Intelligent Automation And Soft Computing, 2020

Vol. 26, no. 1, 107–119
https://doi.org/10.31209/2019.100000132

A Novel Knowledge-Based Battery Drain Reducer for Smart Meters

Isma Farah Siddiqui[1], Scott Uk-Jin Lee[2], Asad Abbas[3]

[1]Department of Software Engineering. Mehran University of Engineering and Technology, Pakistan.
[2]Department of Computer Science and Engineering, Hanyang University ERICA, 55 Hanyangdaehak-ro, Sangnok-gu, Ansan, Gyeonggi-do, 15588, Republic of Korea.
[3]Department of Software Engineering, University of Lahore, Lahore, Pakistan.

ABSTRACT
The issue of battery drainage in the gigantic smart meters network such as semantic-aware IoT-enabled smart meter has become a serious concern in the smart grid framework. The grid core migrates existing tabular datasets i.e., Relational data to semantic-aware tuples in its Resource Description Framework (RDF) format, for effective integration among multiple components to work aligned with IoT. For this purpose, WWW Consortium (W3C) recommends two specifications as mapping languages. However, both specifications use entire RDB schema to generate data transformation mapping patterns and results large quantity of unnecessary transformation. As a result, smart meters use huge computing resources, maximum energy capacity and come across battery drain problems. This paper proposes a novel semantic-aware battery drain optimization strategy 'SPARQL Auto R2RML Mapping (SARM)' that generates custom RDF patterns with precise metadata and avoids use of full schema along with optimized usage of network resources through (i) selective metadata migration, and (ii) optimal battery usage. The proposed approach effectively increases battery life with a balanced proportion of energy consumption and reduces meter load congestion which happens to be another vital reason of battery drain problem. The presented knowledge-based battery drain prevention strategy is evaluated over an RDB dataset using three types of SPARQL queries; Basic, Nested and Join. Furthermore, the R2RML processors evaluated SARM over the most recent Berlin SPARQL Benchmark datasets which depicts that SARM is efficient 40.4% in mapping generation time and 10.46% in average planning time than default RDB2RDF transformations. Finally, SARM significantly improves total execution time of RDB2RDF migration with an efficiency of 8.82% and conserves battery drain by 18.5% over the smart grid data cluster.

KEY WORDS: Smart grid, Smart meter, Battery Energy, Semantic Web (SW); Relational Database to RDF.

1 INTRODUCTION

THE idea of smart grid is about enabling the automation to power generation and electricity management. The grid architecture consists of power system, intelligent agents, on-grid Battery Energy Storage (BES), communication technology and IT infrastructure, IoT-enabled smart meters, distribution ends, electric transportation and finally, data management systems (Tuballa et al., 2016; Abbas et al., 2017). The power system produces electric energy for consumer ends i.e., factories, product plants and for home consumers through plug-and-play interconnects (Amin et al, 2005). The intelligent agents are capable of monitoring, visualizing and coordinating energy needs with in the smart grid (Siddiqui et al., 2017; Baig & Zeadally, 2019). Secondly, they ensure cost effective supply, reduction of carbon emissions and micro-storage of electricity in smart grid (Pipattanasomporn et al., 2009; Siddiqui et al., 2018). On-grid BES provides a back-up to irregular renewable energy and offers a balanced energy supply on demand. BES improves efficiency and management of distribution networks with a facility of reduction in energy cost (Bussar et al., 2013; Azmat et al., 2017). The energy storage

CONTACT Scott Uk-Jin Lee scottlee@hanyang.ac.kr

Table 1. Smart meter battery usage statistics (Longe et al., 2017)

Application	Idle Smart Meter	Active Smart Meter	Active Smart Meter (RDF)
Voltage	2.5 to 2.7 Volts	2.5 to 2.7 Volts	2.5 to 2.7 Volts
Operation	80 μA continues current 10 mA for 11 ms every 2 seconds	120 μA continues current 15 mA for 11 ms every 2 seconds	120 μA continues current 15 mA for 11 ms every 2 seconds
Average current	160 μA	240 μA	240 μA
Temperature	$\pm -20°C ... +60°C$	$\pm -20°C ... +60°C$	$\pm -20°C ... +60°C$
Service Life	0.5 to 1 year	0.3 to 1 year	0.3 to 0.6 year

application over batteries consists of four levels i.e., (i) generation, (ii) transmission, (iii) distribution, and (iv) customer level.

The generation level supply depends on arbitrage, curtailment reductions and capacity firming. The transmission level feed covers frequency and voltage control, curtailment reductions, black staring and investment deferrals. The distribution level support relies on voltage control, curtailment reduction and capacity support. The consumer level supply depends on peak shaving, off-grid and cost management time-line supply (Roberts, 2011). The next important component of grid architecture is the communication technologies such as software and hardware, to provide a medium to control and transmit power generation and battery storage perspectives in the smart grid (Yoldaş et al., 2017). The software perspective includes generation of sensor data profiles, transformation of datasets and storing big size of data into scalable repositories (Kenner et al., 2017). The sensor data generation is carried out through IoT-enabled smart meters in industry standard formats such as 'CSV' and 'JSON' (Siddiqui et al., 2016; Abbas et al., 2018). The smart grid provides a functional layer of semantic web to communicate and store grid analytics. Therefore, the transformation of generated datasets is performed through a semantic-aware strategy i.e., Resource Description Framework (RDF) (Su et al., 2014). The transformed 'Tuple' dataset is stored over smart grid tuple-store (Pena et al., 2011). The grid core synchronizes dataset tuples through big data analytics and stores over giant smart grid repository (Daki et al., 2017; Qureshi et al., 2017; Qureshi et al., 2017; Zhou & Luo, 2017). The hardware perspective discusses wired and wireless scenarios such as copper-wire line, fiber optics, Power Line Communication (PLC) (Sood et al., 2009), ZigBee (Yi et al., 2011), GSM/GPRS/3G/WiMAX (Gungor et al., 2013), and Cognitive Radio (Ghassemi et al., 2010). The grid infrastructure requires peer-to-peer communication between all nodes and therefore, network adopts mesh topology (Gungor et al., 2013). The IoT-enabled smart meter provides two-way communication, data collection, data transformation and programming, security, display, and billing facility to the distribution ends of smart grid (Zheng et al., 2013; Azmat et al., 2018). The functional features of smart meter include usage time rate at battery energy and interval data for monitoring analytics (https://www.smart-energy.com/). The usage rate stores sensor data in idle and active states with the log of battery energy consumption. The monitoring analytics perspective involves fetch and transform dataset to store at the distribution end (Albu et al., 2017). The smart meter provides Real Time Clock (RTC) that records events during the power-cut and functionally rely on the health of battery (Wang & Xie, 2017). The meter battery provides a storage of 1 year at 30 _A with a back-up time of 10 minutes per year at 200 _A between temperature of ±0°C…+50°C at 2.7 V (Longe et al., 2017). The routine battery consumption of an individual smart meter for generating sensor data can be observed from Table-1.

The reason of gradually decreasing lifespan and battery drain depends over the additional workload of RDF transformation (Backes & Meiser, 2013). The semantic-aware data processing channel transports light-weight data chunks and is $\frac{1}{7}$ times less carrier overhead than schema data processing (Wagner et al., 2010). The network nodes involved in passing of RDF tuples consumes 23.7% less computing and network resources than traditional RDB tabular dataset transportation (Zhou et al., 2012). In order to maintain such network performance of smart grid, a candid scheme is needed that minimizes lifespan and battery drain problems with a continuity of default smart meters' data-centric operations. Therefore, we propose SPARQL auto R2RML Mapper (SARM) that generates custom RDF tuples metadata without using complete tabular schema. The proposed approach significantly reduces battery drain problem by adopting fewer computing resources and gradually stabilizes lifespan of smart meter battery.

