ExaQuery: Proving Data Structure to Unstructured Telemetry Data in Large-Scale HPC

ExaQuery: Proving Data Structure to Unstructured Telemetry Data in Large-Scale HPC

Junaid Ahmed Khan
DEI Department, University of Bologna
Italy
junaidahmed.khan@unibo.it

Martin Molan
DEI Department, University of Bologna
Italy
martin.molan2@unibo.it

Matteo Angelinelli
HPC Department, CINECA
Italy
m.angelinelli@cineca.it

Andrea Bartolini
DEI Department, University of Bologna
Italy
a.bartolini@unibo.it

ABSTRACT

High-performance computing (HPC) is the cornerstone of technological advancements in our digital age, but its management is becoming increasingly challenging, particularly as systems approach exascale. Operational data analytics (ODA) and holistic monitoring frameworks aim to alleviate this burden by collecting live telemetry from HPC systems. ODA frameworks rely on NoSQL databases for scalability, with implicit data structures embedded in metric names, necessitating domain knowledge for navigating telemetry data relations. To address the imperative need for explicit representation of relations in telemetry data, we propose a novel ontology for ODA, which we apply to a real HPC installation. The proposed ontology captures relationships between topological components and links hardware components(compute nodes, rack, systems) with job's execution and allocations collected telemetry. This ontology forms the basis for constructing a knowledge graph, enabling graph queries for ODA. Moreover, we propose a comparative analysis of the complexity (expressed in lines of code) and domain knowledge requirement (qualitatively assessed by informed end-users) of complex query implementation with the proposed method and NoSQL methods commonly employed in today's ODAs. We focused on six queries informed by facility managers' daily operations, aiming to benefit not only facility managers but also system administrators and user support. Our comparative analysis demonstrates that the proposed ontology facilitates the implementation of complex queries with significantly fewer lines of code and domain knowledge required as compared to NoSQL methods.

CCS CONCEPTS

• **Information systems → Resource Description Framework (RDF)**; • **Computing methodologies → Ontology engineering**.

KEYWORDS

High performance computing (HPC), Operational Data Analytics(ODA), Resource Description Framework (RDF) ontology, SPARQL

ICPE '24 Companion, May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652898

ACM Reference Format:
Junaid Ahmed Khan, Martin Molan, Matteo Angelinelli, and Andrea Bartolini. 2024. ExaQuery: Proving Data Structure to Unstructured Telemetry Data in Large-Scale HPC. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3652898

1 INTRODUCTION

The rise in complexity of large-scale computing infrastructures driven by post Moore's and Dennard's scaling era presents unprecedented challenges. Key challenges include efficient power management, optimization for parallelism, data movement and storage, software complexity, fault tolerance, scalability, workload diversity, resource allocation, and security. Many data centers explore Operational Data Analytics (ODA) to extract knowledge from monitoring data, enabling control over system parameters and aiding administrators through visualization. Despite extensive research into individual aspects of ODA, comprehensive solutions for production remain rare, particularly given the inherent complexity of HPC[9, 13].

HPC is operated by multiple teams and organizations, each tasked with distinct responsibilities for production. This includes system administrators, facility managers, and user support, who collectively contribute to its efficient operation and management. ODA targets holistic management, where the data includes diverse types such as job tables, sensor time-series data, and other varied representations ranging from log files and configuration files to system metadata. ODA frameworks often rely on NoSQL databases as they allow flexibility with diverse data sources and scalability to handle big data frameworks[11]. Moreover, namespaces adopted in ODA are tailored to the specifics of vendors, sites, or configurations, jeopardizing the portability of knowledge extraction solutions.

Acquiring domain knowledge presents a formidable challenge, as it often relies on undisclosed or dispersed information within various organizations and teams managing similar resources, leading to a fragmented understanding. In ODA, data demonstrates interconnectivity and the true value lies in identifying and harnessing these complex relationships. These relationships encompass various aspects, including the interactions between system components, submitted jobs, their execution on specific compute nodes, event correlations, and topology mapping.

In this work, we propose the first ontology aiming to provide a structured data model that captures these intricate relationships. The current state-of-the-art data center ontologies focus on inventory and infrastructure[4, 5], while the proposed ontology goes further by incorporating topological component relationships and establishing links between hardware components (such as compute nodes, racks, and systems) and job data. This ontology serves as the foundation for constructing a knowledge graph, providing a structured representation of ODA data, facilitating organized retrieval of interconnected data using graph queries. This ontology has been developed specifically for the CINECA Italian Tier-0 supercomputing center[15]. We utilized the Marconi100 (M100) system at CINECA, which employs the Examon ODA framework for holistic monitoring (detailed in sec.3), operating on Cassandra DB and KairosDB (a NoSQL time-series database), utilizing an encoded version of metric names and properties as column names. The results of this manuscript were obtained using a subset of publicly available M100 Examon collected data [3]

Furthermore, this manuscript includes a comparative analysis of query implementation complexity, measured in lines of code (LOC), and domain knowledge required between ontology-based approaches and NoSQL methods. A lower LOC indicates simpler code, while qualitative assessment of domain knowledge requirements is pivotal in determining the user-friendliness of the proposed ontology. The objective is to underscore the significance of ontologies for ODA and illustrate how they can facilitate ODA for HPC.

