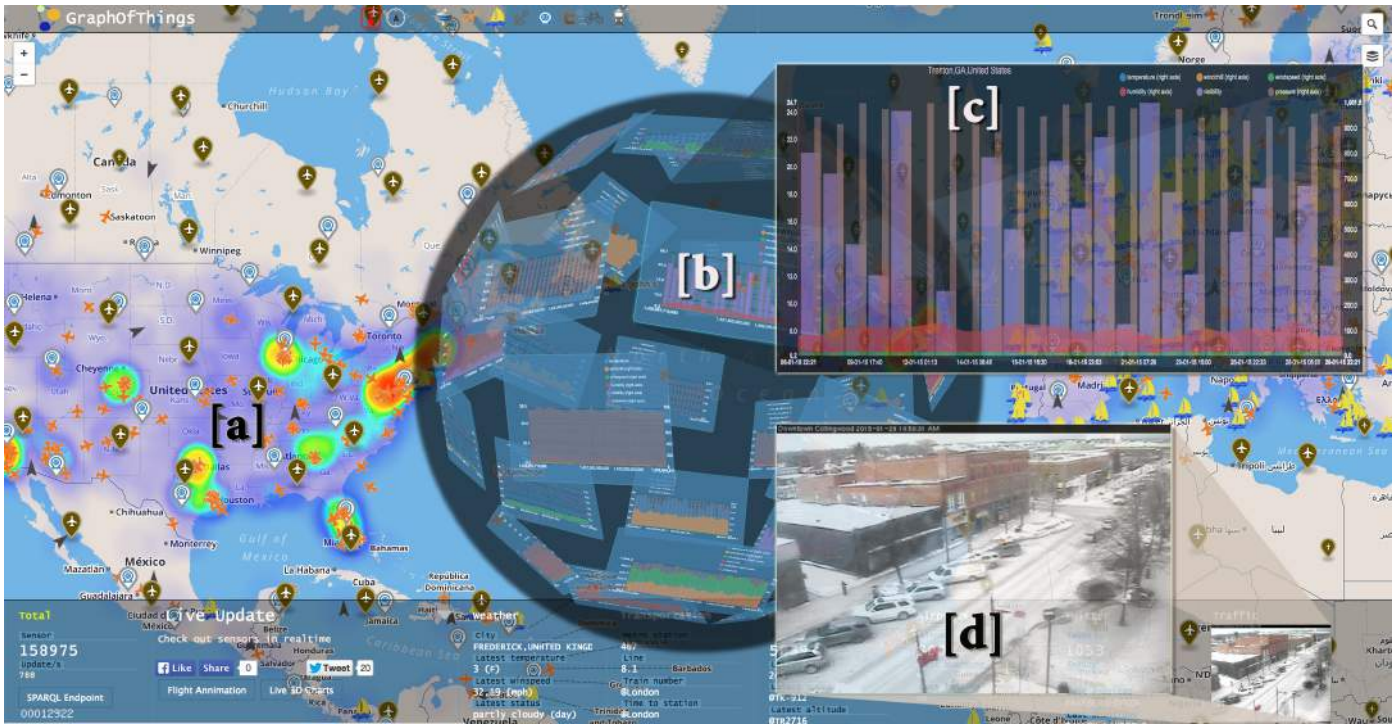


Figure 8: Live View and Visualization of Graph of Things - [a] Heat map, [b,c] 3D layout and chart of historical data, [d] live thumbnails of traffic cameras

stored in GoT. Then corresponding charts of such data are generated by fetching data from GoT SPARQL endpoint using temporal query patterns. For instance, part [c] of the Figure 8 presents the charts of historical weather data from a snapshot of figure [b]. Moreover, the historical data can be used to play back "what happened in the past", e.g. part [d] is playback pop-up of a camera sensor. A play back data can be retrieved to using a time range. Following example query retrieves ship locations from '2015/04/11-03:00' to '2015/04/11-09:00' to play back their movements within an area in the map.

Query 4. Spatial query for ship discovery with time constraint

```
PREFIX spatial: <http://jena.apache.org/spatial#>
PREFIX got: <http://graphofthings.org/ontology/ns/>

SELECT ?vessel
{?vessel spatial:withinCircle (got:Vessel
        '2015/04/11-03:00' '2015/04/11-09:00'
        54.372 -10.1486 20 'miles' 100).
}limit 100
```

To keep the HTML page updated with the data streamed from relevant stream data sources, a Javascript agent of the HTML page registers respective CQELS queries to update the summary of live information in the Live Update Dash board overlayered on the bottom of the map.