The noteworthy contributions of proposed approach are in particular as:

- A novel knowledge-based battery drain reducer using data engineering.
- An advanced approach of stabilizing battery lifespan in smart meter using data analytics.
- A novel SQL-less mapping parser for reducing privacy infringements on smart meter sensor data in grid core.
- A state-of-the-art RDF dataset SPARQL query parsing through Basic, Nested and Join SPARQL queries.
- A novel Java coded tool SARM as R2RML generator for custom RDF tuples for on-the-fly data transformations.

The remaining paper is well-ordered in the following sections. The section 2 deals with related

knowledge of semantic data format and languages used for processing over smart meter battery with state-of-art technologies. The section 3 provides detail explanation of SARM methodology. The section 4 evaluates the proposed approach with examples. The section 5 depicts performance evaluation over transformation of benchmark datasets into RDF tuple. Finally, section 6 concludes the paper along with potential future works.

2 BACKGROUND

THE smart meter RDF tuple processing includes the roles of smart grid, smart meter, sensor data generation, RDF transformation and storage of R2RML data tuples over smart grid repository. As the proposed approach revolves around semantic-aware data-centric strategy and RDF transformation of sensor data, we focus to discuss semantic web technology and integral components only.

Table 2: Common Used Term in this Paper

Term	Description
SW	Semantic Web
W3C	World Wide Web Consortium
RDB	Relational Database
RDBMS	Relational Database Management System
RDF	Resource Description Framework
RDF Tuple/Triple	RDF Data (consist of Subject, Predicate, Object elements)
DM	Direct Mapping
R2RML	RDB to RDF Mapping Language
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
RDF Tuple Store	Datastore for RDF Tuples
IRI	Internationalized Resource Identifier
ETL	Extract-Transform-Load
BSBM	Berlin SPARQL Benchmark
SPARQL Endpoint	SPARQL Processor (Web Interface)

2.1 Resource Description Framework

THE standardized data format for semantic web is Resource Description Framework (RDF); given by Brickley (2010). This model represents an object as a resource in triple format. An RDF triple comprises of three main sections, a subject resource's IRI, a predicate resource's IRI and an object resource's IRI or simply literal. In order to simplify RDF, we consider set notations such as the uppercase letter denotes sets; lower case letters represent elements and relations and calligraphic capital letter depicts sets of identical elements. The significant set abbreviations can be represented as:

1) Ir as set of IRI.
2) L as set of literals.
3) V as set of variables.
4) T as set of terms, set of $(Ir \times L)$.

The RDF tuple is a subset of $(T \times T \times T)$ and the graph as subset of $(Ir \times Ir \times T)$. A tuple T consists of three elements subject, predicate and object as (s, p, o). It is graphically represented as a directed graph having node identity resource, i.e., subject and object and an arc to represent a predicate between two nodes. The resources inter-connects through single object of n resource to single subject of m resource. The most suitable example of object interconnected graph database is DBPedia (Bizer et al., 2009). Unlike the structure of RDF tuple, it is not possible to read and understand a large size of graph dataset. Therefore, translation languages such as Terse RDF Triple Language, also known as, Turtle (ttl) (Prud'hommeaux et al., 2013), RDF/XML (Beckett et al., 2004), Notation3 (N3) (Berners-Lee & Connolly, 2011), RDFa (Adida et al., 2012), and JSON (Sporny et al., 2014) are used to translate large graph datasets into human readable formats. We use Turtle markup language to transform custom RDF tuples in the smart meters. An example RDF tuple in Turtle syntax is written as:

<http://www.abc.com/student/242>
<http://www.abc.com/student_name>
"Lucy".

The above example shows a tuple with subject IRI as <http://www.abc.com/student/242> having predicate IRI as <http://www.abc.com/student_name> with object element of string literal "Lucy". This example RDF tuple elaborates that an RDF resource is available at an IRI <http://www.abc.com> as a smart grid web resource.

2.2 SPARQL Protocol and RDF Query Language

THE RDF view and tuples are retrieved from tuple store and RDB database through SPARQL (Harris et al., 2010). The query does not use case sensitive language and consists of five clauses:

1) PREFIX [none or more] - for declaration of short hand URI prefix,
2) SELECT [one] - for declaring variables to hold query results,
3) FROM or FROM NAMED [none or more] - for identifying RDF tuple dataset,
4) WHERE [one] - for queried tuple statements of RDF graph pattern,
5) Solution sequence modifiers in SPARQL (LIMIT, PROJECTION, REDUCED, OFFSET, ORDER BY, and DISTINCT.) [none or one each] - used in mentioned order for giving sequence to query results.

The literal syntax of SPARQL query to extract RDF tuple data matches with literal syntax of SQL query for tabular RDB data. The SPARQL endpoint is a query processor and grid web server that manages RDF tuple dataset over HTTP transportation protocol. An example SPARQL query is highlighted as:

```
PREFIX type: <www.abc.com/type/>
SELECT ?var1 ?var2
WHERE
{?var1 type:is_of_type ?var2.}
```

The above example query contains a SELECT clause that holds two variables for storing results (i.e., var1 and var2) with a parameter '?' sign of variables. The WHERE clause tracks variable condition and

returns identical tuple entries of subject and object elements. The SPARQL query is flexible to relate n number of semantics with resource and predicate and motivates the proposed approach to adopt this feature for extracting custom RDF tuples.

2.3 Relational Database to Resource Description Framework (RDB2RDF)

The term RDB2RDF indicates a transformation process of tabular dataset into tuples. A mapping file is used for this process having identical pattern and variables. The patterns are matched with the SQL clauses of legacy database RDB and generate an RDF tuple dataset (Calbimonte et al., 2010). Initially W3C RDB2RDF Incubator Group (Sahoo et al, 2009; Malhotra, 2009) reviewed the work of researchers and endorsed various comparison techniques of RDB2RDF data mapping and translations. Later on, the incubator group was acquired by W3C group and they introduced two mapping standards; i. Direct Mapping, and ii. RDB2RDF Mapping Language (Arenas et al., 2011; Das et al., 2012). The Direct Mapping adopts Extract-Transform-Load (ETL) function to generate duplicate RDF dataset of existing RDB schema and store over RDF triple stores. The direct mapping parameters process full schema and consume huge length of computing resources to generate direct graph. Therefore, it is only suitable for large scale enterprise dataset, where energy consumption is not a primary concern. The R2RML mapping uses Turtle language with ETL to generate full and partial RDF triple datasets (Hazber et al., 2016). The Turtle language plays a vital role and provides human readable syntax that supports developer to produce partial RDF triple. This reduces a huge workload of transforming unwanted large RDF triples and consumes less computing resources with minimized battery energy.

