International Journal of Mathematics and
Computer Science, **16**(2021), no. 1, 287–309

High-Performance Computing Based Approach for Improving Semantic-Based Federated Data Processing

Abdulelah A. Algosaibi

Department of Computer Science
College of Computer Science and Information Technology
King Faisal University
Al Ahsa, Kingdom of Saudi Arabia

email: aaalgosaibi@kfu.edu.sa

(Received May 2, 2020, Accepted July 25, 2020)

Abstract

Retrieving RDF datasets from distributed data sources has become an essential process to achieve the vision of the semantic web. The semantic web vision promotes a daily expanding of the semantic graph that requires some processing enhancements including SPARQL endpoint and federated queries to apprehend the endless expansion. The current advanced high-performance computing architecture has been developing rapidly to overcome many issues including performance. High-performance computing can potentially be employed to improve the federated SPARQL query and overall operation including the performance and processing. Thus, this work reviews the techniques to progress the development on separated and remote federated semantic-based data. That is, the high-performance computing environment including hardware architecture and special architecture computing has been surveyed. Moreover, operating semantic graph integration necessities is analyzed. Furthermore, the existing techniques to improve SPARQL performance are studied. Besides, an investigation about the utilization of advanced hardware computing architecture is reviewed in order to enhance the execution of the current SPARQL federation service operation.

Key words and phrases: Semantic Web, Linked Data, Linked Open Data, Parallelization, HPC, CUDA, SPARQL, Federation Query, SPARQL Query Optimization, Federated Data Management.
AMS (MOS) Subject Classifications: 68T09, 68T30, 94-02.
ISSN 1814-0432, 2021, http://ijmcs.future-in-tech.net

1 Introduction

The technique of collaborating and accessing information over the web has evolved resulting in the new era of the web called Semantic Web (SW) [1]. SW is a newer version of the web that supports the existing functionality of the existing one and aimed to make the web machine-understandable [2][3]. It will develop dedicated pieces of data together that allows the machine to understand the contents [4]; that is, computers can interoperate and reason on behalf of users. SW has emerged as a strategic technology for formally and semantically linking data together forming a mesh of knowledge and its main goal is to provide a unified could-based environment for data that holds the semantics in such a way that a machine can understand, process, and utilize. According to the W3C, semantic web technologies "enable people to create data stores on the Web, build vocabularies, and write rules for handling data" [5]. Linked data is defined as "a method of publishing structured data so that it can be inter-linked and become more useful through semantic queries, founded on HTTP, RDF and URIs" [6]. It aggregates some Semantic Web Technologies and provides a simple interface where an application can interact with these open data [7]. As reviewed by the inventor the Semantic Web, the recommendation of helpful Linked Data should be as follows: "(1) use URIs as names for things, (2) use HTTP URIs so that people can look up those names, (3) when someone looks up a URI, useful information is provided using the standards (RDF, SPARQL), and (4) include links to other URIs so that they can discover more things". Linked Open Data (LOD) cloud project shows the advanced level approached by the community. Each node in this cloud diagram represents a distinct data set published as Linked Data. As of March 2019, LOD collected 1,239 datasets with 16,147 links with many domains. For each one of these datasets, a minimum of 1000 triples are required at 50 connections to other datasets on the LOD cloud. It is worth mentioning that one dataset already contain 3,000,000,000 triples as will discussed later.

Computing techniques with high processing powers with a massive amount of memory are one of the main gains of High-Performance Computing (HPC) [8][9]. They utilize parallel computing to achieve many objectives. However, single-processor computers could not provide an effective processing capability that is required for the HPC applications [10]. Parallel computing [8][9][11][12] is a useful technique to enhance the computations efficiency for advanced tasks such as big data; that is, the design of algorithms, models, software adopts the multi-core architecture computers for better performance and scalability [9]. The Semantic Web promotes the expansion of a massive open semantic linked data. This vision must reside in the availability and accessibility to avoid outdated data processing. In order to overcome this issue and its consequences challenges, fast and scalable processing capabilities should be involved. In this research, the nature of the evolving LOD from the federated query point of view with respect to the HPC available environments is studied. The rest of the paper is organized as follows: Section 2 shows federated data management and an overview of the basis of Semantic Web concepts and technologies and it introduces the evolution of data over the semantic web era. In section 3, a spotlight is on the High-Performance Computing (HPC) techniques. In section 4, the semantic architecture is discussed together with a presentation of a case using advanced hardware computing. In section 5, the related literature is discussed. Finally, the conclusion of the paper is in section 6.

2 Federated Data Management

RDF stands for Resource Description Framework [13] that is a directed labeled graph making SPARQL is a semantic graph-matching RDF query language [14]. It is a World Wide Web Consortium (W3C) standard for describing resources over the web. Therefore, employ metadata describes data to deliver more information about elements. This metadata enables the automatic processing of web resources. RDF language can be seen as a data model for constructing the data on the web and relations existing in the world. RDF represents data in the arrangement of Subject-Predicate-Object as shown in figure 1. It is recognized as RDF triples $< S, P, O >$ that number of data together defines a massive collection of triples as an RDF graph. Thus, linking more than one RDF graph form a large number of semantically linked graphs. Unique Resource Identifier (URI) provides a designated

