



Performance assessment of RDF graph databases for smart city services

Pierfrancesco Bellini^{a,b}, Paolo Nesi^{a,b,*}

^a Distributed Systems and Internet Technology Lab, DISIT, University of Florence, Florence, Italy

^b Department of Information Engineering, DINFO, University of Florence, Florence, Italy



ARTICLE INFO

Article history:

Received 6 November 2017

Revised 8 February 2018

Accepted 11 March 2018

Available online 12 March 2018

Keywords:

Smart city

RDF stores

Graph databases

RDF benchmark

Linked data benchmark

ABSTRACT

Smart cities are providing advanced services aggregating and exploiting data from different sources. Cities collect static data such as road graphs, service description, as well as dynamic/real time data like weather forecast, traffic sensors, bus positions, city sensors, events, emergency data, flows, etc. RDF stores may be used to set up knowledge bases integrating heterogeneous information for web and mobile applications to use the data for new advanced services to citizens and city administrators, thus exploiting inferential capabilities, temporal and spatial reasoning, and text indexing. In this paper, the needs and constraints for RDF stores to be used for smart cities services, together with the currently available RDF stores are evaluated. The assessment model allows a full understanding of whether an RDF store is suitable to be used as a basis for Smart City modeling and applications. The RDF assessment model is also supported by a benchmark which extends available RDF store benchmarks at the state the art. The comparison of the RDF stores has been applied on a number of well-known RDF stores as Virtuoso, GraphDB (former OWLIM), Oracle, StarDog, and many others. The paper also reports the adoption of the proposed Smart City RDF Benchmark on the basis of Florence Smart City model, data sets and tools accessible as Km4City <http://www.Km4City.org>, and adopted in the European Commission international smart city projects named RESOLUTE H2020, REPLICATE H2020, and in Sii-Mobility National Smart City project in Italy.

© 2018 The Authors. Published by Elsevier Ltd.

This is an open access article under the CC BY-NC-ND license.

(<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

1. Introduction

Smart cities produce large amount of data having a large variability, variety, velocity, and size; and thus complexity. The variety and variability of data can be due to the presence of several different formats, [3,17,23,27] and to the interoperability among semantics of the single fields and of the several data sets [6]. Static data are rarely updated, for instance once per month/year, which is quite the opposite with dynamic data: they can be updated from once a day up to every minute so as to get real time data. The data velocity is related to the frequency of data update for dynamic data such as position of buses, flow of people status, position of waste collectors, etc. or as data streams. Thus the size of the store grows over time accumulating new data every day and week. At architec-

tural level, smart city solutions typically adopt n-tier architectures [2].

The Resource Description Framework specified by W3C allows the representation of facts using “triples” of the form (subject, predicate, object) where URIs are used to identify the entities and the predicates connecting them. Thus a triple represents the arc of the graph connecting two entities and the predicate describes the kind of relation between the two entities. Moreover, the object part of the triple can also be a low level data type as string, dates, integers, etc., to describe the relations among entities and specific information about them (e.g., name, email, birth date). RDF stores allow storing triples, while the SPARQL query language allows querying them. Some RDF stores can also manage set of triples as a single graph identified by a URI, in this way information about this graph can be provided using other triples (where the subject is the graph itself). It has to be highlighted that SPARQL performs search using triple matching, for example “*?s rdf: type km4c: Hotel*” searches for all the “*?s*” that are of type Hotel, and does not

* Corresponding author at: Department of Information Engineering, University of Florence, Via S. Marta 3, Firenze 50139, Italy.

E-mail addresses: pierfrancesco.bellini@unifi.it (P. Bellini), paolo.nesi@unifi.it (P. Nesi).

URL: <http://www.disit.org> (P. Bellini), <http://www.disit.org> (P. Nesi)

allow performing, in a general way, the typical graph operations like path search¹ or connectivity check.

The usage of RDF stores in the application domain of Smart City is quite recent, since in most cases services are vertically provided. For example, the Intelligent Transport System, ITS, in the city provides information regarding the location of buses and their delay, without addressing the location of city services, flow of people, real time events in the city. The integrated services are typically provided by data aggregators and service providers that perform data integration and allow exploiting integrated data models. Some city data integrators are well-known services such as bike and car sharing, navigator system, tourism information, hotel booking, etc. All these solutions have the need to integrate geo-located information with real time data and events continuously arriving from updated information such as: events, votes, traffic flows, comments, etc. [11,27,18]. As to these applications, RDF stores may be a solution to allow addressing the variability of data, so as to make reasoning on space, time, and concepts [28]. On this regard, comprehensive smart city data models and ontologies can be really effective (see for example <http://smartcity.linkeddata.es/>, produced by Read4SmartCity² project of the European Commission). One of the ranked models in the Read4SmartCity research project is Km4City ontology that is also used as a reference for the definition of the data of the proposed benchmark [6], and adopted in European Commission international projects named RESOLUTE H2020, REPLICATE H2020, and in Sii-Mobility National Smart City project in Italy, as well as in GHOST MIUR and other projects.

In this paper, requirements and constraints to adopt RDF stores (graph databases) as fundamental store for smart cities data aggregation and services are discussed. This analysis can be used to assess the currently available RDF stores (graph databases) according to a model and benchmark. The proposed assessment model allows understanding if an RDF store can be profitably used as a basis for Smart City modeling and applications. To this end a benchmark is presented, it is a benchmark for linked data, RDF/Graph database/stores with a special care to real structures and relationships available in smart city applications. The proposed benchmark SCIRB overcomes the limits of state of the art benchmark in the sector and it has been defined to compare results from different RDF Stores when they are used for smart city services.

1.1. State of the art and related work about RDF store assessment

For the evaluation of RDF stores, specific assessment models and benchmarks have to be defined and adopted. Some of them may be based on real-world datasets, while others provide a program to generate a synthetic dataset. For example, the LUBM benchmark [16] uses a synthetic dataset in the university domain and covers only the SPARQL 1.0 specification [26]. On the contrary, the BSBM benchmark [9] generates a synthetic dataset in the e-commerce domain and covers the SPARQL 1.1 business analytics queries. More recently, in the Linked Data Benchmarks Council project³ two benchmarks were proposed both generating a synthetic dataset, one from the semantic publishing domain (LDBC-SP) and the other from the social networks domain (LDBC-SN). These benchmarks perform a mix of insert/update/delete and query operations and not only the simple query access. Noteworthy is that the current SPARQL standard does not cover the spatial and keyword searches, thus a query involving these aspects has to be constructed by adopting specific constructs. The GeoSPARQL standard

[15] has been developed by the Open Geospatial Consortium to cover spatial searches, while not many solutions currently support this specification. Regarding the benchmark of geo and spatial RDF stores, the Geographica benchmark [14] was proposed by using both a synthetic generated dataset and a real dataset. It analyses the support and performance for advanced spatial relationships among complex spatial entities (e.g., polygons). In [12], the real and synthetic benchmark datasets have been compared showing that synthetic generated datasets are similar to datasets generated for relational database benchmarks (TPC-H) and strongly different from real-world datasets (e.g., dbPedia) being much less structured. In [25], with the SPARQL Performance Benchmark (SP2Bench) a language-specific benchmark framework designed for the most common SPARQL constructs has been proposed. It also includes a data generator and 17 general benchmark queries, which are unsuitable to assess smart city aspects, and more focused on meta-data.

Recently SPARQL has been extended to query real-time data coming from RDF data streams. Generally, the SPARQL query is performed regularly over the streams using a sliding temporal window and static knowledge data. The query results are provided as a continuous stream. Therefore, the query is registered on the server and clients can connect to receive the results. There are some implementations as C-SPARQL [5], SparqlStream [10], CQELS [21] and also specific benchmarks were defined as SRBench [29] using data from weather sensors, LSBench [22] using data from social networks and CityBench [1] using data from smart city sensors. Those kinds of specific benchmark are suitable for streaming data, with queries performing specific requests with limited number of results. W3C also reviewed RDF store benchmarks⁴ highlighting their applicability in assessing different aspects of the RDF stores, and their application on different stores.

Despite this wide state of the art on RDF stores benchmarks, none of the mentioned approaches is specifically suitable for assessing the RDF stores against Smart City. Smart City presents extremely particular and specific conditions exploiting the latest and most challenging constructs of the RDF stores as geo-spatial queries, text queries, time queries and combinations of them. On this regard, in this paper, Smart City RDF Benchmark, SCIRB, has been proposed particularly to assess RDF store behavior on geo-spatial and full text searches, which are partially considered in other well-known RDF store benchmarks at the state of the art, such as LUBM and BSBM.

1.2. Aims of the paper, and its organization

This paper reports the formalization of the proposed *Smart City RDF Assessment Model and Benchmark* and its adoption in comparative assessment of a number of RDF stores, such as Virtuoso, GraphDB (former OWLIM), Apache Jena-Fuseki, Blazegraph, Stardog, 4store, h2rdf+, Oracle, etc. The data and queries adopted for replicating the mentioned assessment have been published on the following web page <http://www.disit.org/smartcityrdfbenchmark>. The dataset is real and is based on Florence Smart City which in turn is grounded on Km4City ontology and model [6,4].