4.2. Lessons Learnt

To test the running time performance of the system, we have recorded average query execution time of 4 example queries in Section 4.1 each month since June 2014. Results reported in Figure 9 show that execution time of queries Q1-Q3 slightly increase but take less than 0.5s to response on 8.5 billion data entries or 140 billion triples so far. However, the execution time

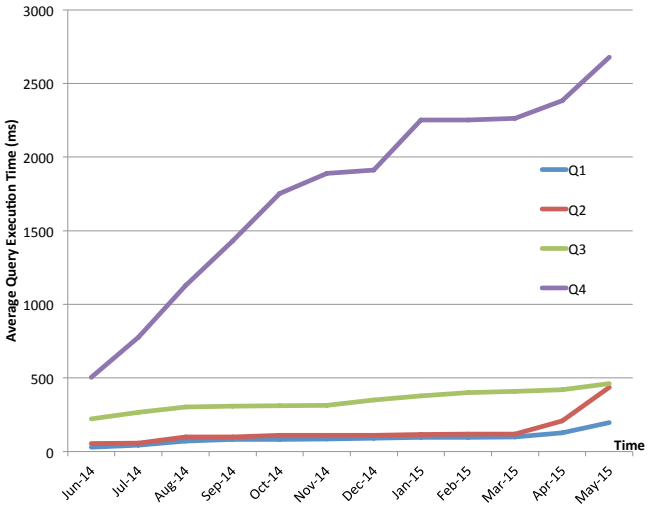
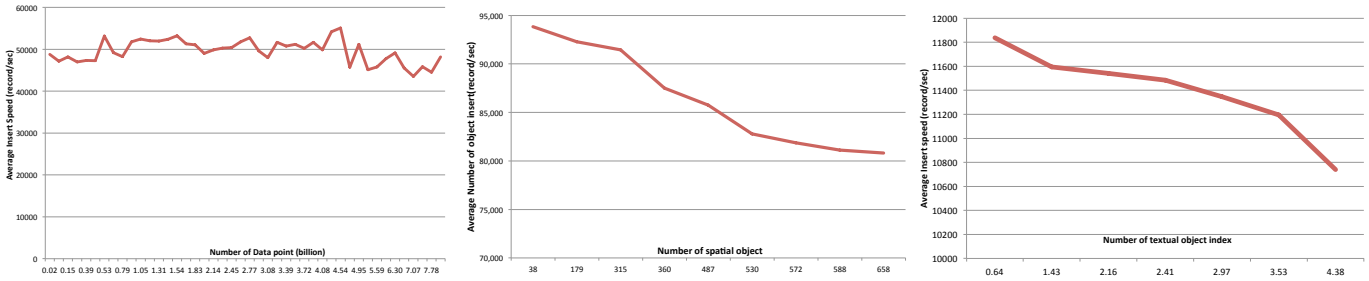


Figure 9: Average query execution time by time collected

of query Q4 has sharply increased from 0.5 seconds to 2.5 seconds. It shows that the more moving objects are added, the considerably heavier work load is pushed to ElasticSearch cluster. However, when the number of time series data are added, the performance of queries with temporal filters are effectively handled by the OpenTSDB cluster.

For the showing the scalability aspect of the system, we have also recorded the maximum indexing throughputs of different types of data, i.e, temporal, spatial and full-text data at certain sizes of data in the storages. The results of Figure 10 show that the indexing throughputs of spatial and full-text decrease



(a) Average temporal indexing throughput (b) Average spatial indexing throughput (c) Average full-text indexing throughput

Figure 10: Indexing throughputs

slightly the ElasticSearch cluster. However, indexing throughputs of time series are quite stable when the data grows even with billions of data entries vs. millions of documents on ElasticSearch. So far, the update throughputs have not hit the limits of system's indexing throughput yet, however, in the long run, more hardware will be needed to added to copy with more ingesting loads and more data archived to the system.

5. Related Work

There is a trend of employing semantic web technologies to solve problems of interoperability problems when integrating heterogeneous and distributed IoT systems. In particular, the Linked Data principles are applied to provide semantic description to the data, sensors and things and also link them to other data sources. Towards this trend, a number of modelling approaches and ontologies used to annotate and describe the IoT data have been developed, such as [31], [13]. OntoSensor [3] constructs an ontology-based descriptive specification model for sensors by excerpting parts of SensorML descriptions and extending the IEEE Suggested Upper Merged Ontology (SUMO) [7]. The work presented in [5] proposes a SSN ontology to describe sensors and sensor networks. The ontology represents a high-level schema model to describe sensor devices, their capabilities, platform and other related attributes in the semantic sensor networks and the sensor Web applications. Another example is the SensorData Ontology developed in [3], which is built based on Observations & Measurements and SensorML specifications defined by the OGC Sensor Web Enablement (SWE) [4].