2.4 RDB2RDF Mapping Language (R2RML)

This R2RML standard language processes RDB schema through Turtle language and compatible mapping processor (Das et al., 2012; Chhaya et al., 2016). The mapping patterns such as tripleMap, subjectMap, predicateObjectMap, predicateMap and objectMap builds an association with RDB schema elements such as table and column. After synchronizing map entities, a tree structure is formed which consists of root, sub-levels and leaves. The root states triplesMap having RDB table entities and extends to sublevels with logicalTable, subjectMap and predicateObjectMap. The tree concludes at leaves having predicateMap and objectMap entities of predicateObjectMap sublevel. The relational schema elements associates n subjectMap to m columns of RDB table, $n + i$ objectMap to $m + x$ columns and p $\{n + i; m + x\}$ predicate to subjectMap and objectMap entities. An example R2RML mapping pattern of RDB 'Student' is written as:

```
@prefix r2r: <http://www.w3.org/ns/r2rml\#>.
@prefix st: <http://www.xyz.com/ns\#>.
<#triplesMapstudent> a r2r:TriplesMap;
r2r:logicalTable [rr:tableName 'student'];
r2r:subjectMap [rr:template
"http://www.xyz.com/student/{S_Id}"];
r2r:predicateObjectMap
[r2r:predicateMap
[r2r:constant student:s_name];
r2r:objectMap
[r2r:column "F_Name"]; ].
```

3 SPARQL AUTO R2RML MAPPING (SARM)

SARM is a custom mapping approach that uses SPARQL query to generate smart grid custom RDF triples. It consists of four phases i.e., (i) Relational Database (RDB) Scenario, (ii) SPARQL-Query Parser, (iii) Generation of R2RML mapping patterns, and (iv) RDB2RDF data transformation. The smart meter collects sensor data and produces 'CSV' dataset in the form of relational database (RDB). The SPARQL parser receives n number of queries and produces a (n) 'ttl' mapping files. The Turtle language perform mapping operation over 'ttl' files and generate m(ttl) mapping patterns. The RDB2RDF processor produces custom RDF dataset over smart meter as seen from Fig.1.

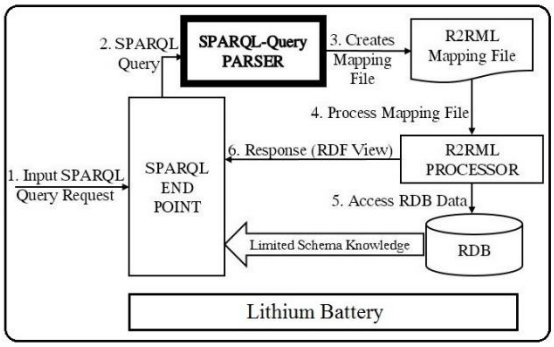

Figure 1. Custom transformation of Smart meter energy data using RDB2RDF

3.1 Relational Database Scenario

The smart meter collects sensor data and build a relational database having n Tables and c (n) columns dataset. Assume a relational database 'Department' built with the help of MySQL RDBMS (Bell., 2018). The RDB consists of six tables i.e., student, teacher, course, laboratory, exam, and grade as seen from Fig.2. The tables are connected through aggregation and cardinality properties such as teacher -> (1; n) <- student, teacher -> (1; 1) <- course, student -> (1; n) <- course, laboratory-> (1; 1) <- student, student-> (1; n) <- grade and exam -> (1; n) <- course cardinalities. The 'Department' RDB contains in-built locks, stored-

procedures, views, key associations and security parameters that gives weight-age to dataset and build huge amount of schema metadata. In case of *n* number of smart meter *z* generate *m* RDB schema, smart meter bears huge amount of RDF triple metadata and consumes large amount of battery energy to transform and dispatch RDF triple over smart grid repository. Therefore, SARM removes in-built RDB functionalities and co-relate tabular data transformation as,

$$RDB_{dataset} = \sum_{\text{Department}_{\text{Tabulardataset}}}^{\text{Department}_{in-built(m)}} (\text{Tables}_{\text{n-m}})$$

3.2 SPARQL-Query Parser

The SPARQL-Query Parser generates p (Table *n*) mapping patterns of RDBDataset through grid query m and store in 'ttl' file m(ttl) as seen from Fig.3. The parsing method is working in two steps:

1) SPARQL query parser: The identification of query p (Table *n*) elements.

2) Generation of R2RML mapping patterns: The association of p (Table n) elements with logicalTable, subjectMap, predicateObjectMap, predicateMap and objectMap parameters. 1) Algorithms for Parsing SPARQL Query and Generating R2RML Mapping Patterns: The default SPARQL query syntax comprises of an optional PREFIX statement and mandatory SELECT and WHERE clauses. The ?variable elements are user-defined and changes with the specification requirement of smart grid. SARM adopts three types of SPARQL query i.e., (i) Basic Select (ii) Nested Select and, (iii) Join Select; to extract smart meter data.

The KEYWORD and variable elements in syntax of SPARQL query are written as:

Basic Select

```
PREFIX Name_of_prefix:<http://IRI-Title
/ns#>
SELECT ?Variable-1 ?Variable-2
WHERE
{ ?Variable-1 Name_of_prefix:Predicate
?Variable-2. }
```

Nested Select

```
PREFIX Name_of_prefix:<http://IRI-Title
/ns#>
SELECT ?Variable-1 ?Variable-2
WHERE { SELECT ?Variable-3 ?Variable-4
    WHERE
    { ?Variable-3 Name_of_prefix:Predicate
  ?Variable-4. }}
```

Join Select

```
    PREFIX Name_of_prefix:<http://IRI-Title
    /ns#>
    SELECT ?Variable-1 ?Variable-2
    WHERE
    {?x Table-1:Variable-1 ?variable-1.
    ?x Table-1:Variable-2 ?variable-2.
    ?x Table-1:Variable-3 ?variable-3.
    ?x Table-1:Variable-4 ?variable-4.
    ?v Table-2:Variable-3 ?variable-3.
    ?v Table-2:Variable-5 ?variable-5.
    ?m Table-3:Variable-4 ?variable-4.
    ?m Table-3:Variable-5 ?variable-5.}
```

The basic syntax of an R2RML mapping template of SARM is written as:

```
@PREFIX rr:<http://www.w3.org/ns/r2rml#>.
@PREFIX namespace:<http://namespaceIRI/
ns#>.
<#triplesMapName >
a rr:TriplesMap;
rr:logicalTable [[rr:tableName
"Tablename"];
rr:subjectMap [rr:template
"http://IRI-Title/namespace/{Subject-
Column}"];
rr:predicateObjectMap
[rr:predicateMap[rr:constant
namepspace:Predicate];
rr:objectMap [rr:column "Object1- Column"]; ];
```

The Algorithm-1 extracts Basic Select mapping file from query Q and includes prefix IRIs and tablename information with WHERE clause condition to retrieve query result. The algorithm starts with identifying the Name_of_prefix followed by an IRI-Title in the input SPARQL query. This Name_of_prefix serves at name-space, triplesMapName_of_prefix and Tablename in generated R2RML mapping file. The IRI-Title replaces namespace IRI and reaches at next line of SPARQL query stating SELECT clause and identifies variable names. The first variable is used as Subject-Column inside SubjectMap and the remaining variables are used as Object-Column for ObjectMap in generated R2RML mapping. After that, the WHERE clause of query is processed and the Name_of_prefix:Predicate is identified as namespace:Predicate at predicateMap in mapping pattern. Each of the triple statement from WHERE clause of SPARQL query indicates a unique predicate which results in different RDF triples.

Algorithm-2 executes Nested Select mapping file from query Q with prefix IRIs and chosen columns of tablename having where clause to another chosen select columns. After the two columns are mapped, table procedure followed by where clause condition is processed and generates query result.

Algorithm-3 uses Join Select query Q with prefix IRIs and overlapped select columns with tablename on Join conditions having where clause condition having Join operators {x; v; m} and generate query result. The operators x, v and m represent three RDB tables to extract dataset.