The paper is structured as follows. In Section 2, the major smart city requirements/demands in modeling and accessing semantic knowledge are reported. The requirements can be used as drivers for features based selection of RDF stores. Section 3 presents the general evaluation methodology for assessing and selecting the RDF stores for smart city applications. It includes an evaluation model, a set of datasets and number of semantic queries in SPARQL. In Section 3.1, the details on the proposed Smart

¹ The SPARQL 1.1 property paths operators may be used to search for a path of properties among entities but it can say only if a path exists among two entities; it is not able to retrieve the actual path.

² <http://www.ready4smartcities.eu/>.

³ <http://ldbouncil.org>.

⁴ <https://www.w3.org/wiki/RdfStoreBenchmarking>.

City RDF Assessment Model; Section 3.2 proposed the Smart City RDF Benchmark and Section 3.3 the corresponding data sets of triples. Section 3.4 proposes a model to analyze the behavior of RDF stores when new upload and update of data are performed. In Section 4, the comparison of most relevant state of the art RDF stores is reported on the basis of the model identified in Section 3.1. Section 5 reports the application of the proposed smart city benchmark in assessing the most featured RDF stores (i.e., Virtuoso, GraphDB, Oracle and StarDog). The analysis has highlighted several interesting aspects connected to the performance of RDF stores in: loading and indexing triples, and in performing geographical and textual queries, also during store updates. In Section 6, the performance of RDF stores when they are used in conjunction with SQL database is proposed, to understand better if either the RDF-SQL-table integration is a profitable solution or it is better to stay on RDF store only. Conclusions are drawn in Section 7.

2. Smart city requirements for RDF stores

When providing services to citizens of a smart city, an RDF/graph store should provide some features that allow the support of specific functionalities. In particular, the following features are reported according to their relevance and classifying them. Therefore, smart city stores should provide support for:

- *spatial indexing (must have)*: providing information near to a given geographical point: as a GPS location. For example, all the services that are currently closed/unavailable to a given point. It should also support advanced geo-spatial functionalities as being able to manage complex geometries (e.g., information along a cycle path, all elements into a given polygonal area).
 - *high performance on spatial querying*.
- *full text indexing (must have)*: allowing the integration of semantic queries with keyword based searches on text which can be present into the attributes and class elements, as triples. Subjects and objects of triples can contain relevant text area such as descriptions, street names, locations names, etc.
 - *high performance on full text querying*.
- *quadruples* (not only triples) to associate dataset metadata with the loaded triples (*must have*). Triples are produced on the basis of data coming from many different sources. Therefore, it is important to track the data source, with metadata and associated licenses. This feature is particularly useful to solve or process licenses during the data usage from clients and via APIs.
- *some kinds of inference* (good to have) such as the basic RDFS or the more advanced OWL2 profiles allowing the inference of new facts from the available data. This may be used to generalize/specialize about entities, to same-as, equivalence, transitive, symmetrical, etc. The inference may imply the materialization of triples in the phase of indexing [7].
- *temporal indexing* (good to have): many information and features are changing over time in smart cities. For example, the weather situation and its related forecast, the traffic flow detected from traffic sensors, the position of buses, and events occurring within the city. For this reason, it is quite important that the RDF store should support temporal search to allow the easy retrieval of temporal data. Moreover, the storing of temporal data (that may change in real time) is the main source for increasing the database size, demanding big data solutions for smart city for volume, velocity and variety, at least.
- *high volume of queries* (good to have). Dealing with bigdata RDF store with many users querying the data is quite challenging, for this reason a clustering solution is needed. It could be a clustering (vertical scale or scale up/down) when the same service is duplicated to allow many concurrent queries and to pro-

vide also a fault tolerance solution. It could be also a scale out clustering (horizontal) when data are split among different servers, as a single server cannot handle all the data.

A very relevant non-functional requirement is due to the fact that when it comes to Smart City applications, they are often exploited by Public Administrations. They ask for: (i) standard solution to avoid the risk of vendor lock-in especially for very new technologies like RDF stores are; (ii) open source solutions to be compliant with typical national laws encouraging open solutions with source code accessible and shareable among several public administrations. Moreover, there should be an active community handling and supporting the product.

3. Evaluation methodology

The *Smart City RDF Assessment Model and Benchmark* evaluation methodology is carried out within two phases. In the **first phase**, the *Smart City RDF Assessment Model* is applied. It consists of an analysis of some general features according to the requirements provided in Section 2, and more particularly to verify if the RDF/graph store provides support for: SPARQL v1.1, inference, triples or quadruples, etc. The successive phase of benchmarking for performance assessment has been carried out only on stores supporting a minimal number of features. In the **second phase**, the *Smart City RDF Benchmark* is applied. It is based on performance tests grounded on a set of SPARQL queries designed by considering all the aspects, and including spatial and full text searches (in many cases the SPARQL queries have been designed by adopting the specific constructs related to the different stores). The execution of the proposed benchmark consists of assessing the performance on the identified queries on three datasets with growing size expanding temporal horizon (1 month, 2 months and 3 months of cumulated real-time data). An additional test is performed to see the behavior of the RDF Store under update queries, when the dynamic data are added to the store.

3.1. Smart city RDF assessment model

According to our requirement analysis for the Smart City RDF assessment model to analyze the RDF stores, the features needed are described as follows. So that the RDF graph database has to provide support for

- SPARQL version 1.0 or 1.1;
- inference type supported as full materialization of triples at load time or materialization at query time, and the inference profiles supported (e.g., RDFS, RDFS+, OWL, OWL2, OWL2-DL, ...);
- triple or quadruple store, check whether it stores only the subject predicate object or it can have also a context URI;
- triples/quadruples physically stored, namely by using a custom indexing or an RDBMS or other external service (e.g., HBase, Cassandra);
- Clustering where replicated nodes are used for high availability and fault tolerant solution;
- Scale Out Clustering where data are allocated on multiple nodes, while no node contains all the data (index sharing);
- Spatial search at Basic level (meaning that it is able to index and retrieve only geolocated points) or at Advanced level (meaning that it is able to index complex shapes, for example polylines);
- full text search, providing the ability to search using keywords;
- the association of triple/quadruples with a temporal validity contexts, thus allowing to easily filter triples by means of temporal constraints;

- size of stores managed as the largest number of triples/quadruples reported to be managed by the RDF store in the literature;
- license under which the RDF store is available, being either open source or commercial;
- development language (e.g., Java, C);
- continuous update: so that if the project is still an active project, date of last activity, date of last release;

Detailed performance testing which should be performed on stores that support minimum set of requirements and in particular providing at least support for:

- SPARQL 1.1 as it provides aggregation functions (group by, count) and other features that were missing in 1.0;
- RDFS inference at load time or query time;
- Quadruples, so that correct metadata can be associated with datasets;
- basic spatial search to allow searching services via position;
- full text search to be able to integrate keyword search with semantic search;
- “Big stores” management in somehow: that is the capability of managing large data store with some technique, scaling for instance.

If the RDF store supports additional features, they are positively considered in the context.

3.2. Smart city RDF benchmark

In this section, the queries at the basis of the *Smart City RDF Benchmark* are presented. The queries performed over the datasets are mainly the ones behind a real Smart City application and the API adopted in Km4City and used in <http://servicemap.km4city.org>. ServiceMap is an accessible open source smart city web application for developers to develop informative totems, while the Km4City API is a set of services accessible from Smart City mobile app delivered on all the available platforms: Apple Store, Google Play, and Windows Market.

It should be noted that the SPARQL recommendation does not cover the geo-spatial queries, nor the full-text queries. Therefore, in order to support those features, RDF store builders/vendors implemented these features by using their own specific syntax. For these reasons, for some queries there is not a unique formulation and the query has to be adapted for each RDF store under test (they can be accessed from the web page of the proposed benchmark <http://www.disit.org/smartcityrdfbenchmark>). In Table 1, the semantic queries at the basis of the *Smart City RDF Benchmark* have been described highlighting: (i) if the query depends on some randomly chosen parameter; (ii) if it uses inference; (iii) if it uses geo-spatial search operators; (iv) if it uses full-text search operators; (v) if it uses some sub-query; (vi) if it use the SPARQL 1.1 “group by” feature; (vii) if it uses the UNION operator; (viii) if the results are using the “order by” construct; and (ix) the number of triples matching constraints used in the query (it is a size of the complexity of the search pattern). The sub-query, group-by, union and order-by impact in the query execution time depending on the query optimization capabilities (e.g., parallel unions, ordering or grouping using indexes, etc.), and the number of triples constraints gives a measure of query complexity. The queries selected for the benchmark are able to test different aspects of a SPARQL engine and provide a wide spectrum of queries complexities, ranging from very simple queries like *Bus-lines* to the *Bus-stop-forecast*.

For example, the query to retrieve the last weather forecast for the municipality of Florence (*Weather-florence*) is the following:

```
PREFIX ...
SELECT ?day ?desc ?minTemp ?maxTemp ?time
?wPred WHERE {
```

```
{
  SELECT DISTINCT ?wRep ?time WHERE {
    ?munic rdf:type km4c:Municipality;
    foaf:name "FIRENZE";
    km4c:hasWeatherReport ?wRep.
    ?wRep km4c:updateTime/schema:value ?time.
  } ORDER BY DESC(?time) LIMIT 1
}
?wRep km4c:hasPrediction ?wPred.
?wPred dcterms:description ?desc;
km4c:day ?day;
km4c:hour "giorno"^^xsd:string.
OPTIONAL {?wPred km4c:minTemp ?minTemp.}
OPTIONAL {?wPred km4c:maxTemp ?maxTemp.}
}
```

The above query, uses a sub-query to find the last weather report received related to the municipality of Florence and from this report the prediction associated is selected and the associated information is returned. This query was selected because it is used to test the sub-query capability and the efficiency of sorting the results to get the most recent.