2 RELATED WORK

In this manuscript, we target the development of ontologies for data centers and HPC suitable for ODAs telemetry. With this regard, Oscar Corcho et al.[5] identify a lack of comprehensive implementations and common data models not only in this field but also across other ICT infrastructure areas. Their work is deemed impactful, showcasing the practical use of ontologies in managing data heterogeneity. Gabriel G. Castañé et al.[4], propose an ontology integrating HPC and cloud. However, its emphasis on HPC-cloud interrelations may limit its relevance to our specific requirement of simplifying query of telemetry data in HPC. Liao et al.[7] introduce an HPC ontology to ensure FAIRness (Findable, Accessible, Interoperable, Reusable) of training datasets and AI models on heterogeneous supercomputers. Their ontology offers controlled vocabularies and formal knowledge representations for data annotation and SPARQL query support, which is not the target of the proposed manuscript. Kousha et al.[6] focus on an HPC ontology tailored for job script submission and AI-assisted tools, unlike this paper which concentrates on ODA telemetry data retrieval. Additionally, Tuovinen et al.[14] present an HPC ontology to make a unified framework capable of adapting queries across different time-series storages. In contrast, the ontology proposed in this manuscript is designed to address a specific set of queries essential for the daily operations of an HPC facility manager/engineer. The aim is to simplify query implementations and reduce the required domain knowledge compared to NoSQL approaches. Additionally, we validate our approach through a comparative analysis to demonstrate its simplicity, thus proving the adoption of data structures to handle unstructured telemetry data in large-scale HPC.

3 BACKGROUND: EXAMON

Examon is a holistic monitoring framework for HPC[2]. It is designed to collect data from various sources, including hardware sensors, software logs, and performance metrics, and stores this data in a NoSQL database (Cassandra, with KairosDB for time-series) in a centralized repository.

Examon's data collection targets a diverse range of sources, as depicted in (Fig.1). The complexity of the collected data encompasses hardware sensors—such as CPU load across all cores, CPU clock, instructions per second, memory accesses, power consumption, fan speed, and ambient and component temperatures—along with workload-related information like job submissions and their characteristics. Additionally, Examon actively monitors compute node availability by capturing warning messages and alarms from diagnostic software tools used by system administrators. The figure further illustrates the granularity of Examon's approach, showcasing separate plugins for each hardware component, each equipped with specific sensors. This design underscores Examon's capacity to manage diverse data sources, contributing to its inherent capability to handle massive data complexity in monitoring. The openly available dataset by Borghesi et al.[3] covers a spectrum of metrics, from hardware parameters to system-related statistics.

Furthermore, Examon employs a specific set of parameters and tags, and to interact with its dataset, it features a dedicated query language known as ExamonQL. This language allows users to access information stored in the database, including metadata, and generate dataframes of the queried results.

4 METHODOLOGY

The methodology involves creating a knowledge graph aligned with Examon's operational principles. This section details the proposed ontology, its specifications, query language for ontology, complex queries for comparison with ExamonQL, and the evaluation criteria for the comparative analysis.

4.1 ODA ontology

In this subsection, we outline the reasoning behind the proposed ontology, followed by its explanation. The Resource Description Framework (RDF) plays a central role in this context, being a web standard essential for ontologies and knowledge graphs. Employing a triple structure—comprising subject, predicate, and object—RDF efficiently represents relationships. In RDF, the Uniform Resource Identifier(URI) uniquely identifies resources, such as classes and properties. These URIs can be in the form of Uniform Resource Locator(URL), providing the means to locate a resource on the internet. In the context of the proposed paper, the resources refer to components and telemetry data. RDF's flexibility in accommodating both literal values and resource descriptions makes it an invaluable tool for constructing ontologies, providing structured models to define concepts and their relationships[8].

4.1.1 Reasoning behind ontology. The proposed ontology follows a novel approach that exploits the holistic nature of ODA's (and