Algorithm 1 SPARQL-to-R2RML Basic Select Mapping Generation Algorithm, SPARQL_Parser(Q)

Input SPARQL Query 'Q'
Output R2RML Mapping
procedure SPARQL-TO-R2RML
  prefix_name:IRI ⟵ prefix:IRI
  prefix_name:IRI ⟵ ¡#triplesMapprefix_name¿
  prefix_name:IRI ⟵ "tablename"
  for variable in SELECT **do**
    variable$x$ ⟵ SubjectMap $//(x:1)$
    for each variable$i$-to-∞ in SELECT **do**
      "column" ⟵ variable$i$ $//(i:2-to-\infty)$
    end for
  end for
  for prefix:predicate in triples $(1-to-k)$ inside WHERE **do**
    "constant ⟵ prefix:predicate$k$ $//(k:2-to-\infty)$
  end for
end procedure

Algorithm 2 SPARQL-to-R2RML Nested Select Mapping Generation Algorithm, SPARQL_Parser(Q)

Input SPARQL Query 'Q'
Output R2RML Mapping
procedure SPARQL-TO-R2RML
  prefix_name:IRI ⟵ prefix:IRI
  prefix_name:IRI ⟵ ¡#triplesMapprefix_name¿
  prefix_name:IRI ⟵ "tablename-1,tablename-2"
  for all variable in SELECT **do**
    variable$x$ ⟵ SubjectMap $//(x:1)$
    for each variable$i$-to-∞ in SELECT **do**
      "column" ⟵ variable$i$ $//(i:2-to-\infty)$
    end for
  end for
  for all prefix:predicate in triples $(1-to-k)$ inside WHERE **do**
    for variable in SELECT **do**
      variable$x$ ⟵ SubjectMap $//(x:1)$
      for each variable$i$-to-∞ in SELECT **do**
        "column" ⟵ variable$i$ $//(i:2-to-\infty)$
      end for
    end for
    "constant ⟵ prefix:predicate$k$ $//(k:2-to-\infty)$
  end for
end procedure

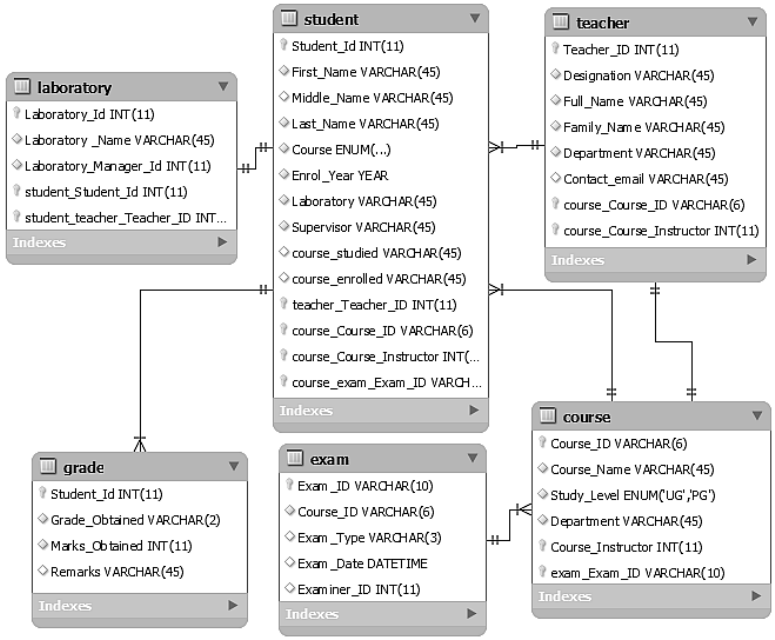

Figure 2. EER Model of Department RDB

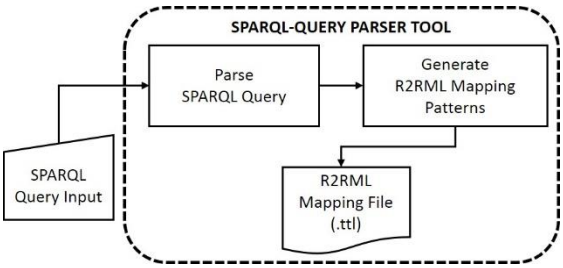

Figure 3. Work-flow of SPARQL-Query Parser

3.3 Generation of R2RML mapping patterns

The generation of R2RML mapping patterns is made at SARM. SARM processes the parsed query and generate custom mapping patterns. The results are stored into a R2RML mapping file in Turtle language (.ttl).

Algorithm 3 SPARQL-to-R2RML Joins Mapping Generation Algorithm, SPARQL_Parser(Q)

Input SPARQL Query 'Q'
Output R2RML Mapping
procedure SPARQL-TO-R2RML
prefix_name:IRI ⟵ prefix:IRI
prefix_name:IRI ⟵ ¡#triplesMapprefix_name¿
prefix_name:IRI ⟵ "tablename-1,tablename-2,tablename-3"
for all variable in SELECT **do**
variable$x$ ⟵ SubjectMap $//(x:1)$
for each variable$i$-to-∞ **in** SELECT **do**
"x:Table-i:column-i" ⟵ variable$i$ $//(i:2-to-\infty)$
"v:Table-i:column-i" ⟵ variable$i$ $//(i:2-to-\infty)$
"m:Table-i:column-i" ⟵ variable$i$ $//(i:2-to-\infty)$
end for
end for
for all prefix:predicate in triples $(1-to-k)$ inside WHERE **do**
"constant(x,v,m) ⟵ prefix:predicate$k$ $//(k:2-to-\infty)$
end for
end procedure

3.4 RDB2RDF data transformation

The data transformation requires processing of R2RML mapping file over RDB2RDF processor. The compiler is programmed to generate RDF views and stores them to the triple store. The processor compiles mapping files in two ways:

1) Compile .ttl file and generate RDF View.

2) Compile .ttl file and store RDF triples over smart meter data repository.

4 EXPERIMENTATION USING SPARQL-QUERY PARSER PROCESS

THIS section is about evaluation of our developed parser. we evaluate SARM over a R2RML processor 'Spyder' (http://www. revelytix.com/content/spyder/), which is an early commercial mapping compiler and supports W3C's standards of R2RML mappings. It is open source software developed by Revelytix and supports MySQL RDB integration for SPARQL query processing. SARM evaluates Basic Select, Nested Select, and Join Select queries and generate RDF triple of student, teacher and course tables in 'Department' RDB. An example for each stated query type is given below:

Basic Select

PREFIX student: <http://example.com/ns#>
SELECT ?Student_Id ?First_Name
?Laboratory ?Supervisor ?Course
Where
?Student_Id student:Student_Name
?First_Name.
?Student_Id student:Lab_Name ?Laboratory.
?Student_Id student:Professor_Name
?Supervisor.
?Student_Id student:Degree_Name ?Course.

Nested Select

PREFIX teacher: <http://example.com/nst#>
PREFIX student: <http://example.com/ns#>
SELECT ?Teacher_Id ?First_Name ?Laboratory
?Department ?Designation
Where
{SELECT ?Student_Id ?First_Name ?Laboratory
?Supervisor ?Course
Where
?Student_Id student:Student_Name
?First_Name.
?Student_Id student:Lab_Name ?Laboratory.
?Student_Id student:Professor_Name
?Supervisor.
?Student_Id student:Degree_Name ?Course.}

Join Select

PREFIX student: <http://example.com/ns#>
PREFIX teacher: <http://example.com/nst#>
PREFIX course: <http://example.com/nsc#>
SELECT ?Student_Id ?First_Name ?Laboratory
?Supervisor ?Course
Where {
?x STUDENT:Student_Id ?student_id.
?x STUDENT:Lab_Name ?lab_name.
?x STUDENT:Lab_ID ?lab_id.
?x STUDENT:Professor_Name ?professor_name.
?v LABORATORY:Lab_ID ?lab_id.
?v LABORATORY:Lab_Name ?lab_name.
?m COURSE:Course_Id ?course_id.
?m COURSE:Student_Id ?student_id.}

When the above mentioned queries are processed by SARM, the R2RML mappings patterns are generated at this stage using Turtle language syntax. The contents of R2RML mapping are shown below.