A query using full text search and geospatial proximity search (*Service-text-latlng*) using the syntax of virtuoso, is:

```
PREFIX ...
SELECT DISTINCT ?ser ?elong ?elat ?sTypeIta
WHERE {
  ?ser ?p ?txt.
  ?txt bif:contains "casa".
  {
    ?ser km4c:hasAccess ?entry.
    ?entry geo:lat ?elat;
    geo:long ?elong;
    geo:geometry ?geo.
    filter(bif:st_intersects(?geo,
      bif:st_point(11.26193046,43.77072194), 0.5))
  } UNION {
    ?ser geo:lat ?elat;
    geo:long ?elong;
    geo:geometry ?geo.
    filter(bif:st_intersects(?geo,
      bif:st_point(11.26193046,43.77072194), 0.5))
  }
  ?ser a ?sType.
  FILTER(?sType!=km4c:RegularService &&
    ?sType!=km4c:Service)
  ?sType rdfs:label ?sTypeIta.
  FILTER(LANG(?sTypeIta)="it")
}
```

This query was selected because it mixes the full-text and geospatial searches with the UNION capability which can be implemented in different ways (e.g., parallel execution). As it occurs with all the RDF benchmarks, the SPARQL queries are specifically tuned for a model. In this case, queries have been designed for the model described in the next section. The complete formalization of the queries, as well as the dataset dumps adopted in the tests reported hereafter, are available at <http://www.disit.org/smartcityrdfbenchmark>

3.3. Datasets of the smart city RDF benchmark

The data used for the evaluation are based on the Km4City knowledge base [6]. The Km4City knowledge base models many aspects of a smart city: structural, energy, mobility, services, administrative, environment, etc. Some of them are static (or quasi-static) data such as (i) the *road graph* modeling the roads, the public administrations; etc. (ii) the “services” available within the city

Table 1
Queries of Smart City/RDF Benchmark.

Query	Description	Parametric	Inference	Geo-spat.	Full-text	Sub-query	Group-by	Union	Order-by	N. triples constraints
Find-address	given the latitude and longitude position it retrieves the nearest address within 100 m.	Y	N	Y	N	N	N	N	Y	10
Municipalities-florence	It retrieves the list of municipalities within the province of Florence.	N	N	N	N	N	N	N	Y	4
Bus-lines	It retrieves the list of bus lines.	N	N	N	N	N	N	N	Y	2
Bus-stops-of-line	given the bus line, it retrieves the complete bus stop list of the line.	Y	N	N	N	N	N	N	Y	8
Lines-of-bus-stop	given a bus stop, it retrieves the lines going past that bus stop.	Y	N	N	N	N	N	Y	Y	8
Bus-stop-latlng	given a position and a radius, it finds the bus stops that are within the radius.	Y	N	Y	N	N	N	N	N	5
Bus-stop-florence	It retrieves all the bus stops in the municipality of Florence.	N	N	N	N	N	N	N	N	6
Bus-stop-forecast	given a bus stop, it finds the next forecasts for the lines going past that bus stop.	Y	N	N	N	Y	Y	N	Y	14
AVM-distribution	It retrieves for each day the count of the received AVM records.	N	N	N	N	N	Y	N	Y	2
Service-florence	It retrieves all the services in the municipality of Florence.	N	Y	N	N	N	N	N	N	23
Service-Acc-Clt-Trs-W&F-florence	It retrieves all the services in the Accommodation, Cultural Activity, TourismService and Wine&Food classes within the municipality of Florence.	N	Y	N	N	N	N	Y	N	27
Service-Htl-B&B-florence	It retrieves all the services in the Hotel and Bed&Breakfast classes within the municipality of Florence.	N	Y	N	N	N	N	Y	N	25
Service-latlng	It retrieves the services within a radius from a latitude, longitude position.	Y	Y	Y	N	N	N	Y	N	16
Service-Acc-Clt-Trs-W&F-latlng	It retrieves all the services in the Accommodation, Cultural Activity, TourismService and Wine&Food classes within a radius from a position.	Y	Y	Y	N	N	N	Y	N	20
Service-Htl-B&B-latlng	It retrieves all the services in the Hotel and Bed&Breakfast classes within a radius from a given position.	Y	Y	Y	N	N	N	Y	N	18

(continued on next page)

Table 1 (continued)

Query	Description	Parametric	Inference	Geo-spat.	Full-text	Sub-query	Group-by	Union	Order-by	N. triples	constraints
Full-text	It retrieves anything matching a keyword	Y	N	N	Y	N	N	Y	Y	10	
Service-text-florence	It retrieves all the services in the municipality of Florence matching a keyword.	Y	Y	N	Y	N	N	Y	N	20	
Service-text-latlng	It retrieves all the services matching a keyword given a position and a radius.	Y	Y	Y	Y	N	N	Y	N	11	
Sensor-florence	It retrieves all the sensors within the municipality of Florence.	N	N	N	N	N	N	N	N	8	
Sensor-latlng	It retrieves all the sensors within a radius from a position.	Y	N	Y	N	N	N	N	N	6	
Sensor-status	It retrieves the latest information associated with a sensor.	Y	N	N	N	N	N	N	Y	13	
Sensor-distribution	It finds for each day the count of the received sensor status updates.	N	N	N	N	N	Y	N	N	3	
Parking-status	It retrieves the latest information associated with a parking lot.	Y	N	N	N	N	N	N	Y	10	
Parking-distribution	It retrieves for each day the count of the acquired parking status records.	N	N	N	N	N	N	Y	Y	3	
Weather-florence	It retrieves the latest forecast available for the municipality of Florence.	N	N	N	N	Y	N	N	Y	11	
Weather-distribution	It retrieves for each day the count of the acquired weather forecasts.	N	N	N	N	N	N	Y	Y	3	

Table 2
Datasets characterization for Smart City Benchmark.

Type	1 month		2 months		3 months	
	Triples	%	triples	%	triples	%
AVM	8.4M	19%	18M	33%	28M	43.1%
Parking	413k	0.9%	976k	1.8%	1.4M	2.1%
Sensors	900k	2%	1.7M	3.1%	2.2M	3.3%
Weather forecast	15k	0%	23k	0%	23k	0%
Total dynamic	9.7M	22%	21M	38%	32.5M	48.5%
Road graph	33.5M	75%	33.5M	60.3%	33.5M	50%
Services	681k	1.5%	681k	1.2%	681k	1%
Other static	286k	0.6%	286k	0.5%	286k	0.4%
Total static	34.5M	78%	34.5M	62%	34.5M	51.4%
Total	44.2M	100%	55.6M	100%	67.5M	100%

(e.g., restaurants, hotels, cycle paths, ...) and associated with the road graph and organized in an hierarchy; (iii) the bus stops, bus lines of the local transportation, (iv) the road sensors available on the roads.

Moreover, dynamic information changing over time is modeled as well and namely: (i) the weather forecasts for the different municipalities; (ii) the status/position of the bus with possible forecasts on the arrival to bus stops; (iii) the status of the parking lots (e.g., number of available parking space); (iv) the readings of traffic sensors; (v) the events defined within the city. Km4City model and data instances are representative of many smart city solutions since it includes multi domains and not only a few of them [2,27]. Moreover, the Km4City model provides a number of hierarchies and structures, and huge data with geolocations in which the inferential aspects of SPARQL queries can be profitably tested.

The testing datasets, comprised of quadruples, have been generated on the basis of Km4City model by using data from the Florence smart city service. The testing dataset is divided in 72 RDF graphs each of is identified by an URI (e.g. http://www.disit.org/km4city/resource/GeolocatedObject/Wifi_kmz). Each RDF graph is generally associated with a dataset provided by an institution/department (e.g., the municipality of Florence) with its own provenience and licensing information.

Three different datasets have been adopted for the assessment. They share the same ‘static’ information and only differ for the dynamic part, having one, two or three months of historical dynamic data, respectively. In Table 2, the numbers of triples for the different parts of the Km4City knowledge base are reported. As you can see, the dynamic parts grow from 22% to 48.5% mostly derived from the AVM (automatic vehicle monitoring, of the ITS) that it is generated out of the data coming for only three bus lines, while the static part is mostly based on structural data like road graph with 34.5 M triples, in all the cases.

3.4. Real-time data set context description

Since in a real context the dynamic data change regularly (e.g., weather status, AVM, sensors and parking), the behavior of the RDF stores should be analyzed also under dynamic conditions like queries, while other processes are performing update/upload. Moreover, in order to test a more realistic case the queries retrieving the last value of dynamic data (e.g., sensor last value) could be arranged by using a model including triples stating, which is the latest obtained value. In this case, a SPARQL query should be used to remove the association with the latest received value and insert the new triple associated with the new reading of values. This query can be regarded as an update query.