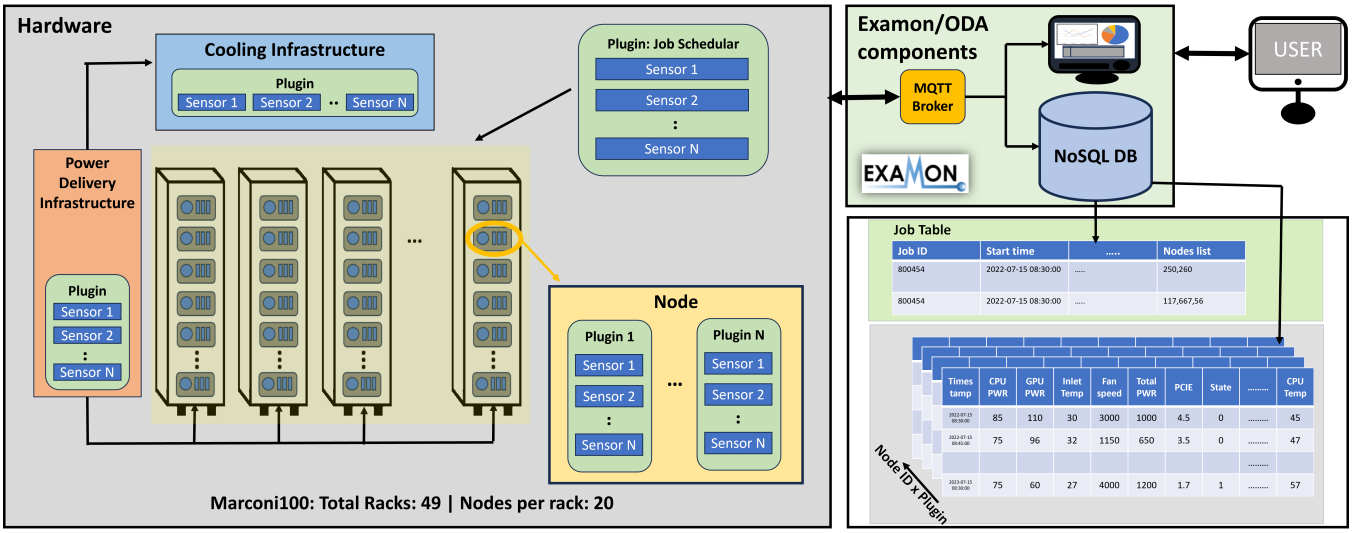

Figure 1: Examon's massive scale and data heterogeneity.

Examon's) monitoring data and the natural ability of knowledge graphs to capture relationships between data. As this ontology is designed to facilitate the work of large-scale HPC center data analysts and facility managers, it is designed to best meet the needs of these users. While Examon is a very powerful tool for holistic monitoring, it requires a thorough knowledge of the data architecture itself. With the proposed ontology, data is organized in a structure that allows easy interrogation by end users. In particular, as will be shown in the following sections, the data analysis process is greatly simplified, allowing a data-driven usage, management, and optimization of supercomputer systems production with workloads such as those proposed by Molan et al.[10].

4.1.2 Ontology creation process. The proposed ontology is developed to establish logical connections among the various data sources within Marconi100, as perceived by system administrators such as facility engineers and managers. Aligned with the underlying principles of Examon (see sec.3), it caters to the meticulous organization of telemetry data illustrated in Figure 1. In Examon, telemetry data is structured in a Plugin-centric manner, with specific plugins housing sensors tailored to each resource within the facility, be it a compute node or a component of the cooling infrastructure. These sensors gather data, which is then stored in individual files within their respective folders in the database, following a clear pathway from Plugin to Sensor to Sensor Reading, culminating in a storage file termed as a "Data Record" within our proposed ontology (see Fig.2).

However, Examon lacks inherent topological information crucial for understanding the physical organization and location of resources, particularly significant for workloads involving graph processing[10]. In an HPC facility, the natural topological structure typically revolves around compute nodes housed in racks, each rack assigned a physical location in the x and y dimensions, with compute nodes stacked within. Consequently, the position of a compute node within the stack becomes the third dimension, denoted as "Position" in our proposed ontology.

Moreover, an integral aspect of any HPC system is the jobs submitted to it. Therefore, our proposed ontology incorporates job-specific information, establishing a natural linkage between submitted jobs and the resources they utilize, which are compute nodes. This holistic approach creates a unified framework wherein every resource within the HPC facility is interconnected with its logical connections—an aspect lacking in the monitoring framework of Examon.

4.1.3 Proposed Ontology. The proposed ontology (Fig.2) presents a significant improvement for ODA in HPC. This structured framework organizes elements such as racks, nodes, positional information, plugin-specific sensors, and their readings. It establishes explicit relationships between HPC and ODA components, including a specific link between submitted job and the resources utilized, a feature lacking in other approaches[4, 5]. The proposed ontology provides a comprehensive model for integrating and understanding sensor data, spatial configurations, job execution, and deployed software/hardware components status in HPC infrastructure.

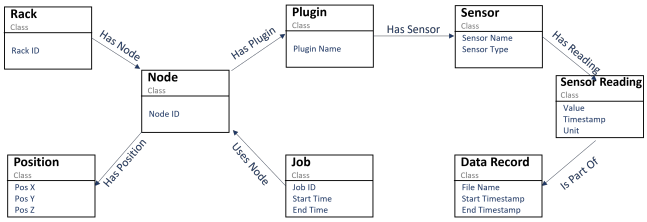

Figure 2: Proposed Ontology

HPC cluster topology consists of multiple racks, each housing a set of compute nodes. ODA frameworks[11] also organize data into different plugins (9 in Examon: Nagios, Ganglia, IPMI, Job table, Slurm, etc), each linked to its corresponding monitored sensors. The proposed ontology mirrors these observations by representing physical components as classes and capturing associated information through properties. Relationships between classes are precisely

defined, aligning with the arrangement of plugins and sensors of Marconi100 and Examon.