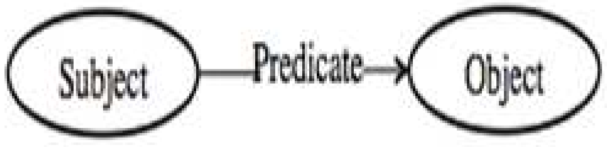

Figure 1: Data representation of RDF

location of the data description or literals (i.e., strings, date, numbers, etc.) or simply a blank node. RDF can efficiently connect the data to build a graph of connected data as the semantic web community currently uses it and practices [15][16]. It also facilitates data sharing and reusing across applications, enterprises, and community boundaries [17][18][19]. Recently, RDF and Linked data gained higher attention; i.e., Dbpedia. This exposed new advanced requirements to access, retrieve, and process distributed and dynamic semantically and coherently connected datasets with optimization methodologies for appropriate query processing. All of these triples require a special query language to be accessed and processed. This is happening via a standard semantic language called SPARQL [20]. The query language is used as an RDF query language. It works with separated and remote RDF datasets. In practice, SPARQL provides a graph pattern concept that allows users to match against a given RDF dataset. SPARQL which stands for SPARQL Protocol and RDF Query Language [17][20], is a standardized RDF graphs query language [20][21] developed by the World Wide Web Consortium [22]. SPARQL protocol systematically accesses diverse data sources [17] through their remote endpoints to generate lexical queries [16]. Since RDF is based on graphs, SPARQL uses graph pattern matching [17] by utilizing triples. The SPARQL query can be structured as the mandate PREFIX declaration to allow abbreviate the representation URIs, the proclaiming request and format, and the query graph pat-tern with some conditions in order to modify the results as summed in figure 2 snapped from [14]. Now,

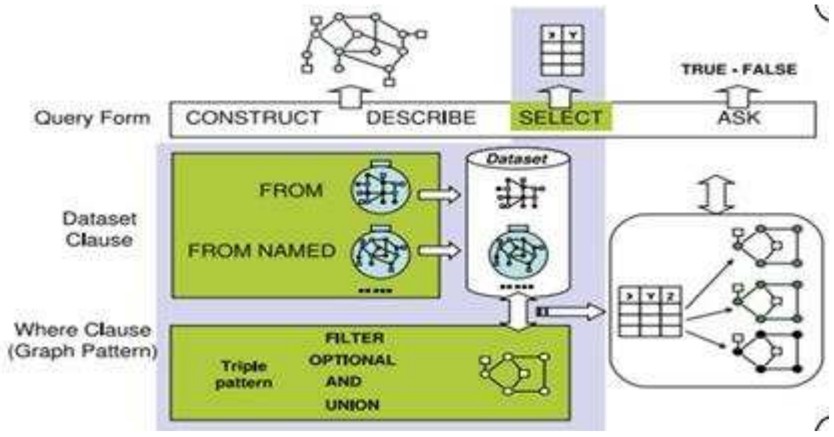

Figure 2: General form of SPARQL query

SPARQL 1.1 is the newer version that allows advanced features including "federated query" that allows queries from distributed SPARQL endpoints; that is, part of the query itself can be invoked against a given SPARQL endpoint. That is, in a federated query, each The SPARQL query is decomposed into sub-queries, and each of them is sent to given SPARQL endpoints to be executed. Once each request is completed, the results are combined and presented. SPARQL allows RDFS (RDF Schema) and Ontology querying and approaching triple stores and federated end-points making different location is processed at once. The federator is responsible for the intelligible sub-querying processing and grouping [23][24][25][26].

3 High-Performance Computing

The monolithic approach is usually followed in querying distributed semantic graph that is producing waiting time in a great mode. However, the nature of the distributed semantic graph environment supports multiple requests concurrently. Therefore, parallelism in this regard utilizes the power of computing in terms of processing a high number of requests in a short time. The process of designing a High Processing Computing (HPC) system requires an appropriate hardware architecture with the relevant parallel algorithm and mostly used for scientific computing, industry, health, defense, engineering, security, or national laboratories [27][28]. The organization, microconnectivity, scale of the resources, and availability of the robust software to handle operation at that given scale are what make HPC id different from typical computers [27]. The processes such as distribution, calculation, and final collection of application data are considered in the designing phase to increase the efficiency of a system. In that sense, massive parallel processing is required for raising computing processing performance. The concept of multicore computers is becoming popular among software designers [8][9][10][11][12]. Three common types of parallelization models: shared memory parallelization [29], a concept that has a shared memory for all processor and distributed memory parallelization [12][29], a concept that has dedicated memory for each processor as described below, and Special Architecture as discussed later.

3.1 Shared Memory Parallelization

The idea of shared memory parallelism is dependent on shared memory to provide simple communication between the different processing nodes. Each processor will perform a dedicated task. A shared variable can be accessed by processors according to the requirements. The different variations of shared memory are Uniform Memory Access (UMA), Non-uniform Memory Access

(NUMA), and Cash-Only Memory Architecture (COMA). Threads are a lightweight process that requires a small amount of development time and fewer efforts to be scheduled by the operating systems. A process can create more than one thread that runs at the same address space as the parent process. Multi-core architecture is a single computing unit that contains some processing unit (called core) at the same chip that may share the memory and run the instructions in parallel [29].

3.2 Distributed Memory Parallelization

Distributed memory parallelization is a unique approach that uses a dedicated memory for each processor [30][31]. It does not have a common memory for processors. It employs a message interface for processors to establish communication between them. Therefore, each processor will have a unique ID in the communication group. When a processor x wants to communicate with a processor y, it sends a direct message to y using its address. The Message Passing Interface (MPI) is a well known specification of a parallel library for establishing communications among processors. It will provide a collection of APIs that can be called by the application code to communicate with other processors. The specification of MPI will provide two types of calls: point to point and collective calls.