Basic Select Mapping

@prefix rr:<http://www.w3.org/ns/r2rml#>.
@prefix student: <http://example.com/ns#>.
<#triplesMapstudent>
a rr:TriplesMap;
rr:logicalTable [rr:tableName ''student''];
rr:subjectMap [rr:template
http://www.example.com/student/''Student_Id''
];
rr:predicateObjectMap
[rr:predicateMap [rr:constant
student:Student_Name];
rr:objectMap [rr:column ''First Name''];];
rr:predicateObjectMap
[rr:predicateMap [rr:constant
student:Lab_Name];
rr:objectMap [rr:column ''Laboratory''];];
rr:predicateObjectMap
[rr:predicateMap [rr:constant
student:Professor_Name];
rr:objectMap [rr:column ''Supervisor''];];
rr:predicateObjectMap
[rr:predicateMap [rr:constant
student:Degree_Name];

rr:objectMap [rr:column ‘‘Course’’];].

Nested Select Mapping

```
@prefix rr:<http://www.w3.org/ns/r2rml#>.
@prefix teacher: <http://example.com/ns#>.
<#triplesMapteacher>
a rr:TriplesMap;
rr:logicalTable [rr:tableName ‘‘teacher’’];
rr:subjectMap [rr:template
http://www.example.com/teacher/‘‘Teacher_Id’’
];
rr:predicateObjectMap
[rr:predicateMap [rr:constant
teacher:Teacher_Name];
rr:objectMap [rr:column ‘‘First Name’’]; ];
[rr:predicateMap [rr:constant
teacher:Laboratory];
rr:objectMap [rr:column ‘‘Laboratory Name’’];
];
[rr:predicateMap [rr:constant
teacher:Department];
rr:objectMap [rr:column ‘‘Department Title’’];];
[rr:predicateMap [rr:constant
teacher:Designation];
rr:objectMap [rr:column ‘‘Position’’]; ];
rr:logicalTable [rr:tableName ‘‘student’’];
rr:subjectMap [rr:template
http://www.example.com/student/
‘‘Student_Id’’];
rr:predicateObjectMap
[rr:predicateMap [rr:constant
student:Student_Name];
rr:objectMap [rr:column ‘‘First Name’’];
];
rr:predicateObjectMap
[rr:predicateMap [rr:constant
student:Lab_Name];
rr:objectMap [rr:column ‘‘Laboratory’’];
];
rr:predicateObjectMap
[rr:predicateMap [rr:constant
student:Professor_Name];
rr:objectMap [rr:column ‘‘Supervisor’’];];
rr:predicateObjectMap
[rr:predicateMap [rr:constant
student:Degree_Name];
rr:objectMap [rr:column ‘‘Course’’];].
```

Join Select Mapping

```
@prefix rr:<http://www.w3.org/ns/r2rml#>.
@prefix student: <http://example.com/ns#>.
<#triplesMapstudent>
a rr:TriplesMap;
rr:logicalTable [rr:tableName ‘‘student’’];
rr:subjectMap [rr:template
http://www.example.com/student/
‘‘STUDENT’’ ];
[rr:predicateMap [rr:constant student:Student_Id];
rr:predicateObjectMap
rr:objectMap [rr:column ‘‘student_id’’]; ];
rr:subjectMap [rr:template
http://www.example.com/student/‘‘STUDENT’’];
[rr:predicateMap [rr:constant student:Lab_Name];
rr:predicateObjectMap
rr:objectMap [rr:column ‘‘lab_name’’]; ];
rr:subjectMap [rr:template
http://www.example.com/student/‘‘STUDENT’’ ];
[rr:predicateMap [rr:constant student:Lab_Id];
rr:predicateObjectMap
rr:objectMap [rr:column ‘‘lab_id’’]; ];
rr:subjectMap [rr:template
http://www.example.com/student/‘‘STUDENT’’];
[rr:predicateMap [rr:constant
student:Professor_Name];
rr:predicateObjectMap
rr:objectMap [rr:column ‘‘professor_name’’]; ];
rr:subjectMap [rr:template
http://www.example.com/laboratory/
‘‘LABORATORY’’];
[rr:predicateMap  [rr:constant laboratory:Lab_ID];
rr:predicateObjectMap
rr:objectMap [rr:column ‘‘lab_id’’]; ];
rr:subjectMap [rr:template
http://www.example.com/laboratory/
‘‘LABORATORY’’];
[rr:predicateMap [rr:constant laboratory:Lab_Name];
rr:predicateObjectMap
rr:objectMap [rr:column ‘‘lab_name’’]; ];
rr:subjectMap [rr:template
http://www.example.com/course/‘‘COURSE’’];
[rr:predicateMap [rr:constant course:Course_Id];
rr:predicateObjectMap
rr:objectMap [rr:column ‘‘course_id’’]; ];
rr:subjectMap [rr:template
http://www.example.com/course/‘‘COURSE’’];
[rr:predicateMap [rr:constant course:Student_Id];
rr:predicateObjectMap
rr:objectMap [rr:column ‘‘student_id’’]; ];
```

5 PERFORMANCE EVALUATION

WE evaluate SARM through Berlin SPARQL Benchmark (BSBM) (Bizer & Schultz, 2009), dataset that presents RDB for an e-commerce scenario. BSBM provides most synthetic large-scale datasets which are very similar with industry level data loads. BSBM also provides set of queries to evaluate system under test for RDB2RDF data transformations. The performance metrics used in this evaluation are; (i) mapping generation time, (ii)planning time, and (iii) total execution time for each SPARQL query.

5.1 Evaluation Environment

The experimentation is conducted over a node having Intel Core processor i5-3470 CPU with speed of 3.20 GHz and with 4.00GB RAM, Microsoft Windows 8.1 OS and 1TB Disk Drive. The BSBM benchmark RDB datasets comprises of 50K, 250K, 1M, 5M, and 25M tuples and occupies approximately 4.69MB, 23.7MB, 96.2MB, 481MB, and 2.36GB of physical storage respectively.

5.2 Evaluation Results

The performance evaluation of SARM is conducted over two scenarios:

1) Total time to parse SPARQL query over SARM and generate mapping patterns in Turtle file.

2) Total time to execute Turtle file and retrieve queried RDB data.

The performance matrices to evaluate SARM are: (i) Mapping Generation Time (MGT), (ii) Average Planning Time (APT) and, (iii) Average Total Execution Time (ATET). We use random Basic select queries for BSBM Benchmark RDB tables and adopt respective schema having variation range between 1 to 20 columns. We processed similar set of queries to Spyder R2RML mapping work-flow and obtained results based on two-step procedure as:

Step 1. MGT obtained through customized and entire RDB schema.

Step 2. APT and ATET obtained through execution of ttl file and retrieval of RDB data as RDF views.

We have used five Basic Select SPARQL queries having custom and entire RDB schema of tables: products, features, vendors, customers, and reviews as Queries: 1, 2, 3, 4, and 5, respectively. We evaluate that time taken for parsing and generation of mapping pattern is 0.02, 0.03, 0.02, 0.03, and 0.02 seconds respectively for each query over SARM. The Spyder generates mapping patterns in 11 seconds for entire schema. In the same way, we use another five Nested Select SPARQL queries over same tables.