Table 1 provides an overview of the proposed ontology's classes and their attributes, where each class represents a component within the HPC system. Table 2 reports the properties of the proposed ontology, outlining their roles and functionalities, which establish relationships between classes.

Class Name	Description
Sensor	Represents individual sensors with attributes: `sensorName` and `sensorType`.
Sensor Reading	Captures sensor data with attributes: `value`, `timestamp`, and `unit`.
Plugin	Represents specific plugins, identified by `pluginName`.
Job	Represents job information with attributes: `jobId`, `startTime`, and `endTime`.
Rack	Represents physical racks with a unique `rackId`, that houses nodes.
Node	Represents individual compute nodes with a unique `nodeId`
Position	Defines spatial coordinates (`posX`, `posY`, `posZ`) of nodes.
Data Record	Represents stored data records in database with attributes: `fileName`, `startTimestamp`, and `endTimestamp`.

Table 1: Classes Overview

Property	Description
Has Plugin	Connects nodes to its plugins.
Has Sensor	Links plugins to associated sensors in the HPC infrastructure.
Has Reading	Connects sensors to its readings.
Uses Node	Associates jobs with the nodes utilized during the job's execution.
Has Node	Establishes a relationship between racks and nodes.
Has Position	Connects nodes to spatial coordinates (X, Y, Z) representing their physical location.
Is Part Of	Links sensor readings to data record, specifying their inclusion in specific files in storage.

Table 2: Properties

4.2 Knowledge graph: Ontology realisation

Ontology is a structured way of representing knowledge, defining concepts and relationships. Meanwhile, a knowledge graph is a graph-based structure built upon the schema set by the ontology, representing information in nodes(components) and edges(relations between components). By constructing a knowledge graph based on the proposed ontology, we enable the implementation of graph queries. These queries would be utilized for the comparative analysis between NoSQL methods. The evaluation criteria are explained in the (sec.4.5).

4.3 ODA Complex Queries

Table 3 reports the complex ODA queries. Query 1,2,3 targets anomaly detection and prediction models that leverage node's proximity information and advance graph algorithms, like[10]. Query 4,5,6 targets the extraction of insights from job data. Overall, these queries are instrumental for root cause analysis of anomalous behaviors arising from the submitted jobs. By delving into job-related data, the aim is to pinpoint irregularities, understand their origins, and ultimately contribute to the reduction of anomalies in HPC operations. This approach aligns with the overarching goal of efficient management of HPC systems through data-driven analytics and insights derived from complex ODA queries.

No.	Query	Description
1	Generate adjacency matrix, each node connected to the closest nodes in a rack.	Finds closest nodes in the same rack and constructs an adjacency matrix.
2	Generate adjacency matrix for the entire compute room, each node connected to nearest neighbors in the 3 dimensions.	Identifies nodes in proximity in the entire compute room to form an adjacency matrix.
3	Generate adjacency matrix for nodes running the same compute job.	Finds node running the same compute job and forming its adjacency matrix.
4	Average job power.	Calculates average power consumption for a specific job.
5	How many jobs are running in a particular node, over a time-period.	Count the number of jobs running on a specific node over a time period.
6	What is the min, max, average temperature when a node is in use, over a given time-period.	Computes temperature statistics of the node in use during a specified period.

Table 3: Selected Queries

4.4 Ontology query language: SPARQL

SPARQL is the preferred choice for ontology querying due to its seamless compatibility with the RDF structure. SPARQL aligns well with RDF's graph representation, making it ideal for expressing and retrieving information from knowledge graphs. Its triple pattern matching capability allows precise matching within RDF triples, enabling users to specify complex relationships. SPARQL's expressiveness and flexibility make it a powerful tool for crafting tailored queries. Standardized by the W3C, SPARQL ensures consistency and interoperability, establishing itself as a state-of-the-art solution for RDF querying[1, 12].

4.5 Evaluation criteria

The evaluation of each query primarily focuses on its simplicity and conciseness. This involves a thorough examination of the complexity, indicated by the Lines of Code(LOC) required for each query. Additionally, the assessment considers the level of domain-specific knowledge necessary for executing the query effectively. A crucial aspect of the evaluation is determining the comprehensibility of each query for individuals with limited to no direct domain knowledge. Traditional metrics such as time to execution and data fetches are not applicable in this context. The knowledge graph based on the proposed ontology resides locally, while the Examon query retrieves information directly from the real Examon installation and its remote database. Consequently, the execution time won't be utilized as a comparative metric in our evaluation. Similarly, regarding data fetches, the extensive historical data in Examon makes the volume queried substantially larger than the minimal

RDF instances (described in sec.5.1) created for experimental purposes. Hence, these metrics are not considered in our evaluation approach.

5 EXPERIMENTAL EVALUATION

5.1 Experiment setup

The knowledge graph using the proposed ODA ontology is created in the TURTLE(.ttl) format. SPARQL and Examon queries are executed in a Python environment. Examon, being operational with accessible historical data, allows retrieval of genuine historical data. Examon utilizes its specific query library, ExamonQL, while RDF and SPARQL execution in Python relies on the RDFlib library.