3.3 Special Architecture

Special Architecture either utilizes parallel computing or employs parallel computing by an innovative approach. In clusters [32], a set of computers are connected together to perform a certain task. In some architecture, all CPUs are on the same board and share the same system bus. In another architecture, a CPU can have its board and peripheral devices. A high-speed network connection is required to join these differently. Scalability is one of the main advantages here; that is, new nodes can be added to the cluster smoothly to increase the cluster system computational power [33]. Clusters may utilize a distributed memory architecture where MPI carries out the communications among cluster nodes. However, if the cluster uses distributed memory architecture, then the connection medium among cluster nodes is exposed to high communication overhead. Alternatively, clusters may utilize shared memory architectures or a hybrid model architecture where some nodes share memory (multi-code processors) while the individual nodes use distributed memory architecture. In grids computing

[34][35][36], geographically separate computational resources are utilized to solve a computational problem in a heterogeneous environment manner that may not necessarily have the same hardware/software architectures; that is, each task is processed in totally different machines allowing overall application parallelism. Through message passing and a special program, these remotely assigned machines can be communicating and managing the running tasks on the same grid. Grid computing requires high task scheduling supervision such that resources on the grid be entirely utilized. Cloud computing [37][38][34] is an emerging computing model that adopts the distributed and virtual use of resources promoting cost reduction and ease of use. In the cloud computing model, the Cloud Service Provider (CSP) offers certain services to individual users and business companies. There are three layers in the cloud framework that represents a layer of computation. These layers are Software-as-a-Service (SaaS) layer, Platform-as-a-Service (PaaS) layer, and Infrastructure-as-a-Service (IaaS) layer. One major difference here among cloud, grid, and cluster computing is that cloud computing does not only offer a model of computation. However, it can provide other services; i.e., storage, special resource, or computational pool renting. Fog computing is an extended version of cloud computing that enlarges and simplifies the concept of cloud computing to overcome some cloud computing limitations; i.e., latency and encourages miniature activities including storage, communicating among devices and data centers [39][40]. Jungle computing is also special architecture that advanced the concept of distributed computing via simultaneous consolidation of different architectures in order to reduce the programming complexity and streamline computational tasks among clusters to achieve peak performance [41][42]. The race of supercomputer development is an active field due to reasons mentioned above. According to [43][44], table 1 shows the top ten fastest high-performance computers in the world as for November 2019. Rmax and Rpeak are scores used to rank supercomputers dependent on their performance utilizing the LINPACK Benchmark. In the table below, Rmax stands for Maximal LINPACK performance achieved, Rpeak stands for Theoretical peak performance, and GFlops stands for Giga (billion) Floating Point Operations Per Second.

Table 1: Top 10 super computers based on the number of cores

No	Vendors	Rmax(GFlops)	Rpeak(GFlops)	Cores
1	Lenovo	330,546,526	610,258,957	12,339,312
2	NRCPC	93,014,594	125,435,904	10,649,600
3	Cray or HPE	162,476,312	249,470,362	6,029,800
4	IBM	216,918,859	287,901,962	6,005,272
5	NUDT	66,081,890	108,454,198	5,34,848
6	Sugon	119,286,000	301,609,541	5,182,664
7	HPE	122,353,488	179,809,284	3,174,784
8	Inspur	102,120,870	219,011,562	2,666,744
9	Atos	64,827,910	103,336,972	2,117,592
10	IBM or NVIDIA or Mellanox	114,129,000	150,445,771	1,880,424

4 The Semantic Graph Requirements and Leveraging

The vision of the semantic web is to allow data published in an organized manner in such a way that they interwoven and serve the semantic queries. Thus, it requires datasets to be accessed remotely and processed the integration of these data; that is, semantic integration allows alignment of the varied statements from triplestores and processes in a federated platform. Processing datasets offline is fine. However, it is against the advancement of realizing the vision of the semantic web. As stated in [45] "Integration requires spending resources on mapping heterogeneous data items, resolving conflicts, cleaning the data, and so on. Such costs can also be huge. The cost of integrating some sources may not be worthwhile if the gain is limited, especially in the presence of redundant data and low-quality data." The lack of proper processing of linked datasets on the cloud such as searching or summarizing was leading to problems in accessing and requesting these data. For these datasets to be processed efficiently, integration is crucial; that is, the challenge is in joining the required datasets that are distributed in different places use diverse URIs and models when representing their entities and schema elements. The integration of records from independent and distributed datasets is required to implements a set of SPARQL requests. SPARQL query optimization handles the varieties of joins methods to integrate RDF datasets including bind join, nested loop join, hash join, symmetric join, multiple hash join, subquery building, join method selection, and join ordering. Integration substance (or integration architecture) can either be materialized or virtual [7]. RDF graph datasets should match LOD principles for leveraging the Semantic Web vision; that is, downloading the required set of RDF graphs to be locally processed would allow for limited data validity and contradict the vision of the semantic web. In "sub-clouds" of LOD, domains are labeled to support the notation of ontologies. Figure 3 shows the geography domain from the LOD cloud. One of the RDF

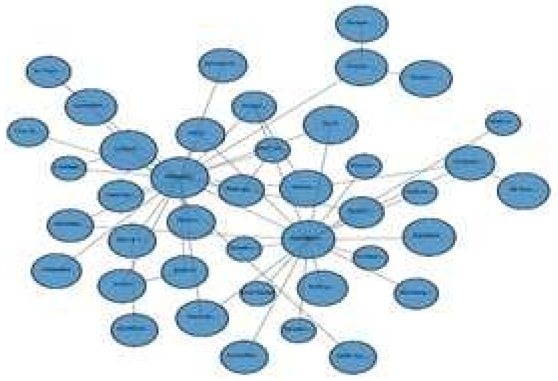

Figure 3: Geography sub-cloud

datasets from geography sub-cloud is Yahoo! GeoPlanet that provides intelligent infrastructure for geo-referencing data. The total size of this data set only is 49,734,022 triples. Another is LinkedGeoData that utilizes OpenStreetMap projects data with total size 3,000,000,000 triples. Concerning the ability to inquiry remote endpoints through the SERVICE clause, these datasets can be contacted simultaneously to convey requests. The federated SPARQL query can be sent to multiple dataset storages as Linked Open Data as shown in figure 4 by [46]. The monolithic state of computing was slow and had performance issues; i.e., reduction of speed [9]. The reason was the execution of a task has completed absolutely before another task. All the tasks should be arranged in a queue and executed in order. Therefore, if there are numerous tasks to be done, say thousands, a greater time would be required for their completion [47]. HPC aimed for increasing the performance and capabilities. A number of developments have taken place to speed up the hardware processing. Recalling the special architecture computing and the great computing power environments available at supercom-