To analyze performance on dynamic update conditions a specific test case has been set up (e.g., traffic, IOT). In order to establish replicable conditions, a tool has been used to regularly generate the status of the 430 sensors using the NTriples format (stored

in a specific context) as standard SPARQL 1.1 Graph Store HTTP Protocol. They are produced and singularly loaded into the store, together with their association with the latest value to the corresponding sensor. Each submission stores 19 triples for each sensor and thus 8056 new triples are stored about every 30 seconds. In this case, the 3 months dataset of Table 2 has been used. Together with the process of upload/update, the server runs at the same time all the queries of the benchmark to assess if updating the triples while querying, either influences or not the query time. Moreover in this case the “sensor-status” query has been changed in order to exploit the “latest value” triple. This kind of benchmark can stress the critical cases of RDF stores, as for multiple reasons the upload/update of value may involve: (i) indexing of triples and text, (ii) inference materialization of triples and then indexing, (iii) the lock of the store to perform changes, thus any process delay or unacceptance to perform regular queries on the base data.

4. Comparing RDF stores with smart city RDF assessment model

In this section, the RDF stores under assessment are compared according to the feature model which has been identified and discussed in Section 3.1. The comparison is carried out with the aim of identifying the stores that are better ranked to be used on smart city applications in terms of provided features. In Table 3, the features supported by the different RDF stores under evaluation are summarized and the values considered as minimum requirements are highlighted. A description of the RDF stores considered in the assessment and reported in Table 3 is given below.

- **Virtuoso 7.2.4** [13], it is mostly known because it is the RDF query engine behind dbpedia.org. It is a SPARQL 1.1 quadruple store developed in C available both via open source and commercial license. The open source version mainly misses the clustering feature. Inference is not materialized at load/indexing time, while query rewrite is performed to support RDFS+ inference. It is backed by the Virtuoso RDBMS and thus SPARQL queries are translated to SQL for that RDBMS. It supports advanced spatial indexing (using RTrees) and supports full text search. The community behind virtuoso is headed by OpenLink Software Ltd and it is quite active.
- **GraphDB SE 7.0.1** (former OWLIM store)⁵ is a commercial solution providing a SPARQL 1.1 endpoint supporting triple/quadruple stores with spatial indexing of geographic coordinates and full text indexing based on Lucene, Apache. It supports inference at load/indexing time with different rule sets (RDFS, OWL2RL, etc.), and such rule sets can be selected by the user. The Enterprise edition allows horizontal scaling with a master node forwarding the insert/update/delete operations to slave nodes. Each single node may support up to 10 billion of triples. The solution is implemented in Java using OpenRDF Sesame. The project is still active and it is managed by Ontotext.
- **Blazegraph** (ex BigData)⁶ is an open source project, providing also a commercial license. It supports triple and quadruple stores. With RDFS+ inference (at load time) it is available only on triple stores. It has a full-text indexing support, and there is a basic geospatial indexing, too. It provides both a horizontal and vertical scaling solution, thus allowing an index to be shared on multiple nodes. A single computer can manage up to 50 billion triples. The project is managed by Systap and it is still active.

⁵ <http://ontotext.com/products/ontotext-graphdb/>.

⁶ <https://wiki.blazegraph.com>.

Table 3

RDF stores' features comparison, where:

OS = Open Source, Cm = Commercial, H = Horizontal cluster, V = Vertical cluster.

RDF Store	Features											
	SPARQLv.	Inference	3/4-uple	Spatial search	Full-text search	Storage	Temporal search	Size (million/billion triples)	License	Dev. Language	Cluster support	Active project
Virtuoso 7.2.4 OS	1.1	RDFS+	4	Adv	Y	RDBMS	N	50BT	OS	C	N	Y
Virtuoso 7.2.4 Comm	1.1	RDFS+	4	Adv	Y	RDBMS	N	50BT	Cm	C	H	Y
Graph DB SE 7.0.1	1.1	OWL2RL	4	Bas	Y	custom	N	10BT	Cm	Java	H	Y
Stardog 4	1.1	OWL2	4	Adv	Y	custom	N	10BT	Cm	Java	H	Y
Oracle 12c	1.1	RDFS, OWL2	4	Adv	Y	custom	N	1TT	Cm	C/Java	H	Y
Apache Jena-Fuseki	1.1	RDFS OWL-Lite	3	Bas	Y	custom (TDB)	N	1.7BT	OS	Java	N	Y
Apache Jena-Fuseki	1.1	No	4	Bas	Y	custom (TDB)	N	1.7BT	OS	Java	N	Y
Blazegraph 2.1.2	1.1	RDFS+	3	Bas	Y	custom	N	50BT	OS	Java	V&H	Y
Blazegraph 2.1.2	1.1	No	4	Bas	Y	custom	N	50BT	OS	Java	V&H	Y
CumulusRDF	1.1	No	3	No	N	Cassandra 1.2	N	120MT	OS	Java	V	(Y)
Strabon	1.1	No	3	Adv	N	RDBMS	Y	500MT	OS	Java	N	(Y)
4store	1.1	No	4	No	N	custom	N	15BT	OS	C	V	N
h2rdf+	1.0	No	3	No	N	HBase	N	2.7BT	OS	Java	H&V	N

- **CumulusRDF** [20] is an open source project based on OpenRDF Sesame using Apache Cassandra 1.2 as NoSQL storage layer. It does not support inference and can store only triples. Since it is based on Cassandra, it supports vertical scaling for storage of the indexes on the nodes in the cluster, while only one node is used to perform queries.
- **Stardog 4.1.1**⁷ is a commercial RDF quadruple store developed by Clark&Parsia (developer of the well-known OWL reasoner Pellet). It supports SPARQL 1.1 and OWL2 inference at query time, full-text indexing and search, and spatial indexing and search. It allows horizontal scaling, and it is a quite active project. Stardog may support 10 billion triples store on single node while the community version manages up to 25 million triples.
- **Strabon** [19] is an open source SPARQL 1.1 store developed to support both spatial and temporal search [8]. It is based on PostGIS extension of Postgres RDBMS; it does not support inference, nor full-text search. It only provides support for storing triples (the context URI associated with the triple is used for temporal linking). No clustering solution is available.
- **4store**⁸ is an open source quad RDF store developed in C supporting a clustering solution which stores the quads on different nodes (max 32). It does not support any inference, any

full-text search, nor geospatial search. The activity seems to be moved to 5store, which is a corresponding commercial version.

- **h2rdf+** [24] is an open source triple store based on HBase and Hadoop platform. It supports only the SPARQL 1.0 specification, and does not support any inference, any full-text indexing, nor geo-spatial search. Being based on HBase and Hadoop, it provides horizontal and vertical scaling.
- **Apache Jena-Fuseki 2.3.1**⁹ is an open source SPARQL 1.1 engine integrated within the java based Apache Jena framework. Jena provides the quads RDF storage layer which could be native on file system (TDB), based on a SQL DBMS (SDB) or in memory. Jena provides also the inference support (supporting RDFS, OWL-Lite or using custom rules) but it works only on triple stores and not on quadruples stores, moreover it supports full-text and basic spatial indexing based on Lucene or Solr. No clustering solution has been reported.
- **Oracle Database 12c**, the well-known Oracle database solution provides support for RDF graphs, full-text & spatial indexing/search but it does not support the standard SPARQL HTTP query protocol, it can be integrated by using the open source Jena framework with Fuseki or Joseki tools. Moreover Oracle solution provides inference (RDFS, OWL2RL and custom rules).

As a conclusion of this section, it is evident from Table 3, that the RDF store solutions supporting all the minimum requirements

⁷ <http://stardog.com/>.

⁸ <http://4store.org/>.

⁹ <https://jena.apache.org>.

Table 4

RDF stores performance of data loading.

“NA” means that the information is *not available* (impossible to measure).

	Triples load time	Stated triples	Stored triples	Size (of which: full text index size, spatial index size)
GraphDB – 1 month	1 h 8m	44,274,756	84,425,185	8.5GB (299 MB, 66 MB)
GraphDB – 2 months	1 h 48m	55,617,333	104,041,312	10GB (379 MB, 67 MB)
GraphDB – 3 months	2 h 10m	67,082,202	124,015,329	13GB (459 MB, 70 MB)
Virtuoso – 1 month	16m	44,274,820	46,259,439	2.2GB (NA, NA)
Virtuoso – 2 months	21m	55,619,789	57,669,629	2.8GB (NA, NA)
Virtuoso – 3 months	31m	67,084,661	69,200,459	3.5GB (NA, NA)
Stardog – 1 month	1 h 19m	44,273,368	44,273,368	4.8GB (341 MB, 131 MB)
Stardog – 2 months	1 h 24m	55,615,945	55,615,945	5GB (318 MB, 129 MB)
Stardog – 3 months	2 h 58m	67,080,814	67,080,814	6.2GB (493 MB, 138 MB)
Oracle – 1 month	6 h 18m	44,270,460	78,744,647	25GB (NA, NA)

are Virtuoso 7.2.4 open source and commercial edition, GraphDB Standard Edition 7.2, Stardog 4 and Oracle 12c. Therefore, only these RDF stores have been assessed in term of performance, as reported in Section 5. Apache Jena-Fuseki has been considered in Section 6, when the performance of RDF stores together with SQL databases is evaluated.