To initiate the process, we load the .ttl ontology file and populate the RDF graph by traversing the tables of examon's historical data and selecting small batches of a few instances from each table and expressing them in the RDF triple format, thereby constructing the knowledge graph referred to as combined_graph in these queries. The PREFIX at the start of each query serves as a unique identifier for the entire ontology, with each component's identifier as its extension. The PREFIX remains consistent in all SPARQL queries and is explicitly defined as follows: "cineca_m100" is the prefix for the ontology with its base Unique Resource Identifier (URI), "rdf" is the prefix for the RDF namespace, and "xsd" is the prefix for the XML Schema namespace, used for defining datatypes in RDF. These prefixes simplify the notation in SPARQL queries by providing shorthand representations for longer URIs.

5.2 Query implementation

In this section, we will analyze the implementation of each query in both SPARQL and ExamonQL, providing a detailed comparison

5.2.1 Query 1: Generate adjacency matrix, each node connected to the closest nodes in a rack and Query 2: Generate adjacency matrix for the entire compute room, each node connected to nearest neighbors in the 3 dimensions. These two queries are centered around obtaining topological information, specifically in the context of identifying compute nodes in close physical proximity. This focus is crucial for graph-based machine learning and artificial intelligence, where precise spatial information is essential for generating adjacency matrices. It's noteworthy that these two queries are not feasible to execute using Examon due to the absence of spatial information in Examon. We present the SPARQL query aligned with the proposed ontology (Fig. 2) for further exploration. This process involves retrieving the positions of all nodes within a rack and presenting the results.

```
query = f"""SELECT ?node ?nodeId ?pos
WHERE {{
 cineca_m100:rack{rack_number} cineca_m100:
    hasNode ?node .
 ?node cineca_m100:nodeId ?nodeId .
 ?node cineca_m100:hasPosition ?pos .
}}"""
```

Query 1 and 2: SPARQL

The final manipulation process may differ based on different edge connectivity strategies. We combine the first two queries into a single subsection due to their similarity and shared requirements. Notably, the semantic nature of this query establishes a hierarchy, starting from identifying the target rack to its nodes and positions. SPARQL's semantic clarity enables intuitive understanding, even for individuals with limited domain knowledge familiar with the ontology and its basic concepts.

5.2.2 Query 3: Generate adjacency matrix for nodes running the same compute job. This query focuses on job-specific analysis and the direct linkage in the proposed ontology between job and nodes makes its implementation simpler (lower LOC count, fewer parameters and namespaces based on proposed ontology which are not specific to an ODA framework or HPC facility) than in ExamonQL. This structure can be utilized as follows by identifying the job by its "job_id" and examining its "usesNode" property to retrieve the list of nodes where this job was executed. Whereas in Examon, accessing specific data is more intricate due to the absence of direct relations between its ODA components. Retrieving particular information necessitates a deep understanding of Examon and its heterogeneous data types. Users must possess domain knowledge (covering both ODA's data types and HPC internal structure) to identify the relevant data source, determine which data table holds the needed information, and navigate the complete ODA framework to access the necessary data.

```
query = f"""SELECT ?node ?nodeId
WHERE {{
    cineca_m100:Job{job_id} cineca_m100:usesNode
    ?node .
    ?node cineca_m100:nodeId ?nodeId .
}}"""
```

Query 3: SPARQL

```
#Setup for Marconi100
sq.jc.JOB_TABLES.add('job_info_marconi100')
data = sq.SELECT('*').FROM('job_info_marconi100')
    .WHERE(job_id={jobId}).TSTART({job_start_time
    }).TSTOP({job_end_time}).execute()
#create dataframe of the query result
df = pd.read_json(data)
print(df['cpus_alloc_layout'][0])
```

Query 3: Examon

5.2.3 Query 4: Average job power. Implementation of this query in SPARQL begins by identifying the nodes used and retrieving start and end times for a job. It then follows a relationship pathway from these nodes to their associated plugins and subsequently to their sensors. In this particular instance, the query selects the "total_power" sensor. Following this, the query proceeds to collect all readings from the selected sensor and apply a filter based on the job's timestamp to narrow down the readings to those within the job's specified period. Finally, the query concludes by grouping each node's values using the groupby command.

```
query = f"""SELECT ?nodeId ?unit (AVG(?powerValue
    ) AS ?averagePower)
WHERE {{
 cineca_m100:Job{job_id} cineca_m100:usesNode ?
    node ;
    cineca_m100:startTime ?startTime ;
    cineca_m100:endTime ?endTime .
```

```
 ?node cineca_m100:hasPlugin/cineca_m100:
    hasSensor ?sensor ;
    cineca_m100:nodeId ?nodeId .
 ?sensor cineca_m100:sensorName "total_power" ;
      cineca_m100:hasReading ?reading .
 ?reading cineca_m100:value ?powerValue ;
       cineca_m100:timestamp ?timestamp ;
       cineca_m100:unit ?unit .
 FILTER(?timestamp >= ?startTime && ?timestamp <=
     ?endTime)
}} GROUP BY ?nodeId"""
```