We evaluate that time taken for parsing and generation of mapping patterns is 0.19 seconds, 0.21 seconds, 0.2 seconds, 0.21, and 0.2 for each query over SARM respectively. The Spyder still generates mapping patterns in 11 seconds. Similarly, we use five Join Select SPARQL queries having custom select and entire RDB schema of tables: products, features, vendors, customers and reviews. We evaluate that time taken for parsing and generation of mapping patterns is 0.45 seconds, 0.47 seconds, 0.43 seconds, 0.47, and 0.46 for each nested query over SARM respectively. The Spyder's time to generate mapping is same as 11 seconds, due to the fact that it generates mapping pattern for entire schema with each processing. The results are observed in Figure 4(a), 5(b) and, 5(c). As evident from Figures 4(a), 4(b) and, 4(c) that SARM reduces a huge time slice in RDB2RDF with each generation of mapping file. It is found as a suitable approach for RDB2RDF over large scale smart data stores, where RDB schema is often evolved and updated. After mapping files are generated through SARM and Spyder, we perform planning procedure using generated map file over R2RML processor. This process identifies and matches mapping patterns with fetched RDB table instances using metadata from R2RML maps and stored procedure calls. We used 5 mapping files of Basic Select SPARQL query to evaluate planning times of generated mapping files over R2RML processor.

We observe that SARM is 18.4% efficient than Spyder in Query-1 map file planning, 8.8% efficient than Spyder in Query-2 map file planning, 9% efficient than Spyder in Query-3 map file planning, 8.7% efficient than Spyder in Query-4 map file planning, and in last, the SARM is observed 7.4% efficient than Spyder in Query-5 map file planning, as seen from Figures 5(a, b, c, d, and e). At last, we execute map files over R2RML processor to retrieve RDB dataset. We evaluate total execution time of the steps involves i.e., (i) Query submission to SPARQL End Point, (ii) SARM map file generation, (iii) Map file planning and execution over R2RML processor, (iv) Generation of SQL and retrieval of RDB dataset over generated SQL. We use 5 Basic Select SPARQL queries to the overall execution steps of customized and entire RDB schema and evaluate average total execution time (ATET). We observe that for Query-1, SARM is observed 9% efficient than Spyder in total execution workflow, 8.6% efficient than Spyder in Query-2 total execution work-flow, 8.9% efficient than Spyder in Query-3, 9.5% efficient than Spyder in Query-4 total execution work-flow, and in last, we observe that SARM is 8.1% efficient than Spyder in Query-5 total execution workflow, as seen from Figures 6(a, b, c, d, and e). After evaluating MGT, ATET and APT, we find a prominent decrease in computing resources of smart meter. Moreover, we observe that smart meter consumes much lower battery energy than traditional RDF tuple generation processing and decrease energy drain with increment of lifespan of smart meter as shown in Table-III. The detailed percentage improvements of SARM over Smart meter battery against each query is also given in details, which clearly shows the significant battery performance improvement using SARM.

Table 3: Impact (percentage improvement) of SARM over Smart Meter Battery

	Query 1	Query 2	Query 3	Query 4	Query 5
MGT	89.6%	92.8%	89.3%	91.4%	90.3%
APT	18.4%	8.8%	9%	8.7%	7.4%
ATET	9%	8.6%	8.9%	9.5%	8.1%
Computing Resource	14.76%	16.29%	18.46%	17.37%	16.38%
Battery usage %	2.7%	3.2%	3.9%	2.9%	3.6%
Battery Drain Decrement	4.8%	4.7%	4.9%	5.1%	4.6%
Battery Lifespan	0.08 year	0.09 year	0.08 year	0.09 year	0.08 year

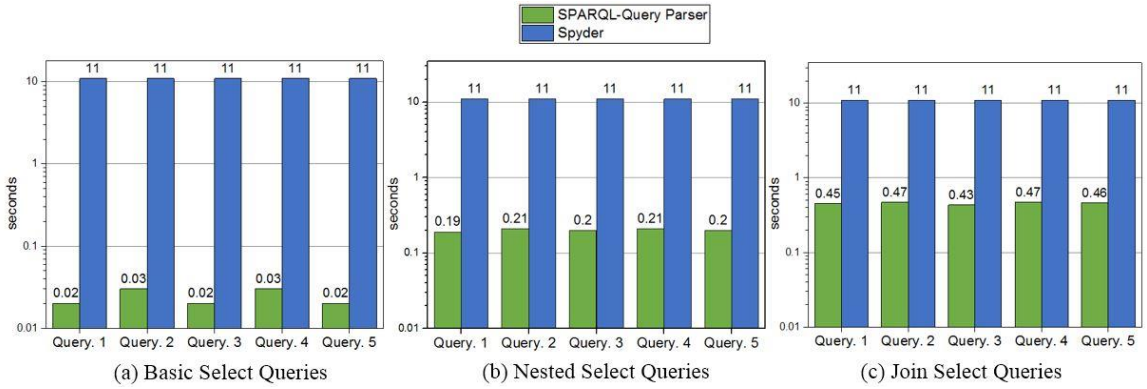

Figure 4 (a) (b) (c). "Mapping Generation Time (MGT)"

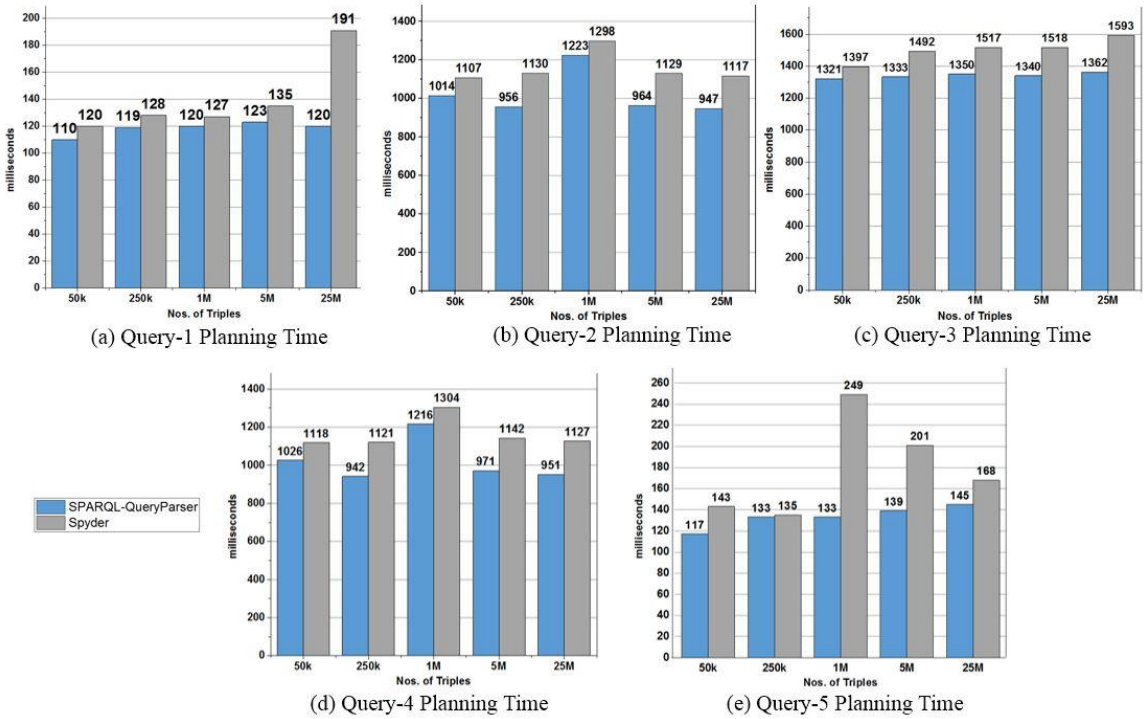

Figure 5. (a), (b), (c), (d) and (e) "Average Planning Time (APT)"