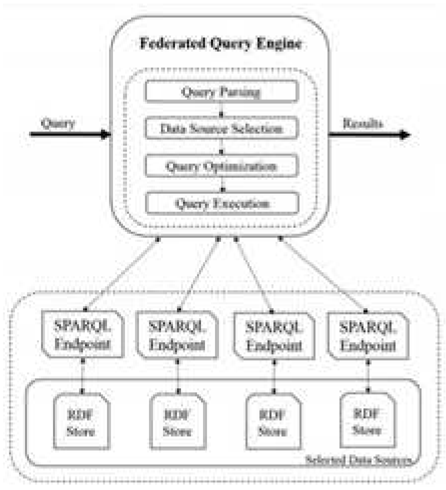

Figure 4: SPARQL Federated query processing

puter centers, proper adoption of joint methods would result in improving overall semantic web performance operations including federated SPARQL query processing. Some computing architectures are available as Massage Passing Interface (MPI) that supports cluster nodes with their individuals RAM [48]. OpenMP is another option that upkeep many platforms shared memory multiprocessing [27]. Also, there is OpenCL (stands for Open Computing Language) that provides great capabilities in task-scheduling on GPU [49]. In this case, proper query order is vital here to speed up the processing and shorten the waiting time. Moreover, utilizing the power of parallel computing advances the trend of the semantic evolution. HPC literature introduced some ways of solving tasks concurrently to greatly reduce the amount of execution time hence increasing the performance in federation services or mechanisms fitted with the federation services. The implementation of shared memory architecture allows writing sequential programs that share the same memory among all computing nodes. Special care should be taken during the access of shared memory by different nodes. Moreover, shared memory architecture will not impose any computational overhead as communication among nodes takes place through the shared memory. On the contrary, distributed memory architecture does not assume that worker nodes share any kind of memory between them. Therefore, processors can take advantage of passing messages to communicate with each other, which may introduce communication overhead due to the frequent communication between the worker nodes. One of the famous massive parallel architecture is called Compute Unified Device Architecture (CUDA) that offers wrappers for advanced programming languages such as Java and Python. Thus, this work is considered as an example of implementing great integrated SPARQL querying performance processing.

4.1 Compute Unified Device Architecture (CUDA)

In this section, a technical discussion is presented in order to analyze one of the well known parallel computing architecture (Compute Unified Device Architecture CUDA). This discussion prorates the availability of this architecture. It offers wrappers for advanced programming languages such as Java and Python that supports SPARQL query processing. CUDA is implemented in the Graphics Processing Units (GPU). Generally speaking, GPU has many advantages including low cost, simple, raw computational power, and great memory bandwidth [50]. It utilizes the application of parallel computing technology or capability. This technology enhances the performance of the computer game industry by improving the quality of graphics of a game and physics calculation. The foremost advantages of CUDA are the fast execution time and accurate mathematical calculations that result in the production of realistic gaming and generation of very high-quality images. The NVidias CUDA technology supports scalable parallel programs with enabling a wide variety of applications such as sparse matrix, search engines, models in physics, and computations in chemistry among others. The architecture is shown in figure 5 from [51]. CUDA uses the same kernels, which are parallel, to recent General Purpose Graphical Processing Unit (GPGPU) models. The difference between the two models is that CUDA provides flexible threads creation, shared and global memory, thread blocks, and better synchronization [52]. In CUDA architecture, there are two parts: the host and the device. The program is running into a host program in CPU. That is, a minimum of one parallel kernels for the device execution on GPU. A clear arrangement among threads is needed. Thread is a basic unit for the data manipulation inside CUDA. The thread indexed as a multidimensional vector (one-dimensional, two-dimensional, or three-dimensional thread index). All the threads access concurrently blocks shared memory. Each block can be identified using a one-dimensional or two- dimensional index and all grids will share a global memory [52][53]. In CUDA, shared memory is on a chip that increases efficiency when compared to local-global memory (latency is 100 times lesser than the global shared memory) [54][50] Here, threads block have access to shared memory. The threads must be synchronized so that correct results are provided during parallel threads cooperation. To endorse