Query 4: SPARQL

In implementing this query in ExamonQL, we observe that the number of lines for both query types is almost the same, yet it appears more complex than the SPARQL query. The complexity arises because there is no inherent relationship between data sources in Examon, which requires the user to connect the dots, necessitating the users to be well-acquainted with each separate data source, its tables, and the contents of each table to successfully execute this query. The user has to navigate through different data sources and establish the necessary connections manually. To facilitate this process, the use of helper functions in Python becomes essential, further contributing to the complexity of the query implementation. In Examon, two sub-queries are required: one to gather job-related data and another to retrieve sensor readings. Users must integrate job information from the first sub-query into the second to obtain the final value. This multi-step process adds complexity compared to the straightforward SPARQL query.

```
def get_data(node_to_get,start_time,end_time):
  data = sq.SELECT('*') \
    .FROM('total_power').WHERE(node=node_to_get).
    TSTART(str(start_time)).TSTOP(str(end_time)).
    execute()
  value = data.df_table['value']
  return value
sq.jc.JOB_TABLES.add('job_info_marconi100')
data = sq.SELECT('*') \
    .FROM('job_info_marconi100').WHERE(job_id=
    jobId).TSTART({start_time}).TSTOP({end_time})
    .execute()
# create df of the query result
df = pd.read_json(data)
# get the allocated nodes list
dict_of_nodes = df['cpus_alloc_layout'][0]
nodes = list(dict_of_nodes.keys())
start_time = format_TS(str(df['start_time'][0]))
end_time = format_TS(str(df['end_time'][0]))
for node in nodes:
  df = get_data(node,start_time,end_time)
  avg = df.sum()/len(df)
```

Query 4: Examon

5.2.4 Query 5: How many jobs are running in a particular node, over a time-period. The implementations clearly show that the SPARQL query requires fewer lines of code (LOC) compared to the ExamonQL query. This pattern aligns with the observation in Query 4, where a single SPARQL query is used instead of two ExamonQL subqueries due to the lack of connection between separate data sources in Examon. Moreover, the semantic nature of SPARQL provides a logical structure that is easier to understand for individuals with a basic understanding of the proposed ODA ontology. In contrast, the ExamonQL implementation underscores the necessity of domain knowledge to achieve the desired output.

```
query = f"""SELECT (COUNT(?job) as ?jobCount)
WHERE {{
 ?job a cineca_m100:Job ;
 cineca_m100:usesNode cineca_m100:Node{node_id} ;
 cineca_m100:startTime ?startTime ;
 cineca_m100:endTime ?endTime .
 FILTER(?startTime <= "{end_time}"^^xsd:dateTime
    && ?endTime >= "{start_time}"^^xsd:dateTime)
}}"""
```

Query 5: SPARQL

```
def get_nodes_list(jobId,time_period):
  data = sq.SELECT('*')\
  .FROM('job_info_marconi100')\
  .WHERE(job_id=str(jobId))\
  .TSTART(time_period[0]).TSTOP(time_period[1]).
    execute()
  # create df of the query result
  df = pd.read_json(data)
  # get the allocated nodes list
  dict_of_nodes = df['cpus_alloc_layout'][0]
  try: nodes = list(dict_of_nodes.keys())
  except: pass
  # create df of the query result
  df = pd.read_json(data)
  df['cpus_alloc_layout'][0]
  nodes = list(dict_of_nodes.keys())
  return nodes
# Setup for Marconi100
sq.jc.JOB_TABLES.add('job_info_marconi100')
data = sq.SELECT('*') \
    .FROM('job_info_marconi100').TSTART({
    start_time}).TSTOP({end_time}).execute()
df = pd.read_json(data)
job_ids = df['job_id'].to_numpy()
count = 0
for job_id in job_ids:
  try: nodes_list = get_nodes_list(job_id,
    time_period)
       count += nodes_list.count(node_to_check)
  except: pass
```

Query 5: Examon

5.2.5 Query 6: What is the min, max, average temperature when a node is in use, over a given time-period. In the implementation of this query, we can see that ExamonQL requires a lot more lines of code(LOC) as compared to SPARQL. Additionally, in the case of ExamonQL for this query, the necessity for four sub-queries to obtain the final results further emphasizes the increased complexity in comparison to the more concise SPARQL implementation.

```
query = f"""SELECT ?nodeId (AVG(?temperature) as
    ?avgTemperature) (MIN(?temperature) as ?
    minTemperature) (MAX(?temperature) as ?
    maxTemperature)
WHERE {{
```

```
 ?job rdf:type cineca_m100:Job ;
    cineca_m100:startTime ?jobStart ;
    cineca_m100:endTime ?jobEnd ;
    cineca_m100:usesNode ?node .
 ?node cineca_m100:hasPlugin/cineca_m100:
    hasSensor ?sensor ;
    cineca_m100:nodeId ?nodeId .
 ?sensor cineca_m100:sensorName "temperature" ;
    cineca_m100:hasReading ?reading .
 ?reading cineca_m100:value ?temperature ;
    cineca_m100:timestamp ?timestamp ;
    cineca_m100:unit ?unit .
 FILTER(?jobStart <= "{end_time}"^^xsd:dateTime
    && ?jobEnd >= "{start_time}"^^xsd:dateTime)
}}GROUP BY ?nodeId"""
```