5. Assessing RDF stores with smart city RDF benchmark

The performance evaluation has been carried out by considering: (i) the loading/indexing time for knowledge base initialization, (ii) the execution time without any update for spatial and non-spatial queries, and (iii) query execution time while the sensors data were regularly updated.

The performance has been evaluated using a server Ubuntu 14.04 with 8GB RAM, CPU, Intel Xeon E5-2680@2.8GHz with 20 logical processors, HD at 15,000 RPM.

5.1. Assessing loading/indexing

Table 4 reports the results for the loading/indexing time concerning the different previously discussed datasets, respectively. It should be remarked that Virtuoso is the fastest, GraphDB and Stardog perform similarly (about 5 times slower than Virtuoso) and Oracle is the slowest being about twenty three times slower than Virtuoso. Due to the performance of Oracle 12c in loading, the decision was to only test the 1 month data set. On the other hand, GraphDB and Oracle perform inference at load time while Virtuoso and Stardog at query time, under user request. For this reason, the number of triples indexed by GraphDB is typically 80% bigger than those of Virtuoso. As to Virtuoso, the slight increment of triples stored/indexed with respect to the ones provided to the RDF store (2.1 M for the 3 months case) is due to the transformation of the geo:lat and geo:long triples in a geo:geometry with POINT() to enable the geo-spatial indexing. While in the same case, as to GraphDB, the increment of about 57M of triples is due to the materialization of triples via inference at the indexing/loading time.

5.2. Assessing query execution time

Tables 5 and 6 focus on the results for the average query execution time concerning GraphDB, Virtuoso, Stardog and Oracle and related to the different time horizons of one, two and three months, respectively. Table 5 reports the performances for non-spatial queries and Table 6 for spatial queries. The queries have been tested performing a pseudo-random sequence of 1000 queries repeated two times with some pseudo-random arguments in order to reduce the caching effect. The sequence of performed queries has been the same for each test execution, so as to test the same sequence on different systems. The table reports for each query the average response time and the maximum number of

results obtained for each type of query when the number of results depends on the parameter randomly chosen (e.g., lines of a bus stop) or from the different dataset used (e.g., the AVM, sensor, parking and weather distribution queries). When considering the poor performance by Oracle 12c in loading and also in the query times, it was decided to only test the 1 month case. Moreover, a bug in the Oracle plugin for Apache Jena did not allow performing spatial queries via the HTTP protocol and this is the reason why Oracle 12c does not appear in Table 6.

If observing the query results (see Table 5), when no spatial and full text search and inference are involved, the performances of Virtuoso and GraphDB are comparable, in some cases GraphDB is even better ranked. When inference is needed (e.g., in the test cases *Service-florence*, *Service-Acc-Clt-Trs-W&F-florence*, *Service-Htl-B&B-florence*), as to Virtuoso the inference had to be enabled on the single constraint involving a general class (e.g., all services in the Accommodation class). While if the inference is enabled, generally on the query, the internal automated query rewrite takes a longer time (may be related to the size of the exploited ontology). For example, for query *Service-Acc-Clt-Trs-W&F-florence* the time grows from an average of 2.9 s to an average of 19.6 s (on the 3 months dataset). In those cases, the GraphDB results are better ranked. Stardog generally is the slowest on all the queries.

When considering the spatial indexing (see Table 6) in Virtuoso, some mistakes have been detected using the *st_intersection* function. In some cases, Virtuoso returned an error, in other cases it provided a smaller number of results than the correct number; Virtuoso could provide different results for the same query for the three different datasets, even if they do not differ for the part considered in the query. On the other hand, in Virtuoso, if the *st_distance* function is used, all the obtained results have been verified to be correct, apart from few cases on the border (due to the numerical computation in measuring distances). The usage of the distance function for Virtuoso is a good solution in most cases, while the query optimizer seems to avoid the exploitation of the spatial index. This fact may be deduced out of a comparison among the results of the formalization of query *Find-address*: in two cases by using: (i) *st_distance* function it takes about 5.7 s, while (ii) with *st_intersect* function it takes about 0.14 s.

Another aspect to be considered is the mixing of spatial query with text search query (for example, in query *Service-text-latlng(500m)*). With GraphDB and also with Stardog, we obtained a higher execution time, hitting in some cases the timeout. In this case where spatial and text are combined for Virtuoso, the intersect function returned an error, while the distance function performed very well.

Regarding the analytic queries (for example: *Weather-distribution*, *AVM-distribution*) which count the daily number of records of the weather forecasts, bus, sensor data, parking status for the three datasets, both solutions have provided acceptable execution time (less than 5 s). In this case, Virtuoso is better

Table 5
RDF stores average query time of non-spatial queries (best performances in bold).

Query	GraphDB			Virtuoso			StarDog			Oracle	
	1 month (ms)	2 months (ms)	3 months (ms)	1 month (ms)	2 months (ms)	3 months (ms)	1 month (ms)	2 months (ms)	3 months (ms)	1 month (ms)	Number of results
Municipalities-florence	7	10	121	8	15	9	127	173	129	2,391	46
Bus-lines	17	18	91	6	7	6	125	156	141	2,325	85
Bus-stops-of-line	50	26	28	65	68	62	194	211	172	36,661	135 (max)
Lines-of-bus-stop	7	12	18	21	23	20	210	235	210	6457	11 (max)
Bus-stop-florence	100	113	126	374	291	281	216	258	201	34,071	1108
Bus-stop-forecast	96	413	444	632	2065	2008	2028	3072	5084	259,577	15
AVM-distribution	914	1893	2767	26	58	70	1442	2417	3772	26,844	89 (max)
Service-florence	7106	7841	10,150	2170	2135	2158	3689	3667	3514	> 10min	3259
Service-Acc-Clt-Trs-W&F-florence	8,158	8274	8318	2386	2917	2930	4118	4110	6416	> 10min	1179
Service-Htl-B&B-florence	3311	3296	4,035	537	845	766	3640	3782	3448	> 10min	234
Full-text	314	750	618	64	96	67	166,202	214,344	215,937	136,243	1389 (max)
Service-text-florence	286,842	295,057	284,573	1981	3621	5661	165,860	202,919	209,364	126,833	51 (max)
Sensor-florence	21	48	46	82	93	84	785	615	483	7349	62
Sensor-status	598	1101	1560	56	146	163	295	384	392	173,612	1
Sensor-distribution	939	1867	2665	174	328	399	672	1060	1,346	178,272	78 (max)
Parking-status	83	188	309	72	87	100	1,388	1339	1053	40,823	1
Parking-distribution	455	1096	1628	61	131	203	223	373	451	30,444	83 (max)
Weather-florence	9	19	93	46	60	71	181	182	149	5047	5
Weather-distribution	12	23	19	7	18	11	126	141	128	2342	38 (max)

Table 6
RDF stores average query time of spatial queries (the best performances in bold).

Query	GraphDB			Virtuoso (intersect)			Virtuoso (distance)			Stardog			Number of results
	1 month (ms)	2 months (ms)	3months(ms)	1 month(ms)	2 months(ms)	3months(ms)	1 month(ms)	2 months(ms)	3months(ms)	1 month(ms)	2 months(ms)	3months(ms)	
Find-address	47	180	218	219	160	143	5762	5965	5776	2495	2848	2367	1
Bus-stop-latlng(100 m)	8	8	9	33	63	63	28	31	28	2105	1791	1885	1 (max)
Bus-stop-latlng(200 m)	16	17	26	88	80	23	29	27	30	1820	1723	1638	3 (max)
Bus-stop-latlng(500 m)	33	52	48	147	182	166	34	41	33	1781	1853	1810	20
Bus-stop-latlng(1 km)	82	135	155	76	–	265	42	43	47	2095	2116	2334	93 (max)
Bus-stop-latlng(2 km)	192	302	361	89	–	–	77	77	76	2418	2822	2787	306 (max)
Bus-stop-latlng(5 km)	463	669	782	–	–	–	201	201	205	2883	3245	2815	1003 (max)
Service-latlng(100 m)	691	1111	324	2788	2117	915	582	761	754	3970	4581	5123	41 (max)
Service-latlng(200 m)	189	393	364	680	462	357	271	308	287	3679	4230	3846	130 (max)
Service-latlng(500 m)	1131	1256	1212	627	513	549	401	421	391	4110	4303	4467	784 (max)
Service-latlng(1 km)	2924	3236	3205	1566	1085	1238	613	602	609	4406	5475	5492	1810
Service-latlng(2 km)	6087	6550	6548	1377	1688	–	1062	1167	1079	5832	6929	6204	3718 (max)
Service-latlng(5 km)	11,192	13,069	12,712	3054	–	–	1900	1996	1912	6585	7494	6551	6666 (max)
Service-Acc-Clt-Trs-W&F-latlng(100 m)	74	125	87	880	921	773	1260	1209	1073	5978	6076	4986	37 (max)
Service-Acc-Clt-Trs-W&F-latlng(200 m)	196	221	226	1351	1039	1152	1243	1223	1225	5186	6195	5418	113 (max)
Service-Acc-Clt-Trs-W&F-latlng(500 m)	948	1091	1602	2159	1709	1698	1159	1187	1232	6130	6738	5933	650 (max)
Service-Acc-Clt-Trs-W&F-latlng(1 km)	3016	3453	3801	–	–	3701	1546	1465	1437	5679	7631	7147	1555 (max)
Service-Acc-Clt-Trs-W&F-latlng(2 km)	4731	5644	7999	–	–	–	1706	1807	1619	6451	8669	7353	2224 (max)
Service-Acc-Clt-Trs-W&F-latlng(5 km)	7983	9610	9974	–	–	–	1785	1938	1813	8024	9669	7646	3102 (max)
Service-Htl-B&B-latlng(100 m)	29	38	29	420	466	392	541	563	631	3843	3985	3886	7 (max)
Service-Htl-B&B-latlng(200 m)	65	84	97	605	471	418	529	606	587	3865	4083	3953	28 (max)
Service-Htl-B&B-latlng(500 m)	293	371	393	1119	724	1032	555	666	544	4055	4605	4424	151 (max)
Service-Htl-B&B-latlng(1 km)	789	1065	1162	810	–	1253	552	630	578	4971	5500	5506	363 (max)
Service-Htl-B&B-latlng(2 km)	1390	1729	2020	–	–	–	617	683	681	5664	6573	6278	488
Service-Htl-B&B-latlng(5 km)	2421	3144	8281	–	–	–	673	744	682	6516	7009	7053	611(max)
Service-text-latlng(500 m)	204,936	236,937	256,328	433	148	324	242	64	73	> 7min	> 7min	> 7min	21 (max)
Sensor-latlng(100 m)	10	12	13	–	–	63	20	27	30	1842	1639	1604	0
Sensor-latlng(200 m)	30	45	36	–	102	54	21	27	32	1800	1734	1635	1
Sensor-latlng(500 m)	41	62	198	118	73	163	18	23	26	1788	2069	1923	4
Sensor-latlng(1 km)	109	159	210	81	–	252	19	24	27	2556	2184	2257	14
Sensor-latlng(2 km)	229	335	444	–	–	–	22	29	29	2372	2798	2670	29
Sensor-latlng(5 km)	514	721	888	–	–	–	23	30	38	2961	2947	2837	56