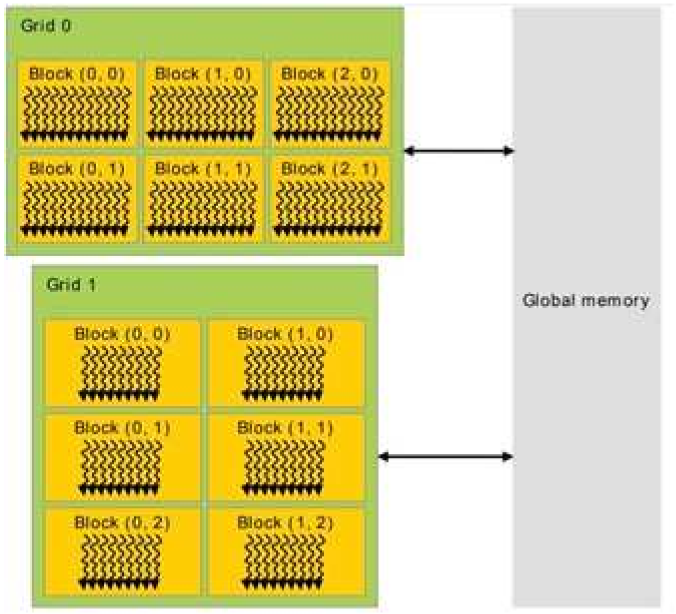

Figure 5: The architecture of CUDA

the presentation of this approach, the parallel algorithm on the graphical processing units (GPUs) with architecture similar to NVidias Compute Unified Device Architecture (CUDA) would assist in transferring a wide variety of data in less amount of time. It employs the parallel execution of data to execute multiple tasks at the same time to increase the processing speed. That is, with further tuning it would accommodate the nature of the distributed semantic graph. In the query engine, an abstracted layer of manual configuration would be needed. In this case here, some wrappers for Java, Python, and other programming languages are available. It will save time and support numerous applications on a single computer. It is an essential technique for an ever-evolving world [54]. Consequently, each SPARQL query federation engine can configure a local ad hoc federation operation rather than loading data that hold back the notation of the semantic web. The momentous growth of modern semantic-based technology and innovation will be anticipated in the future. The fresh areas will comprise the storage of the data processed from the LOD and the demonstration of the intelligence with human interface, as well as linguistic and ontological data.

5 Literature Review

With the increasing quantity of RDF datasets at the LOD cloud, there is a need for efficient performance to retrieve diverse data from variant and large sources. Accordingly, this section restricts the discussion to study the giant semantic graph query processing and optimization with respect to a high-performance computing point of view. The literature shows that some research have been conducted to improve the performance of SPARQL queries in the literature including federated queries [46][55][56][57][58][59][60][61][15]. Query optimization has always been critical as it concerns a humongous web of linked data of differing structural complexity from dissimilar multiple data sources [15][21][62][16][17] even in the earliest types of databases; and though this generation has already amassed several related works of literature to understand and solve it [16][22][17], researches keep pouring in to apply different strategies that promise optimum performance. In databases, query optimization allows to compares some query approaches and indicates one of them that satisfies the least expected cost of processing [63]. Some pieces of literature mentioned above discussed decomposing the original query and reordering the fragments to shorten execution time. It is worth mentioning that aggregating collected fragments from these voluminous resources have led to expensive query performance in terms of resources, communication, and time [15] [20][8]. Arising issues include distributed processing of SPARQL queries [15], comparing classes of queries, data partitioning, and data replication [15]. Lately, Other concerns have been raised like the focus on in-memory models rather than on-disk [17], the application to different graph shapes [17], and the emphasis on distributed join operations [15]. Some of them are either statistics-based [62][15][16] or heuristics-based [22][17][18][19][21]. Among the latter, two are prediction-based [19][21] which have a lot of interest among researchers. The statistics-based approach means that pre-computed statistics resulted from several SPARQL query executions on an RDF document [15][19]. On the other hand, heuristics-based recognize a heuristic definition after a series of investigation and applies it [22][18]. The former is more expensive to implement and maintain compared to the latter in terms of time and resources. However, it may be more efficient for some specific data sets [62][16].

In [22] Hartig configured Jena Framework to resolve some challenges in the performance of the queries by rearranging the query and putting the more specific part first proved the importance of decomposing and reordering its parts. OptARQ [62] is a prototype for optimizing queries in SPARQL. They used static optimization which are general rules for executing a query execution plan (QEP) where a query is rewritten after query parsing and syntax checking. By using statistics in the selectivity of triple or SEL(T), they tried to order query patterns to minimize transitional result sets for each level of execution by estimating the cost of subject, predicate, and object. Its fundamental approach works strongly in federated SPARQL queries. The work in [26] has developed a semantic facilitation approach to process multiple heterogeneous sources. Federation environments infrastructures were reviewed with an assessment of available SPARQL federation in [64]. In [20], an analysis of federation over SPARQL Endpoint was developed to facilitate the federation framework. In [65], the authors argued that the number of sources affects the query runtime. In [66] The authors have presented some joins queries to perform parallel SPARQL processing. In [15], Li, et. al. proposed a statistical model in the optimization of SPARQL queries. They created a Web of Linked Classes from a Web of Linked Data that is comparably smaller and made source selection faster. SemWIQ [16] primarily used RDF statistics to support SPARQL query federation by distributing a query to multiple SPARQL endpoints. Here, all subjects in triples were designated as variables and evenly considered the addition of Description Logic (DL) constraints. SQGM [22] covers all the phases of query processing and used transformation rules combination to rewrite rules as the base for heuristics to enable the Jena-based query to utilize a fast path algorithm. It first translates a SPARQL query into a SPARQL Query Graph Model before rewriting the query for faster execution. Although the proposal is practical, it does not optimally rewrite the queries. In [47], the authors claim federated repositories are necessary to retrieve data across the web. They claimed that data federation is in most cases unmanageable due to the technical difficulties that counter the process of addressing multiple data sources at a go. The paper has described a data federation structure from a semantic mediation layer basis and an advanced query engine with the ability to interface to various heterogeneous data sources. The performance and usability signals are also exhibited in the paper in a bid to demonstrate the applicability of the method and its feasibility. Ant Colony Optimization (ACO) in SPARQL queries [17] reorders the triple patterns by first determining a weight matrix and process it using ACO algorithms such as Ant System, Elitist Ant System, and MAX-MIN Ant System. Their experiment included other algorithms for comparison on four query shapes: chain, star, cyclic, and chain star. Though their focus on in-memory models of RDF data proved faster performance compared to on-disk, its limitation on scaling remains an issue. Wu et. al.[18] proposed a parallel RDF engine that can select from available RDF data partitioning methods. They used a traditional approach in partitioning SPARQL queries into subqueries, introduced two heuristic-based approaches to delineate the sources, and another algorithm to automatically apply the suitable heuristic (among several possibilities) according to the structure of the query graph. Hasan [19] and Torre-Bastida. et. al. [21] employed a prediction-based approach in query optimization. The work by Hasan [19] introduced a Machine Learning (ML) approach to optimize SPARQL queries by comparing it to previously executed queries. They treated SPARQL queries as vectors and applied KNN and SVM machine learning algorithms. On the other hand, Torre-Bastida et. al. [21] balanced two query optimization goals on runtime speed and quality of queries by introducing a queuing system when SPARQL queries are run and an intelligent query relaxation technique. The research [67] studied query optimization methods (i.e. semijoins and dynamic programming). Also, some other work as in [68] had inspected the query optimization in substructures in the several SPARQL queries. In [69] the Adaptive Query Processing represents a new direction in improving query performance involving data integration and data streams. The classes of adaptation were grouped between adaptive selection ordering through A-Greedy technique and adaptive join processing through history-independent pipelined execution, history-dependent pipelined execution, and non-pipelined execution. Some problems brought forth include parametric query optimization, parallelism, routing/adaptation policies, and larger-than-memory execution. The research [70] questions the performance and scalability of using selectivity estimation of SPARQL queries in Basic Graph Pattern (BGP) and heuristics using the Lehigh University Benchmark (LUBM). It found that the selected ordering of the stored triple patterns of BGP have grave impact on the characteristics of heuristics, implying that queries can either be high-selective or low-selective. The research [56] highlights the importance of data integration and interoperability in accessing Linked Open Data. The authors suggested hybrid data catalogue and link predicate awareness. The authors of research [46] present a discussion on data source selection, join, and query optimization methods including parallelism in order to improve the performance. The work [61], proposed an approach in which a parallel model shows an improved distributed SPARQL processing. The research [71] set up queries into parallel code to request graph methods for the construction of answers. Research [60] presented an approach for querying distributed Linked Data sources using SPARQL 1.0 that incorporate a parallelization infrastructure to execute subqueries at dif-