Query 6: SPARQL

```
def get_nodes_list(jobId,time_period):
  data = sq.SELECT('*') \
    .FROM('job_info_marconi100').WHERE(job_id=str
    (jobId))\
    .TSTART(time_period[0]).TSTOP(time_period[1])
    .execute()
  # create df of the query result
  df = pd.read_json(data)
  # get the allocated nodes list
  dict_of_nodes = df['cpus_alloc_layout'][0]
  try: nodes = list(dict_of_nodes.keys())
  except: pass
  # create df of the query result
  df = pd.read_json(data)
  df['cpus_alloc_layout'][0]
  nodes = list(dict_of_nodes.keys())
  return nodes
# Setup for Marconi100
sq.jc.JOB_TABLES.add('job_info_marconi100')
data = sq.SELECT('*').FROM('job_info_marconi100')
    .TSTART({start_time}).TSTOP({end_time})\
    .execute()
df = pd.read_json(data)
job_ids = df['job_id'].to_numpy()
node_used_in_job_list = []
for job_id in job_ids:
  try: nodes_list = get_nodes_list(job_id,
    time_period)
    if (node_to_check in nodes_list):
      print(job_id,nodes_list)
      node_used_in_job_list.append(job_id)
  except: pass
def get_data(node_to_get,metric,start_time,
    end_time):
  data = sq.SELECT('*').FROM(metric).WHERE(node=
    node_to_get).TSTART(str(start_time)).TSTOP(
    str(end_time)).execute()
  value = data.df_table['value']
  return value
def get_job_time(jobId):
  data=sq.SELECT('*').\
  FROM('job_info_marconi100')\
  .WHERE(job_id=str(jobId),node=node_to_check)\
  .TSTART({start_time}).TSTOP({end_time})\
  .execute()
  df = pd.read_json(data)
  start_time= format_TS(str(df['start_time'][0]))
  end_time= format_TS(str(df['end_time'][0]))
  return start_time,end_time
each_job_df = []
for job in node_used_in_job_list:
  start_time,end_time = get_job_time(job)
  try:
    df = get_data(node_to_check,metric,start_time
    ,end_time)
    each_job_df.append((max(df),min(df),(df.sum()
    /len(df))))
  except: print("error")
```

Query 6: Examon

6 DISCUSSION

The evaluation of SPARQL queries against ExamonQL provides valuable insights into their efficiency and usability for querying topological information and conducting job-specific analyses within HPC environments (see sec. 5.2). In queries 1 and 2, SPARQL's semantic clarity and alignment with the proposed ontology enable intuitive querying, starting from rack identification to node positions. In contrast, Examon lacks spatial information, rendering such queries unfeasible in ExamonQL. For queries 3, 4, 5, and 6, the direct linkage between jobs and their utilized nodes in the proposed ontology simplifies query implementation, resulting in fewer lines of code and reduced complexity compared to ExamonQL. Additionally, SPARQL's filtering capabilities lead to a more concise and logical query structure, whereas ExamonQL's fragmented queries lead to increased complexity.

Overall, SPARQL consistently demonstrates advantages in efficiency and usability across all six queries, offering a structured framework that simplifies query development and comprehension. In contrast, ExamonQL's manual connection requirements and fragmented querying pose challenges for users, necessitating a deeper understanding of the underlying connectivity between different data sources.

7 CONCLUSION

In this manuscript, we presented an ontology for ODA and a comparative analysis with state-of-the-art ODA methods. The comparative analysis of complex ODA queries implemented in Examon and SPARQL sheds light on the practical applicability of SPARQL, showcasing its efficiency and clarity in query execution(fewer LOC and less domain knowledge requirements). SPARQL's semantic nature allows users to comprehend queries by following the logical structure outlined in the proposed ontology. With even basic knowledge of the proposed ontology, its classes, and relationships, users can easily grasp the query's intent. This feature enhances accessibility and comprehension without necessitating extensive domain expertise. SPARQL query seamlessly aligns with the inherent relations in the HPC data, making queries transparent and aiding a straightforward understanding. Future work involves further refining the ontology, assessing capabilities with more complex queries, and converting historical Examon datasets into RDF format for deployment in graph databases for further comparative analysis.

ACKNOWLEDGMENT

This research was partly supported by the EuroHPC EU Regale project (g.a. 956560), the HE EU Graph-Massivizer project (g.a. 101093202), and also the Spoke "FutureHPC & BigData" of the ICSC – Centro Nazionale di Ricerca in "High Performance Computing, Big Data and Quantum Computing", funded by European Union – NextGenerationEU. We also express our gratitude to CINECA for their collaboration and providing access to their machines.