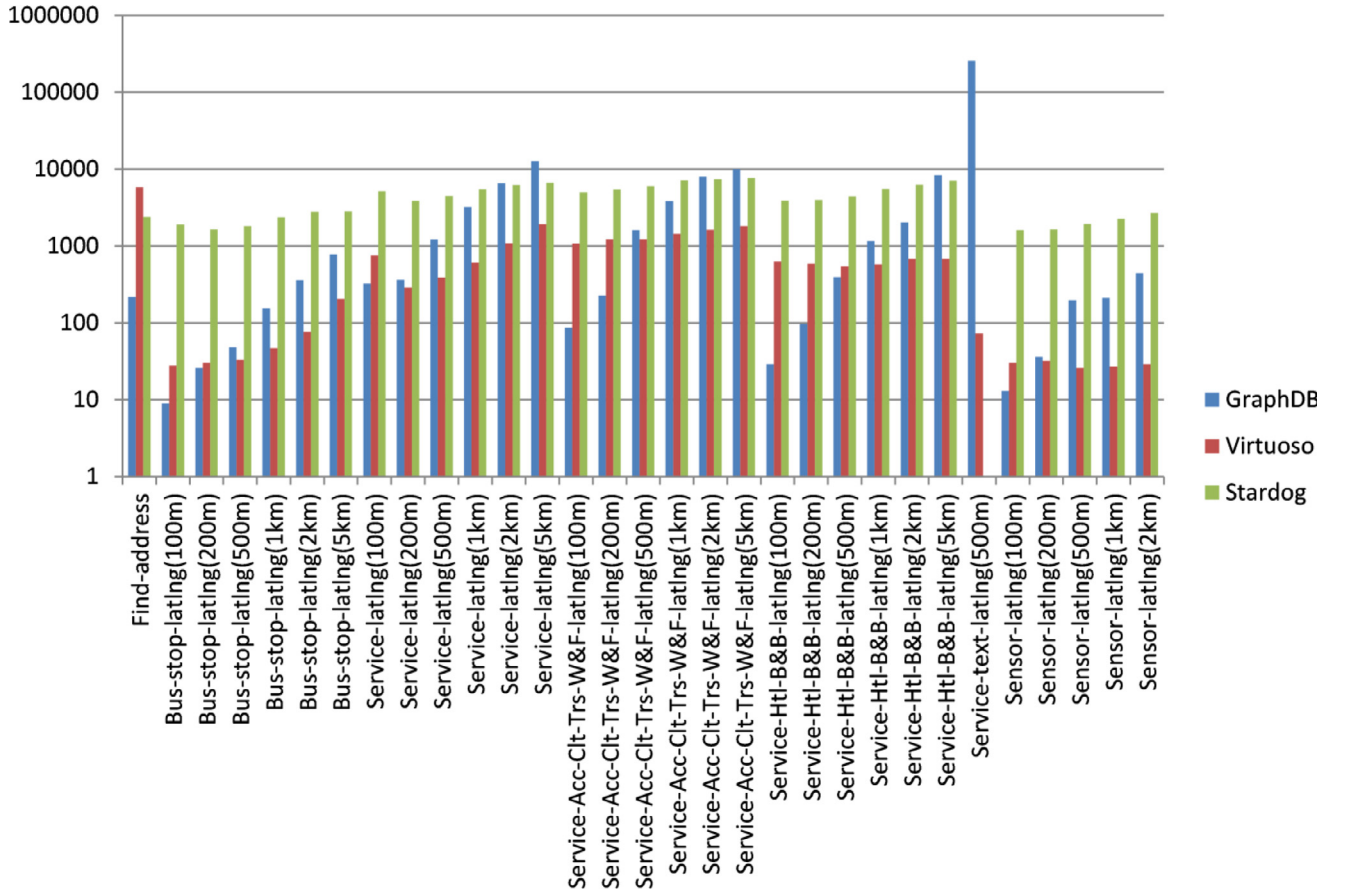


Fig. 1. Spatial queries performance for GraphDB, Virtuoso (using distance function) and Stardog for the 3 months data set.

ranked with less than 0.5 s of execution time. Moreover, Virtuoso presents a less growing factor with respect to GraphDB.

A section of Table 6 results for spatial queries has been reported in Fig. 1. The adoption of the logarithmic scale leads to appreciate the smarter performance of Virtuoso in the cases of reference tests.

We calculated the standard deviation of the query times (on the 3 month dataset and Virtuoso using the distance function) for each query, in general we found that Virtuoso shows the lowest standard deviation among GraphDB and Stardog in most queries (36 out of 52, while GraphDB 6 and Stardog 10). For example for the *full-text* query we calculated a standard deviation of 1,402 ms for GraphDB, 78 ms for Virtuoso and 9,525 ms for Stardog, while the *Bus-stops-florence* query shows a standard deviation of 111 ms for GraphDB, 115 ms for Virtuoso and 74 ms for Stardog.

5.3. Assessing query execution time under update/load

In this section, the behavior of RDF stores under update has been tested according to the assessing conditions and data set explained in Section 3.4. During the test, the time to upload/update new triples for all the sensors and mark them as the ‘latest value’ has been recorded and reported in Table 7. Therefore, the minimum, maximum, average time and the standard deviation of the upload and update time are reported for each RDF store. From the results, it is clear that Virtuoso turned out to be the smartest, since it performed the update of the 430 sensors within 20 s, while GraphDB did the same in 37 s, and StarDog had an average of 42 s and with a maximum time in just one case of 13 minutes to upload new triples for all the 430 sensors. As to Oracle with Apache Jena-Fuseki it was not possible to send the triples for the 430 sensors through Fuseki, since the communication was hanging; while

Table 7
Sensor data upload performance.

	GraphDB	Virtuoso	Stardog
Total mean time (ms)	4135.59	1290.05	42,498.80
Mean upload time (ms)	2105.06	893.52	41,526.02
Mean update time (ms)	2030.53	396.53	972.78
Minimum total time (ms)	1810.00	480.00	6050.00
Maximum total time (ms)	37,294.00	20,678.00	791,083.00
Std. dev of total time (ms)	2197.81	2082.30	76,121.02

when sending the data for only 10 sensors the average time was about 16 s with a maximum of 2.5 min.

In order to evaluate the impact of the update/upload action on query performance, the mean number of queries per hour (MNQPH) has been computed for each RDF store in presence or absence of ongoing upload/updates. MNQPH has been computed as the ratio of total time needed to run a large number of queries of the benchmark and the number of queries. In particular, some of the queries such as: “Service-text-lating(500 m)”, “Service-text-florence” and “Full-text” have been not used because they typically generate on GraphDB and StarDog many timeouts, that could create too noise on assessing query performance during update/load activity.

In the assessment, the time for sensor data access is strongly reduced provided that the “Sensor-status” query requests the latest value triple marker. The results are reported in Table 8, where you can see that the MNQPH is decreased in all the cases, shifting from the value registered with RDF store under no updates up to the value registered during the store updates. The decrease in performance is due to the fact that the query has to wait for the

Table 8

Store performance in presence and absence of updates during benchmark.

RDF store	MNQPH		
	No updates	During updates	Loss in performance
GraphDB	2117.00	1799.93	14.97%
Virtuoso	4584.16	4362.21	4.84%
Stardog	1620.24	876.63	45.89%

Table 9

Performance in accessing to the last value of sensors.