ferent endpoints, concurrently. The research in [72] proposed adaptive query engine that acclimates linked data SPARQL queries implementation scheduler.
As per this work survived, most of the work related to source selection, join methods, and query optimization methods of existing query federation engines leaving the utilization of high-performance computing capabilities, however, still not achieving the desired performance. The literature shows that the full utilization of high-performance computing is still not noticeable in the literature. The vast majority of the work concerned with improving SPARQL at the software level. Few works of literature inspect or configure the SPARQL query processing and optimization that properly utilize high-performance computing including special architecture high-performance computing. Besides, due to the complexity of handling distributed connections and processing the results, the semantic graph management tools are not providing great low-level handling of the federated queries. That is, most of the modern architecture of the semantic frameworks and other data repositories are yet to adapt and utilize the concept of high-performance computing. And, the evolving of the semantic graph tasks including SPARQL querying requires employing HPC to improve the overall query federation processing.

6 Conclusion

This study has provided an overview of an approach to advance the processing of federated data management from various research papers. The emerging technologies of special HPC architecture would utilize the features of advanced processing to provide service to end-users or to combine processing nodes from different locations and provide a uniform and coherence computational powers to end-users. In this work, the well-known HPC architectures have been summarized in order to share with the community the possible intersection between fields to advance semantic web tasks such as browsing, summarizing, querying, searching, organizing, etc. Also, a spotlight to the field is introduced for further deep configuration in order to utilize HPC with the evolution of semantic graphs. To end this, query optimization of query federation engines have been studied to improve the performance. However, configuring and employing high-performance computing architecture to improve overall query federation processing is still an open challenge. Therefore, one goal of this paper is to provide inspiration to incorporate that which would allow increasing the semantic web applications.

It is worth mentioning that the combination of two models of parallel processing techniques can produce a hybrid model where some components of the parallel system use shared memory while other components will employ a distributed approach. The hybrid model will combine both the Central Processing Unit (CPUs) and GPUs to solve a problem. In this case, CPUs may utilize distributed memory architecture and GPUs will employ a shared memory model. In some scenarios, one cannot easily decide the type of HPC architecture to implement the same application. In other words, the application has to process a huge number of data at the same time. There is a need to transfer an amount of data among the processing nodes. HPC architecture can be a key factor for the generation of better results. Therefore, the decision to deploy certain applications using a parallel approach should be application-dependent. The parallel processing technique operates in a defined environment. That is, manipulating the processing of execution will be faster and produce accurate results. The query federation engines depend on: no preprocessing per query and unbound predicate queries. That is, the selection of data sources using a metadata catalog may develop some performance issues if no proper configuration on the HPC environment. In this regard, query optimization with appropriate HPC design architecture would permit greater capabilities in managing semantic federated data including improving the performance. The general principles of such models are including uniform semantic querying of the distribution and heterogeneity. Moreover, it has investigated the methods for parallel distribution of data to transfer several forms of data at the same time. The challenges of query optimization are anticipated to increase with trending semantic graph evolution. Semantics web technologies are expanding that to take account of individuals to share data in applications or websites. Thus, the special architecture of high-performance computing with proper query optimization techniques would reduce the cost of processing and increase the performance.

References

[1] T. Berners-Lee, J. Hendler, O. Lassila, "The semantic web," Scientific American, 2001.

[2] G. Antoniou, P. Groth, F. van Harmelen, R. Hoekstra, A Semantic Web Primer, Third ed., MIT Press, 2012.

[3] C. Weiss, P. Karras, A. Bernstein, "Hexastore: Sextuple indexing for semantic web data management," Proc. VLDB Endow., 2008.