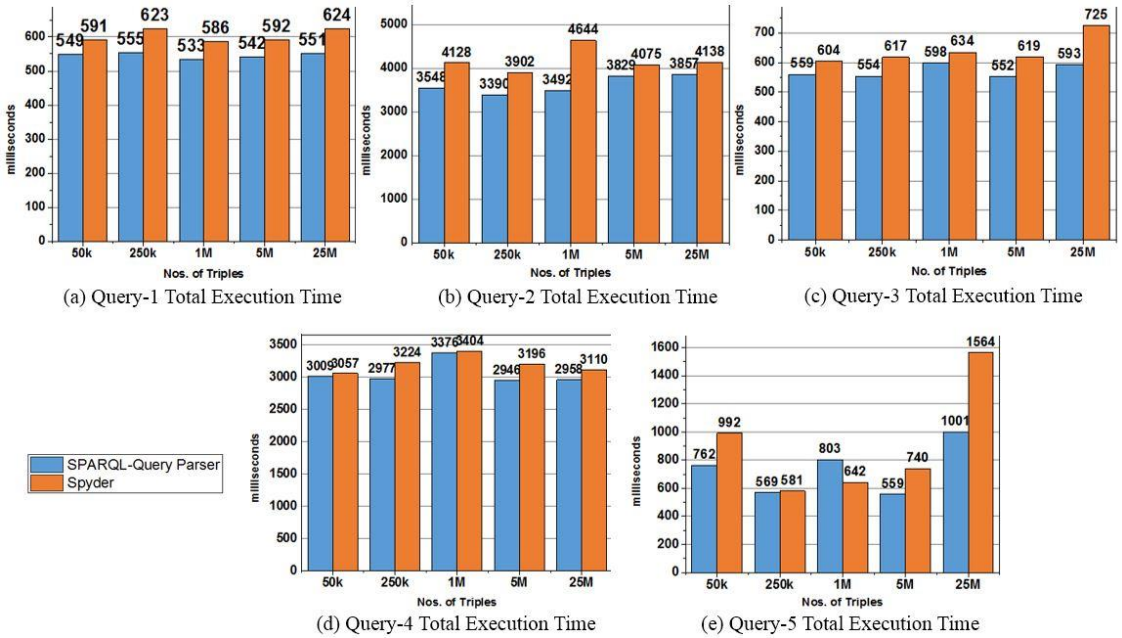

Figure 6. (a), (b), (c), (d) and (e) "Average Total Execution Time (ATET)

6 CONCLUSIONS AND FUTURE WORKS

THE approach presented is towards an effective knowledge-based strategy of reducing battery drain problem in smart meters. The state-of-art smart meters are collecting RDB dataset and generating RDF tuple datasets. The RDF tuple generation involves full schema of RDB and consumes huge amount of battery energy. As a result, batteries are affected with drain problem resulting in low lifespan. We resolve this problem by presenting a knowledge-based solution for data-centric operations for battery data of smart meters. The proposed approach is evaluated and compared with existing state-of-art techniques of RDB2RDF. We observe that SARM significantly decreases APT and ATET time and reduces battery drain problem with an effective increment of lifespan.

In the future, our focus would be towards other semantic-aware knowledgebase solutions for data-centric operations in smart batteries of smart grid.

7 ACKNOWLEDGMENT

THIS work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIP) (No. NRF-2016R1C1B2008624).

8 REFERENCES

Abbas, A., Siddiqui, I. F., Lee, S. U. J., & Bashir, A. K. (2017). Binary pattern for nested cardinality constraints for software product line of IoT-based feature models. IEEE Access, 5, 3971-3980.

Abbas, A., Siddiqui, I. F., Lee, S. U. J., Bashir, A. K., Ejaz, W., & Qureshi, N. M. F. (2018). Multi-objective optimum solutions for IoT-based feature models of software product line. IEEE Access, 6, 12228-12239.

Adida, B., Birbeck, M., McCarron, S., & Herman, I. (2012). RDFa Core 1.1. W3C Recommendation, June 2012. World Wide Web Consortium. http://www. w3. org/TR/2012/REC--rdfa--core--20120607.

Albu, M. M., Sănduleac, M., & Stănescu, C. (2017). Syncretic use of smart meters for power quality monitoring in emerging networks. IEEE Transactions on Smart Grid, 8(1), 485-492.

Amin, S. M., & Wollenberg, B. F. (2005). Toward a smart grid: power delivery for the 21st century. IEEE power and energy magazine, 3(5), 34-41.

Arenas, M., Prud'hommeaux, E., & Sequeda, J. (2011). Direct mapping of relational data to RDF. W3C Working Draft.

Azmat, M., Liaqat, U. W., Qamar, M. U., & Awan, U. K. (2017). Impacts of changing climate and snow cover on the flow regime of Jhelum River, Western Himalayas. Regional environmental change, 17(3), 813-825.

Azmat, M., Qamar, M. U., Huggel, C., & Hussain, E. (2018). Future climate and cryosphere impacts on the hydrology of a scarcely gauged catchment on the Jhelum river basin, Northern Pakistan. Science of the Total Environment, 639, 961-976.

Backes, M., & Meiser, S. (2013). Differentially private smart metering with battery recharging. In Data Privacy Management and Autonomous Spontaneous Security (pp. 194-212). Springer, Berlin, Heidelberg.Wagner, A., Speiser, S., & Harth, A. (2010, November). Semantic web

technologies for a smart energy grid: Requirements and challenges. In In proceedings of 9th International Semantic Web Conference (ISWC2010) (pp. 33-37).

Baig, Z., & Zeadally, S. (2019). Cyber-security risk assessment framework for critical infrastructures. Intelligent automation and soft computing, 25(1), 121-129.

Beckett, D., & McBride, B. (2004). RDF/XML syntax specification (revised). W3C recommendation, 10(2.3).

Bell, C. (2018). Introducing the MySQL 8 Document Store. Apress.

Berners-Lee, T., & Connolly, D. (2011). Notation3 (N3): A readable RDF syntax. W3C team submission, 28.

Bizer, C., & Schultz, A. (2009). The berlin sparql benchmark. International Journal on Semantic Web and Information Systems (IJSWIS), 5(2), 1-24.

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., & Hellmann, S. (2009). DBpedia-A crystallization point for the Web of Data. Web Semantics: science, services and agents on the world wide web, 7(3), 154-165.

Brickley, D. (2000). Resource Description Framework Schema (RDF/S) Specification 1.0. W3C Recommendation.

Bussar, R., Lippert, M., Bonduelle, G., Linke, R., Crugnola, G., Cilia, J., ... & Marckx, E. (2013). Battery energy storage for smart grid applications. Association of European Automotive and Industrial Battery Manufacturers.

Calbimonte, J. P., Corcho, O., & Gray, A. J. (2010, November). Enabling ontology-based access to streaming data sources. In International Semantic Web Conference (pp. 96-111). Springer, Berlin, Heidelberg.

Chhaya, P., Lee, K. H., Shin, K. S., Choi, C. H., Cho, W. S., & Lee, Y. S. (2016, July). Using D2RQ and Ontop to publish relational database as Linked Data. In 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN) (pp. 694-698). IEEE.

Daki, H., El Hannani, A., Aqqal, A., Haidine, A., & Dahbi, A. (2017). Big Data management in smart grid: concepts, requirements and implementation. Journal of Big Data, 4(1), 13.

Das, S., Sundara, S., & Cyganiak, R. (2012). R2RML: RDB to RDF Mapping Language, W3C Working Draft.