RDF store	Mean time to sensor status access, Time in ms		
	Case 1 (no update)	Case 2 (no update)	Case 3 (update)
GraphDB	1561	31	334
Virtuoso	163	46	174
Stardog	393	208	554

store unlock. Among the RDF stores considered, Virtuoso presented the lower reduction in performance. Moreover, as stated above, the benchmark occurred in some time outs with to GraphDB and StarDog stores in the absence of updates; typically 46 and 96 times for the whole benchmark. The number of timeouts is more than twice in presence of updates.

Table 9 reports the mean time to get access to the latest value of a sensor series (the *Sensor-status* query) in three cases: (1) using the order by clause and without concurrent updates, (2) using the “latest value” triple without concurrent updates and (3) using the latest value triple during concurrent updates. For all the stores we can see that when avoiding the sort and using the “latest value”, the time needed to access is reduced. However, performing a concurrent update increases access time of a significant amount (i.e., more than 10 times for GraphDB, 3.8 times for Virtuoso and 2.7 for StarDog). According to the access mean time values, Virtuoso could perform better than the others in all the cases.

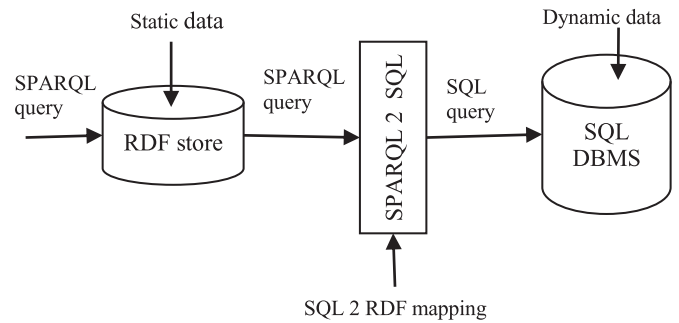
6. Evaluation of RDF stores with SQL mappings

In the smart city context, it often occurs to have a huge number of static information and a large number of triples due to the real time data, growing in time, as already seen in the test data sets. Therefore, you are supposed to take advantage of mixing RDF store with classical tables and SQL for real time data. Therefore, this section is meant to assess the integration of RDF stores used for storing the static datasets, with SQL DBMS used for storing the dynamic/real-time data. This aspect is added to RDF stores by using an SQL-RDF mapping which allows any querying of SQL DBMS using SPARQL.

As to the problem of mapping relational databases to RDF, this is not new and there are several solutions both commercial and open source.¹⁰ Oracle, Stardog and Virtuoso provide support for mapping relational data as RDF. Moreover, in 2012 W3C published R2RML¹¹ a standardized language to describe relational to RDF mappings.

Two queries of the benchmark dealing with real-time data were used for the evaluation of this feature, presented before. In Fig. 2, the reference simple architecture representing the mixture of RDF store and SQL database and thus of the configuration considered in this assessment test is reported.

The RDF Store and solutions we tested where (i) Virtuoso that has an integrated SPARQL-SQL mapper for its internal relational

**Fig. 2.** Query architecture when RDF store is mixed with SQL store.

DBMS, and (ii) D2R¹² with MySQL with Apache Jena-Fuseki as front-end RDF store using the SERVICE clause of SPARQL to use the D2R SPARQL endpoint. In fact, the SERVICE clause of SPARQL 1.1 allows the exploitation from which a query on a SPARQL server of the results coming from a SPARQL query executed on another SPARQL server, thus allowing any joining of the results of two separate worlds. In this context, one SPARQL endpoint is used for storing static data, while the other is for dynamic data.

The testing queries adopted for the evaluation have been described in Table 1: (i) *Sensor-status* to retrieve the latest value obtained from a specific traffic sensor, and (ii) *Bus-stop-forecast* to obtain the latest forecasts on bus arrival for a specific bus stop. The first query is very simple: it orders the results by observation time and gets the most recent values; whereas when it comes to bus forecasts, it involves a more complex query using a sub query. As specified in the benchmark, the above mentioned two queries were adapted to be used in the new query architecture; all the queries are reported in <http://www.disit.org/smartcityrdfbenchmark>.

The queries were tested with an increasing number of records on the SQL DBMS (28k, 50k, 100k, 200k, 300k, 500k and 1M). In Figs. 3 and 4, the query execution time for four different cases has been provided. Such cases are:

- Virtuoso static data as RDF and using the SQL mapping for dynamic data.
- Fuseki RDF store for the static data and D2R with MySQL for the dynamic data.
- Virtuoso RDF with both static and dynamic data.
- Fuseki RDF with both static and dynamic data.

Fig. 3 shows that as to the *sensor status* query on the single instance, Virtuoso RDF store *outperforms* the other solutions providing the results in the shortest time, while Fuseki RDF is not well ranked with respect to the other solutions. Please note that the Sensors status query of Table 1 is quite simple and presents only parameters. And construct for order-by. Moreover, among the SQL mapping solutions, Virtuoso performs better than Fuseki&D2R.

As to the *bus-stop forecast query*, Fig. 4 shows that Virtuoso RDF (virtRDF) is better ranked with respect to the other solutions, and both the SQL mapping based solutions have the lowest performance (i.e., highest execution time). As explained in Table 1, the query adopted for the assessment presents parameters, sub-query the constructs as group-by and order-by. It can be seen from Fig. 4 that full RDF based solutions (virtuoso and fuseki) have an almost linear trend while for SQL based mappings (virtuoso and D2R) the trend is almost exponential (see the log scale of Fig. 4). Please note that, for some cases the queries cannot be completed in a reasonable time when dealing with a number of records greater than 100,000 for fuseki + D2R and greater than 300,000 for virtRDF + SQL.

¹⁰ <https://www.w3.org/2001/sw/rdb2rdf/wiki/Implementations>.

¹¹ <http://www.w3.org/TR/r2rml/>.

¹² <http://d2rq.org/>.

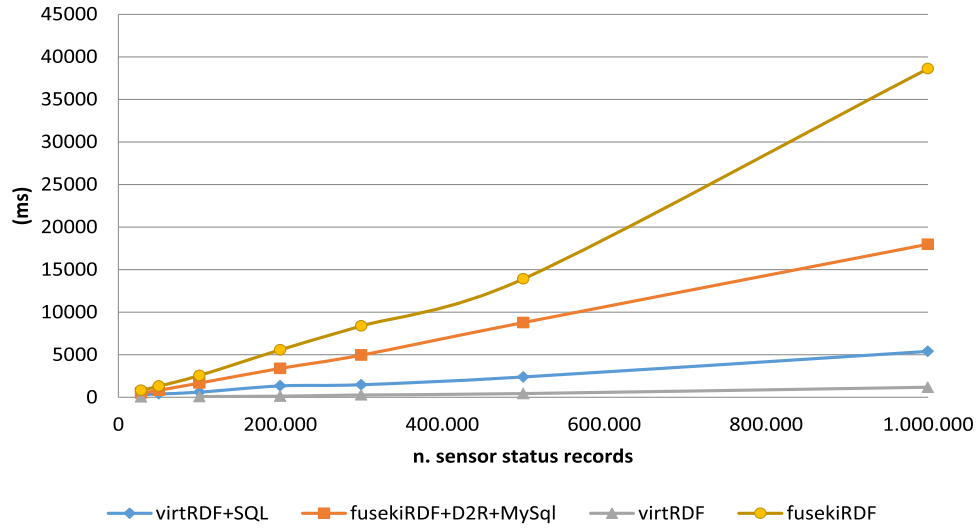


Fig. 3. Performance comparison among native RDF Store vs RDF Stores using integration with SQL: query on *sensors status* values, see query description on Table 1.

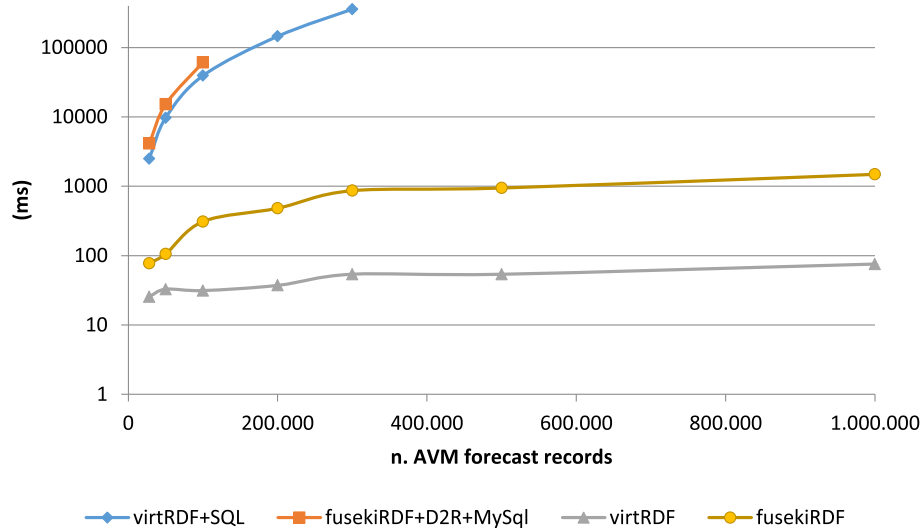


Fig. 4. Performance comparison among native RDF Store vs RDF Stores using integration with SQL: query on *bus stop forecast/delay* query time, see query description on Table 1.