[4] S. Schlobach, C. A. Knoblock, "Dealing with the messiness of the web of data," Journal of Web Semantics, 2012.

[5] Semantic Web–W3C. [Online]. Available: https://www.w3.org/standards/semanticweb/ [Accessed: 28-Mar-2020].

[6] C. Bizer, T. Heath, T. Berners-Lee, "Linked data - The story so far," Int. J. Semant. Web Inf. Syst., 2009.

[7] M. Mountantonakis, Y. Tzitzikas, "Large-scale semantic integration of linked data: A survey," ACM Comput. Surv., 2019.

[8] M. J. Flynn, Some computer organizations and their effectiveness," IEEE Trans. Comput., 1972.

[9] "Computer architecture: pipelined and parallel processor design," Choice Rev. Online, 1995.

[10] F. Nielsen, Introduction to HPC with MPI for Data Science, 2016.

[11] C. Chen, H. Pouransari, S. Rajamanickam, E. G. Boman, E. Darve, "A distributed-memory hierarchical solver for general sparse linear systems," Parallel Comput.,(2018).

[12] V. Cholvi, E. Jiménez, A. Fernàndez Anta, "Interconnection of distributed memory models," J. Parallel Distrib. Comput., **69,** no. 3, (2009), 295–306.

[13] G. Schreiber, Y. Raimond, "Rdf 1.1 Primer," W3C Working Group Note 2, 2014.

[14] M. Arenas, C. Gutierrez, J. Pèrez, "On the Semantics of SPARQL," in Semantic Web Information Management, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, 281-307.

[15] X. Li, Z. Niu, C. Zhang, "Towards efficient distributed SPARQL queries on linked data," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2014.

[16] A. Langegger, W. Wöß, M. Blöchl, "A semantic web middleware for virtual data integration on the web," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2008.

[17] E. G. Kalayci, T. E. Kalayci, D. Birant, "An ant colony optimisation approach for optimising SPARQL queries by reordering triple patterns," Inf. Syst., (2015).

[18] B. Wu, Y. Zhou, H. Jin, A. Deshpande, "Parallel SPARQL query optimization," in Proceedings of the International Conference on Data Engineering, 2017.

[19] R. Hasan, F. Gandon, "A machine learning approach to SPARQL query performance prediction," in Proceedings of the 2014 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Workshops, WI-IAT 2014, 2014.

[20] W. S. W. Group, "SPARQL 1.1 Overview," 2013.

[21] A. I. Torre-Bastida, E. Villar-Rodriguez, M. N. Bilbao, J. Del Ser, "Intelligent SPARQL endpoints: Optimizing execution performance by automatic query relaxation and queue scheduling," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2016.

[22] O. Hartig, R. Heese, "The SPARQL query graph model for query optimization," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2007.

[23] O. Hartig, "Zero-knowledge query planning for an iterator implementation of link traversal based query execution," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2011.

[24] O. Hartig, "SQUIN: A Traversal Based Query Execution System for the Web of Linked Data," in Proceedings of the 2013 international conference on Management of data - SIGMOD 13, 2013, p. 1081.

[25] V. Fionda, C. Gutierrez, G. Pirrò, "Semantic navigation on the web of data: Specification of routes, web fragments and actions," in WWW12 - Proceedings of the 21st Annual Conference on World Wide Web, 2012.

[26] A. Gaignard, J. Montagnat, C. F. Zucker, O. Corby, "Semantic Federation of Distributed Neurodata," 2012.

[27] T. Sterling, M. Anderson, M. Brodowicz, "High performance computing modern systems and practices," 2017.

[28] A. Geist, D. A. Reed, "A survey of high-performance computing scaling challenges," International Journal of High Performance Computing Applications, (2017).

[29] G. Barlas, "Multi core and GPU programming integrated approach" 2014.

[30] J. Protic, M. Tomasevic, V. Milutinovic, "Distributed shared memory: concepts and systems," IEEE Parallel Distrib. Technol., 1996.

[31] F. Ozgruner, F. Ercal, "North Atlantic Treaty Organization, Scientific Affairs Division, and NATO Advanced Study Institute on Parallel Computing on Distributed Memory", 1993.

[32] N. Sadashiv, S. M. D. Kumar, "Cluster, grid and cloud computing: A detailed comparison", 2011, 477–482.

[33] K. Kaur, A. K. Rai, "A Comparative Analysis: Grid, Cluster and Cloud Computing," Int. J. Adv. Res. Comput. Commun. Eng., (2014).

[34] M. Mariotti, O. Gervasi, F. Vella, A. Cuzzocrea, A. Costantini, "Strategies and systems towards grids and clouds integration: A DBMS-based solution," Futur. Gener. Comput. Syst., **88,** (2018), 718–729.

[35] I. Merelli, "Infrastructure for High-Performance Computing: Grids and Grid Computing," in Encyclopedia of Bioinformatics and Computational Biology, Elsevier, 2019, 230–235.

[36] U. Schwiegelshohn et al., "Perspectives on grid computing," Futur. Gener. Comput. Syst., **26,** no. 8, (2010), 1104–1115.

[37] W. Lin, S. Xu, L. He, J. Li, "Multi-resource scheduling and power simulation for cloud computing," Inf. Sci., (2017).

[38] L. Coppolino, S. DAntonio, G. Mazzeo, L. Romano, "Cloud security: Emerging threats and current solutions," Comput. Electr. Eng., (2017).

[39] S. Mostafavi, W. Shafik, "Fog Computing Architectures, Privacy and Security Solutions," J. Commun. Technol. Electron. Comput. Sci., **24,** no. 0, (2019), 1–14.