From ontology-based modelling approaches above, Patni et al. [28] have developed an RDF dataset[17] containing expressive descriptions of 20,000 US weather stations. The dataset contains over 1.7 billion RDF triples and was the first dataset for publishing sensor data as Linked Data. In [12], the authors describes a SenseWeb platform which provides graphical user interfaces to annotate the IoT data using concepts obtained from linked open data cloud (e.g. DBPedia and GeoNames) and also other local domain ontologies. The annotated data is published as RDF triples and is available via a common SPARQL endpoint.

While there are plenty of proposals of publishing sensors data using RDF but they are quite a few of them systematically addressing the issues of expressiveness, performance and scalability of RDF stores used in such systems. There are several complimentary work on support spatial-temporal queries on RDF stores. For example, to enable spatiotemporal analysis, in [29], Perry et al. propose SPARQL-ST query language and introduce the formal syntax and semantics of their proposed language. SPARQL-ST is the extended from SPARQL language to support complex spatial and temporal queries on temporal RDF graphs containing spatial objects. With the same goal of SPARQL-ST, Koubarakis et al. propose st-SPARQL [14]. They introduce stRDF as a data model to model spatial and temporal information and stSPARQL to query against stRDF. These query languages in [29, 14] could process a limited amount of static RDF data, but none of them address the challenge of high update throughput of the IoT data. Moreover, scalability and elasticity for hosting such a massive amount of dynamic data are still challenging issues for the Semantic Web community so far. Our work presented in this article is a new evolution of our series of efforts [18, 21, 19] on managing IoT data together with other related work in the community. In this article, we systematically touched most of aspects of employing Linked Data to enable graph-based search and discovery of IoT.

6. Conclusions and Future work

The article presents a system aiming to build a Knowledge Graph for connected things, the Graph of Things (GoT). The GoT provides a unified graph-based view of data generated by connected things. GoT provides not only sensing data from sensors but understandings the world around physical things, e.g, meaning of sensor readings, sensing context and real world relationships among things, facts and events. GoT has been adding millions of records per hour, roughly more than 10 billion RDF triples per month. GoT puts very first steps towards a graph-based search engine for Internet of Things.

Gearing towards a realtime search engine, we are implementing some ranking algorithms on the graph structure of GoT to

[17] http://wiki.knoesis.org/index.php/LinkedSensorData

enable context-based recommendation features. Subsequently, data quality properties will be added as a metric for such ranking algorithms. Reasoning capability is also a desired feature of GoT but interestingly challenging future goal to achieve in context of big data requirements. To incorporate public IoT data sources with private ones, we intend to extend some state of the art on access control policies for RDF data to GoT.

Acknowledgements

This publication has emanated from research supported in part by Irish Research Council under Grant No. GOIPD/2013/104 and by European Union under and Grant No. FP7-287661 (GAMBAS) and Grant No. FP7-ICT-608662 (VITAL).

[1] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd VLDB*, VLDB '07, pages 411–422. VLDB Endowment, 2007.

[2] Raian Ali, Carlos Solis, Mazeiar Salehie, Inah Omoronyia, Bashar Nuseibeh, and Walid Maalej. Social sensing: When users become monitors. In *Proceedings of the 19th ACM SIGSOFT Symposium*, ESEC/FSE '11, pages 476–479, New York, NY, USA, 2011. ACM.

[3] Barnaghi, P.M., Meissner, S., Presser, M., Moessner, and K. Sense and sensability: Semantic data modelling for sensor networks. In *Proceedings of the ICT Mobile Summit 2009*, 2009.

[4] Mike Botts, George Percivall, Carl Reed, , and John Davidson. Overview and high level architecture. In *GeoSensor networks*, 2008.

[5] Michael Compton, Payam Barnaghi, Luis Bermudez, Ral Garca-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, and Kerry Taylor. The {SSN} ontology of the {W3C} semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17(0):25 – 32, 2012.

[6] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD*, KDD '14, pages 601–610, New York, USA, 2014.

[7] Eid, M., Liscano, R., El Saddik, and A.:. A universal ontology for sensor networks data. In *Proceedings of the IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*, pages 59–62, 2007.