In both examined cases, the Virtuoso RDF store performed better than the other solutions. In the event of a simple mapping (as the one of sensors), the performance of Virtuoso SQL mapping is still acceptable (about 5 seconds for 1 M records), while when the query becomes more complex, both SQL mapping solutions perform unacceptably and for greater datasets they were unable to provide a result in a reasonable time.

7. Conclusions

The usage of RDF stores to store smart city data is becoming of wide interest for several applications. In this paper we have proposed a *Smart City RDF Assessment Model* for a comparative study about the state of the art on RDF stores according to their main features and in particular on the SPARQL aspects/features. In addition, the *Smart City RDF Benchmark* has been proposed. The benchmark is based on (i) some datasets of triples (that are grounded on Km4City ontological model) accessible from <http://www.disit.org/smartcityrdfbenchmark>, it can be used only for benchmarking purpose; (ii) a set of SPARQL queries declined for different SPARQL constructs. The benchmark has been defined for smart city services to compare results which can be obtained by using different RDF

Stores. In the benchmark, particular emphasis has been given to geo-spatial and full text searches, since such aspects have been only partially considered and addressed by the general state of the art benchmarks such as LUBM and BSBM. As a general remark, the produced benchmark can be profitably used in several other contexts where similar aspects are modeled.

The comparison addressed a number of well-known RDF stores such as Virtuoso, GraphDB, StarDog, and Oracle for the performance aspects. As a general consideration about performance, it should be noted that Virtuoso performs better in presence of less selective queries, thus providing a higher number of results. On the contrary, GraphDB performs better when specific results are searched, thus when a smaller number of results are requested. As to Virtuoso, some small problems have been detected. For example, with the *st_intersect* function that, in order to get results, constrained us to rewrite queries using *st_distance* function. It seems that in this case the spatial indexing structure is not used and the optimizer does not exploit it as the starting point for a selection.

Moreover, the performance of native RDF stores with standard SQL DBMS integration was investigated, as well. From the performed tests, the solutions adopting SPARQL to SQL rewriting need

to be improved, since when it comes to complex queries, the performance is still very poor with respect to full RDF solutions.

Acknowledgement

The authors would like to say thanks to ONTOTEXT for granting us the access to a trial version of their RDF store. This works has been developed in the framework of Km4City activity for RES-OLUTE H2020, and for REPLICATE H2020 (both are European Commission funded International Projects) and for Sii-Mobility Smart City National project (<http://www.sii-mobility.org>).

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:[10.1016/j.jvlc.2018.03.002](https://doi.org/10.1016/j.jvlc.2018.03.002).

References

- [1] MuhammadIntizar Ali, Feng Gao, Alessandra Mileo, CityBench: a configurable benchmark to evaluate RSP engines using smart city datasets, 14th Int. Semantic Web Conference, 2015.
- [2] L. Anthopoulos, P. Fitsilis, Exploring architectural and organizational features in smart cities, in: Advanced Communication Technology (ICACT), 2014 16th International Conference on, 2014, pp. 190–195, doi:[10.1109/ICACT.2014.6778947](https://doi.org/10.1109/ICACT.2014.6778947), 16–19 Feb.
- [3] C. Balakrishna, Enabling technologies for smart city services and applications, in: Next Generation Mobile Applications, Services and Technologies (NGMAST), 2012 6th International Conference on, 223, 2012, p. 227. 12–14 Sept.
- [4] C. Badii, P. Bellini, D. Cenni, A. Difino, P. Nesi, M. Paolucci, Analysis and assessment of a knowledge based smart city architecture providing service APIs, Future Generation Computer Systems, Elsevier, 2017 <http://dx.doi.org/10.1016/j.future.2017.05.001>.
- [5] D.F. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, C-sparql: sparql for continuous querying, in: Proc. of WWW, ACM, 2009, pp. 1061–1062.
- [6] P. Bellini, M. Benigni, R. Billero, P. Nesi, N. Rauch, Km4City ontology bulding vs data harvesting and cleaning for smart-city services, Int. J. Vis. Lang. Comput. (2014).
- [7] P. Bellini, I. Bruno, P. Nesi, N. Rauch, Graph databases methodology and tool supporting index/store versioning, J. Vis. Lang. Comput. (2015). <http://www.sciencedirect.com/science/article/pii/S1045926X15000750>, doi:[10.1016/j.jvlc.2015.10.018](https://doi.org/10.1016/j.jvlc.2015.10.018).
- [8] K. Bereta, P. Smeros, M. Koubarakis, Representing and querying the valid time of triples for linked geospatial data, The 10th Extended Semantic Web Conference (ESWC 2013), 2013 May 26–30.
- [9] C. Bizer, A. Schultz, The Berlin SPARQL benchmark, Int. J. Semant. Web Inf. Syst. 5 (2) (2009) 1–24.
- [10] J.-P. Calbimonte, O. Corcho, A.J.G. Gray, Enabling ontology-based access to streaming data sources, in: ISWC, 2010, pp. 96–111.
- [11] H. Chourabi, Taewoo Nam, S. Walker, J.R. Gil-Garcia, S. Mellouli, Karine Nahon, T.A. Pardo, Hansjochen Scholl, Understanding smart cities: an integrative framework, in: System Science (HICSS), 2012 45th Hawaii International Conference on, 2012, p. 2289,2297. 4–7 Jan.
- [12] S. Duan, A. Kementsietsidis, K. Srinivas, O. Udrea, Apples and oranges: a comparison of RDF benchmarks and real RDF datasets, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD '11). ACM, New York, NY, USA, 2011, pp. 145–156.
- [13] O. Erling, I. Mikhailov, Virtuoso: RDF support in a native RDBMS, in: Semantic Web Information Management, Springer, 2009, pp. 501–519.
- [14] G. Garbis, K. Kyzirakos, M. Koubarakis, Geographica: a benchmark for geospatial RDF stores, the 12th International Semantic Web Conference (ISWC 2013), 2013 October 21–25.
- [15] GeoGeoSPARQL. Open Geospatial Consortium, “GeoGeoSPARQL - a geographic query language for RDF Data”. Sept. 10 <http://www.opengeospatial.org/standards/geosparql>. 2012.
- [16] Y. Guo, Z. Pan, J. Heflin, Lubm: a benchmark for owl knowledge base systems, J. Web Semant. 3 (2–3) (2005) 158–182.
- [17] Kehua Su, Jie Li, Hongbo Fu, Smart city and the applications, in: Electronics, Communications and Control (ICECC), 2011 International Conference on, 2011, p. 1028,1031. 9–11 Sept..
- [18] Zaheer Khan, Ashiq Anjum, SaadLiaquat Kiani, Cloud based big data analytics for smart future cities, in: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing (UCC '13). IEEE Computer Society, Washington, DC, USA, 2013, pp. 381–386. <http://dx.doi.org/10.1109/UCC.2013.77>.
- [19] K. Kyzirakos, M. Karpathiotakis, M. Koubarakis, Strabon: a semantic geospatial DBMS, the 11th International Semantic Web Conference (ISWC 2012), 2012 11–15 November.
- [20] G. Ladwig, A. Harth, CumulusRDF: linked data management on nested key-value stores, 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011), 2011.
- [21] D. Le-Phuoc, M. Dao-Tran, J.X. Parreira, M. Hauswirth, A native and adaptive approach for unified processing of linked streams and linked data, in: ISWC, 2011, pp. 370–388.
- [22] D. Le-Phuoc, M. Dao-Tran, M. Pham, Linked stream data processing engines: facts and figures, in: International Semantic Web Conference (ISWC 2012), 1380, Springer, Boston, USA, 2012, pp. 300–312.
- [23] C.E.A. Mulligan, M. Olsson, Architectural implications of smart city business models: an evolutionary perspective, Commun. Mag. IEEE vol.51 (6) (2013) 80,85 June.
- [24] N. Papailiou, I. Konstantinou, D. Tsoumakos, P. Karras, N. Koziris, H2RDF+: high-performance distributed joins over large-scale RDF graphs, in: Big Data, 2013 IEEE International Conference on, 2013, p. 255,263. 6–9 Oct..
- [25] Michael Schmidt, et al., SP²Bench: a SPARQL performance benchmark, Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on. IEEE, 2009.
- [26] W3C Consortium, SPARQL 1.1 Query Language, W3C Recommendation, 21 March, 2013 <http://www.w3.org/TR/sparql11-query/>.
- [27] WellingtonM. da Silva, Alexandre Alvaro, GustavoH.R.P. Tomas, RicardoA. Afonso, KelvinL. Dias, ViniciusC. Garcia, Smart cities software architectures: a survey, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13). ACM, New York, NY, USA, 2013, pp. 1722–1727. <http://dx.doi.org/10.1145/2480362.2480688>.
- [28] Zaheer Khan, Ashiq Anjum, SaadLiaquat Kiani, Cloud based big data analytics for smart future cities, in: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing (UCC '13). IEEE Computer Society, Washington, DC, USA, 2013, pp. 381–386. <http://dx.doi.org/10.1109/UCC.2013.77>.
- [29] Y. Zhang, M.-D. Pham, O. Corcho, J.-P. Calbimonte, SRBench: a streaming RDF/SPARQL benchmark, in: Proc. of the 11th International Semantic Web Conference ISWC 2012, Boston, USA, 2012 Nov.