REFERENCES

[1] Waqas Ali, Muhammad Saleem, Bin Yao, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. 2022. A survey of RDF stores & SPARQL engines for querying knowledge graphs. *The VLDB Journal* 31a, 3 (May 1 2022), 1–26. https://doi.org/10.1007/s00778-021-00711-3

[2] Andrea Bartolini, Francesco Beneventi, Andrea Borghesi, Daniele Cesarini, Antonio Libri, Luca Benini, and Carlo Cavazzoni. 2019. Paving the Way Toward Energy-Aware and Automated Datacentre. In *Proceedings of the 48th International Conference on Parallel Processing: Workshops* (Kyoto, Japan) *(ICPP 2019)*. Association for Computing Machinery, New York, NY, USA, Article 8, 8 pages. https://doi.org/10.1145/3339186.3339215

[3] Andrea Borghesi, Carmine Di Santi, Martin Molan, Mohsen Seyedkazemi Ardebili, Alessio Mauri, Massimiliano Guarrasi, Daniela Galetti, Mirko Cestari, Francesco Barchi, Luca Benini, Francesco Beneventi, and Andrea Bartolini. 2023. M100 ExaData: a data collection campaign on the CINECA's Marconi100 Tier-0 supercomputer. *Scientific Data* 10, 1 (18 May 2023), 288. https://doi.org/10.1038/s41597-023-02174-3

[4] Gabriel G. Castañé, Huanhuan Xiong, Dapeng Dong, and John P. Morrison. 2018. An ontology for heterogeneous resources management interoperability and HPC in the cloud. *Future Generation Computer Systems* 88 (2018), 373–384. https://doi.org/10.1016/j.future.2018.05.086

[5] Oscar Corcho, David Chaves-Fraga, Jhon Toledo, Julián Arenas-Guerrero, Carlos Badenes-Olmedo, Mingxue Wang, Hu Peng, Nicholas Burrett, José Mora, and Puchao Zhang. 2021. A High-Level Ontology Network for ICT Infrastructures. In *The Semantic Web – ISWC 2021*, Andreas Hotho, Eva Blomqvist, Stefan Dietze, Achille Fokoue, Ying Ding, Payam Barnaghi, Armin Haller, Mauro Dragoni, and Harith Alani (Eds.). Springer International Publishing, Cham, 446–462.

[6] Pouya Kousha, Vivekananda Sathu, Matthew Lieber, Hari Subramoni, and Dhabaleswar K. Panda. 2023. Democratizing HPC Access and Use with Knowledge Graphs. In *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis* (<conf-loc>, <city>Denver</city>, <state>CO</state>, <country>USA</country>, </conf-loc>) *(SC-W '23)*. Association for Computing Machinery, New York, NY, USA, 243–251. https://doi.org/10.1145/3624062.3624094

[7] Chunhua Liao, Pei-Hung Lin, Gaurav Verma, Tristan Vanderbruggen, Murali Emani, Zifan Nan, and Xipeng Shen. 2021. HPC Ontology: Towards a Unified Ontology for Managing Training Datasets and AI Models for High-Performance Computing. In *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*. 69–80. https://doi.org/10.1109/MLHPC54614.2021.00012

[8] Brian McBride. 2004. *The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS*. Springer Berlin Heidelberg, Berlin, Heidelberg, 51–65. https://doi.org/10.1007/978-3-540-24750-0_3

[9] Dejan Milojicic, Paolo Faraboschi, Nicolas Dube, and Duncan Roweth. 2021. Future of HPC: Diversifying Heterogeneity. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*. 276–281. https://doi.org/10.23919/DATE51398.2021.9474063

[10] Martin Molan, Junaid Ahmed Khan, Andrea Borghesi, and Andrea Bartolini. 2023. Graph Neural Networks for Anomaly Anticipation in HPC Systems. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 239–244.

[11] M. Ott, W. Shin, and et al. 2020. Global Experiences with HPC Operational Data Measurement, Collection and Analysis. In *2020 IEEE International Conference on Cluster Computing*.

[12] Bastian Quilitz and Ulf Leser. 2008. Querying Distributed RDF Data Sources with SPARQL. In *The Semantic Web: Research and Applications*, Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 524–538.

[13] Woong Shin, Vladyslav Oles, Ahmad Maroof Karimi, J. Austin Ellis, and Feiyi Wang. 2021. Revealing Power, Energy and Thermal Dynamics of a 200PF Pre-Exascale Supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 12, 14 pages. https://doi.org/10.1145/3458817.3476188

[14] Lauri Tuovinen and Jaakko Suutala. 2021. Ontology-based Framework for Integration of Time Series Data: Application in Predictive Analytics on Data Center Monitoring Metrics. 151–161. https://doi.org/10.5220/0010650300003064

[15] Wikipedia. 2021. CINECA — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=CINECA&oldid=954269846. [Online; accessed 04-December-2021].