[8] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370, 2005.

[9] Lukasz Golab and Theodore Johnson. Consistency in a stream warehouse. In *CIDR'11*, pages 114–122, 2011.

[10] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. In *Proceedings of the 23rd international joint conference on Artificial Intelligence*. AAAI Press, 2013.

[11] Jiewen Huang, Daniel J. Abadi, and Kun Ren. Scalable sparql querying of large rdf graphs. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '11, pages 1123–1134, 2011.

[12] A. Kansal, S. Nath, J. Liu, and F. Zhao. Senseweb: An infrastructure for shared sensing. *IEEE MultiMedia*, 2007.

[13] Kim, J.-H., Kwon, H., Kim, D.-H, Kwak, H.-y, Lee, and S.-J. Building a service-oriented ontology for wireless sensor networks. In *Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science*, pages 649–654, 2008.

[14] M. Koubarakis and K. Kyzirakos. Modeling and querying metadata in the semantic sensor web: the model strdf and the query language stsparql. In *Proc. Extended Semantic Web Conference*, volume 12, 2010.

[15] Vijay Krishnan and Christopher D Manning. An effective two-stage model for exploiting non-local dependencies in named entity recognition. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 1121–1128, 2006.

[16] Danh Le Phuoc. *A Native And Adaptive Approach for Linked Stream Processing*. PhD thesis, National University of Ireland, Galway, 2013.

[17] Danh Le-Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred 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.

[18] Danh Le-Phuoc and Manfred Hauswirth. Linked open data in sensor data mashups. In *Proceedings of the 2nd International Workshop on Semantic Sensor Networks*, SSN09, pages 1–16, 2009.

[19] Danh Le-Phuoc, Hoan Quoc Nguyen-Mau, Josiane Xavier Parreira, and Manfred Hauswirth. A middleware framework for scalable management of linked streams. *Web Semantics: Science, Services and Agents on the World Wide Web*, 16:42–51, 2012.

[20] Danh Le-Phuoc, Hoan Quoc Nguyen-Mau, Josiane Xavier Parreira, and Manfred Hauswirth. A middleware framework for scalable management of linked streams. *Web Semantics: Science, Services and Agents on the World Wide Web*, 0(0), 2012.

[21] Danh Le-Phuoc, Josiane Xavier Parreira, Michael Hausenblas, Yuanbo Han, and Manfred Hauswirth. Live linked open sensor database. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, pages 46:1–46:4, New York, NY, USA, 2010. ACM.

[22] Danh Le-Phuoc, Hoan Nguyen Mau Quoc, Chan Le Van, and Manfred Hauswirth. Elastic and scalable processing of linked stream data in the cloud. In *The Semantic Web–ISWC 2013*, pages 280–297. Springer, 2013.

[23] Danh Le-Phuoc, Josiane Xavier Parreira, and Manfred Hauswirth. Linked stream data processing. In Thomas Eiter and Thomas 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.

[24] Kisung Lee and Ling Liu. Scaling queries over big rdf graphs with semantic hash partitioning. In *Proceedings of the Very Large Database Endowment*, VLDB '07. VLDB Endowment, 2014.

[25] Anmol Madan, Manuel Cebrian, David Lazer, and Alex Pentland. Social sensing for epidemiological behavior change. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, UbiComp '10, pages 291–300, New York, NY, USA, 2010. ACM.

[26] Nathan Marz and James Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2015.

[27] Meenakshi Nagarajan, Karthik Gomadam, Amit P. Sheth, Ajith Ranabahu, Raghava Mutharaju, and Ashutosh Jadhav. Spatio-temporal-thematic analysis of citizen sensor data: Challenges and experiences. In *WISE'09*, WISE '09, pages 539–553, 2009.

[28] H. Patni, C. Henson, and A. Sheth. Linked sensor data. In *Collaborative Technologies and Systems (CTS), 2010 International Symposium on*, pages 362–370, May 2010.

[29] M. Perry, P. Jain, and A. P. Sheth. Sparql-st: extending sparql to support spatiotemporal queries. *Journal of Geospatial Semantics and the Semantic Web*, 12:25–32, 2011.

[30] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics, 2009.

[31] Russomanno, D.J, Suchanek, Kothari, C., and Klaus Thomas. Sensor ontologies: from shallow to deep models. In *Proceedings of the Thirty-Seventh southeastern Symposium on System Theory*, 2005.

[32] Amit Sheth. Citizen sensing, social signals, and enriching human experience. *IEEE Internet Computing*, 13(4):87–92, July 2009.