[40] H. Madsen, B. Burtschy, G. Albeanu, F. Popentiu-Vladicescu, Reliability in the utility computing era: Towards reliable Fog computing," in 20th International Conference on Systems, Signals and Image Processing, (2013), 43–46.

[41] M. Hajibaba, S. Gorgin, "A Review on Modern Distributed Computing Paradigms: Cloud Computing, Jungle Computing and Fog Computing," J. Comput. Inf. Tech., **22,** no. 2, (2014), 69–84.

[42] J. Zarrin, R. L. Aguiar, J. P. Barraca, "HARD: Hybrid Adaptive Resource Discovery for Jungle Computing," J. Netw. Comput. Appl., **90,** (2017), 42–73.

[43] Y. Robert et al., TOP500," in Encyclopedia of Parallel Computing, Boston, MA: Springer, 2011, 2055–2057.

[44] "Home–TOP500 Supercomputer Sites," The most powerful high-performance computers from the market. [Online]. Available: https://www.top500.org/ [Accessed: 18-May-2020].

[45] X. L. Dong, B. Saha, D. Srivastava, "Less is more: Selecting sources wisely for integration," Proc. VLDB Endow., (2012).

[46] D. Bednarek, J. Dokulil, J. Yaghob, F. Zavoral, "Using Methods of Parallel Semi-structured Data Processing for Semantic Web," in Third International Conference on Advances in Semantic Processing, 2009, 44–49.

[47] D. Kossmann, "The state of the art in distributed query processing," ACM Comput. Surv., (2000).

[48] "Using MPI: Portable parallel programming with the message-passing interface," Comput. Math. with Appl., (2000).

[49] A. Munshi, OpenCL programming guide, Addison-Wesley, 2011.

[50] M. Ujaldòn, "CUDA Achievements and GPU Challenges Ahead," Springer, Cham, 2016, 207–217.

[51] NVIDIA, CUDA C++ Programming Guide," Program. Guid., (2019), 1–261.

[52] Y. Feng, L. Zhao, J. Yang, "Tuning Schema Matching Systems using Parallel Genetic Algorithms on GPU," Int. J. Mod. Educ. Comput. Sci., (2010).

[53] R. S. Sinha, S. Singh, V. K. Banga, "Accelerating Genetic Algorithm Using General Purpose GPU and CUDA," Int. J. Comput. Graph, **7,** no. 1, (2016), 17–30.

[54] J. Nickolls, I. Buck, M. Garland, K. Skadron, "Scalable parallel programming with CUDA," Queue, 2008.

[55] B. Quilitz, U. Leser, "Querying distributed RDF data sources with SPARQL," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2008.

[56] J. Feng, C. Meng, J. Song, X. Zhang, Z. Feng, L. Zou, "SPARQL Query Parallel Processing: A Survey," in Proceedings of the IEEE 6th International Congress on Big Data, BigData Congress, 2017.

[57] N. A. Rakhmawati, J. Umbrich, M. Karnstedt, A. Hasnain, M. Hausenblas, "A comparison of federation over SPARQL endpoints frameworks," in Communications in Computer and Information Science, 2013.

[58] M. Saleem, A. C. Ngonga Ngomo, "HiBISCuS: Hypergraph-based source selection for SPARQL endpoint federation," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2014.

[59] O. Görlitz, S. Staab, "SPLENDID: SPARQL endpoint federation exploiting voiD descriptions," in CEUR Workshop Proceedings, 2011.

[60] A. Schwarte, P. Haase, K. Hose, R. Schenkel, M. Schmidt, "FedX: Optimization techniques for federated query processing on linked data," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2011.

[61] X. Wang, T. Tiropanis, H. C. Davis, "LHD: Optimising linked data query processing using parallelisation," in CEUR Workshop Proceedings, 2013.

[62] A. Bernstein, C. Kiefer, M. Stocker, "OptARQ: A SPARQL Optimization Approach based on Triple Pattern Selectivity Estimation," Univ. Zurich, Dep., 2007.

[63] T. Neumann, "Query Optimization," in Encyclopedia of Database Systems, Springer New York, (2018), 3009–3015.

[64] N. A. Rakhmawati, J. Umbrich, M. Karnstedt, A. Hasnain, M. Hausenblas, "Querying over Federated SPARQL Endpoints —A State of the Art Survey," 2013.

[65] M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, A. C. N. Ngomo, "A fine-grained evaluation of SPARQL endpoint federation systems," in Semantic Web, 2016.

[66] T. Wang, P. Yuan, X. Liao, H. Jin, "Parallel Processing SPARQL Theta Join on Large Scale RDF Graphs," in IEEE Global Communications Conference, GLOBECOM 2018 - Proceedings, 2018.

[67] O. Görlitz, S. Staab, "Federated data management and query optimization for linked open data," Stud. Comput. Intell., 2011.

[68] R. Gomathi, "Efficient Optimization of Multiple SPARQL Queries," IOSR J. Comput. Eng., **8,** no. 6, (2013), 97–101.

[69] A. Deshpande, J. M. Hellerstein, V. Raman, "Adaptive query processing: why, how, when, what next," in Proceedings of the ACM SIGMOD international conference on Management of data, 2006, 806.

[70] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, D. Reynolds, "SPARQL basic graph pattern optimization using selectivity estimation," in Proceeding of the 17th International Conference on World Wide Web, 2008.

[71] V. G. Castellana, A. Tumeo, O. Villa, D. Haglin, J. Feo, "Composing data parallel code for a SPARQL graph engine," in Proceedings - SocialCom/PASSAT/BigData/EconCom/BioMedCom, 2013.

[72] M. Acosta, M. E. Vidal, T. Lampo, J. Castillo, E. Ruckhaus, "ANAPSID: An adaptive query processing engine for SPARQL endpoints," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2011.