Das, S., Sundara, S., & Cyganiak, R. (2012). R2RML: RDB to RDF Mapping Language, W3C Working Draft.

Ghassemi, A., Bavarian, S., & Lampe, L. (2010, October). Cognitive radio for smart grid communications. In 2010 First IEEE International Conference on Smart Grid Communications (pp. 297-302). IEEE.

Gungor, V. C., Sahin, D., Kocak, T., Ergut, S., Buccella, C., Cecati, C., & Hancke, G. P. (2013). A survey on smart grid potential applications and communication requirements. IEEE Transactions on industrial informatics, 9(1), 28-42.

Gungor, V. C., Sahin, D., Kocak, T., Ergut, S., Buccella, C., Cecati, C., & Hancke, G. P. (2013). A survey on smart grid potential applications and communication requirements. IEEE Transactions on industrial informatics, 9(1), 28-42.

Harris, S., & Seaborne, A. (2010). SPARQL 1.1 Query. W3C Working Draft 22 October 2009.

Hazber, M. A., Li, R., Xu, G., & Alalayah, K. M. (2016, August). An approach for automatically generating R2RML-based direct mapping from relational databases. In International Conference of Pioneering Computer Scientists, Engineers and Educators (pp. 151-169). Springer, Singapore.

Kenner, S., Thaler, R., Kucera, M., Volbert, K., & Waas, T. (2017). Comparison of smart grid architectures for monitoring and analyzing power grid data via Modbus and REST. EURASIP Journal on Embedded Systems, 2017(1), 12.

Longe, O., Ouahada, K., Rimer, S., Harutyunyan, A., & Ferreira, H. (2017). Distributed demand side management with battery storage for smart home energy scheduling. Sustainability, 9(1), 120.

Malhotra, A. (2009). W3c rdb2rdf incubator group report. W3C Incubator Group Report.

Pena, A., & Penya, Y. K. (2011). Distributed semantic repositories in smart grids. In 2011 9th IEEE International Conference on Industrial Informatics (pp. 721-726). IEEE.

Pipattanasomporn, M., Feroze, H., & Rahman, S. (2009, March). Multi-agent systems in a distributed smart grid: Design and implementation. In 2009 IEEE/PES Power Systems Conference and Exposition (pp. 1-8). IEEE.

Prud'hommeaux, E., Carothers, G., Beckett, D., & Berners-Lee, T. (2013). Turtle–terse rdf triple language. Candidate Recommendation, W3C, 41.

Qureshi, F. M. N., & Shin, D. R. (2016). RDP: A storage-tier-aware Robust Data Placement strategy for Hadoop in a Cloud-based Heterogeneous Environment. KSII Transactions on Internet & Information Systems, 10(9).

Qureshi, N. M. F., Shin, D. R., Siddiqui, I. F., & Chowdhry, B. S. (2017). Storage-tag-aware scheduler for hadoop cluster. IEEE Access, 5, 13742-13755.

Roberts, B. P., & Sandberg, C. (2011). The role of energy storage in development of smart grids. Proceedings of the IEEE, 99(6), 1139-1144.

Sahoo, S. S., Halb, W., Hellmann, S., Idehen, K., Thibodeau Jr, T., Auer, S., ... & Ezzat, A. (2009). A survey of current approaches for mapping of relational databases to RDF. W3C RDB2RDF Incubator Group Report, 1, 113-130.

Siddiqui, I. F., Abbas, A., & Lee, S. U. J. (2016). A hidden markov model to predict hot socket issue in smart grid. Journal of Theoretical and Applied Information Technology, 94(2), 408.

Siddiqui, I. F., Lee, S. U. J., Abbas, A., & Bashir, A. K. (2017). Optimizing lifespan and energy consumption by smart meters in green-cloud-based smart grids. IEEE Access, 5, 20934-20945.

Siddiqui, I. F., Qureshi, N. M. F., Shaikh, M. A., Chowdhry, B. S., Abbas, A., Bashir, A. K., & Lee, S. U. J. (2018). Stuck-at Fault Analytics of IoT Devices Using Knowledge-based Data Processing Strategy in Smart Grid. Wireless Personal Communications, 1-15.

Sood, V. K., Fischer, D., Eklund, J. M., & Brown, T. (2009, October). Developing a communication infrastructure for the smart grid. In 2009 IEEE Electrical power & energy conference (EPEC) (pp. 1-7). IEEE.

Sporny, M., Kellogg, G., Lanthaler, M., & W3C RDF Working Group. (2014). Json-ld 1.0-a json-based serialization for linked data. W3C Recommendation, 16, 127.

Su, X., Zhang, H., Riekki, J., Keränen, A., Nurminen, J. K., & Du, L. (2014). Connecting IoT sensors to knowledge-based systems by transforming SenML to RDF. Procedia Computer Science, 32, 215-222.

Tuballa, M. L., & Abundo, M. L. (2016). A review of the development of Smart Grid technologies. Renewable and Sustainable Energy Reviews, 59, 710-725.

Wang, X., & Xie, J. (2017, January). An iterative method for accelerated degradation testing data of smart electricity meter. In AIP Conference Proceedings (Vol. 1794, No. 1, p. 040002). AIP Publishing.

Yi, P., Iwayemi, A., & Zhou, C. (2011). Developing ZigBee deployment guideline under WiFi interference for smart grid applications. IEEE transactions on smart grid, 2(1), 110-120.

Yoldaş, Y., Önen, A., Muyeen, S. M., Vasilakos, A. V., & Alan, İ. (2017). Enhancing smart grid with microgrids: Challenges and opportunities. Renewable and Sustainable Energy Reviews, 72, 205-214.

Zheng, J., Gao, D. W., & Lin, L. (2013, April). Smart meters in smart grid: An overview. In 2013 IEEE Green Technologies Conference (GreenTech) (pp. 57-64). IEEE.

Zhou, Q., & Luo, J. (2017). The study on evaluation method of urban network security in the big data era. Intelligent Automation & Soft Computing, 1-6.

Zhou, Q., Simmhan, Y., & Prasanna, V. (2012, November). Incorporating semantic knowledge into dynamic data processing for smart power grids. In International Semantic Web Conference (pp. 257-273). Springer, Berlin, Heidelberg.

9 DISCLOSURE STATEMENT

NO potential conflict of interest was reported by the authors.

10 NOTES ON CONTRIBUTORS

Isma Farah Siddiqui received her Ph.D. in Computer Science and Engineering from Hanyang University, South Korea. Since 2010, she has been an Assistant Professor in Department of Software Engineering at Mehran University of Engineering and Technology, Pakistan. Her research interests include semantic web and semantic data analytics, Big Data analytics, context-aware data processing of Internet of Things and green cloud computing.

Scott Uk-Jin Lee received his Ph.D. degree in computer science from the University of Auckland, New Zealand. After the doctoral degree, he worked as post-doctoral research fellow at Commissariat à l'énergie atomique et aux énergies alternatives (CEA), France. He is currently an associate professor in the Department of Computer Science and Engineering at Hanyang University, Republic of Korea. His research interests include Software Engineering, Formal Methods, Software Quality Management, Software Product Line, Requirement Engineering, Web, Semantic Web, CyberSecurity and Internet of Things.

Asad Abbas received his BS degree in Information Technology from University of Punjab Pakistan in 2011. He received scholarship for MS leading to Ph.D. program from Higher Education Commission (HEC) Pakistan in 2014 and received Ph.D. degree in Computer Science and Engineering from Hanyang University South Korea in 2018. Currently, he is serving as Assistant Professor at Software Engineering Department, University of Lahore, Lahore, Pakistan. His research interests include Software Product Line, Software Requirement Traceability and IoT Applications.