A Demonstration of Lusail – Querying Linked Data at Scale

Essam Mansour[§] Ibrahim Abdelaziz[‡] Mourad Ouzzani[§]
Ashraf Aboulnaga[§] Panos Kalnis[‡]
[§] Qatar Computing Research Institute, HBKU, [‡] King Abdullah University of Science & Technology
{emansour, mouzzani, aaboulnaga}@hbku.edu.qa, {ibrahim.abdelaziz, panos.kalnis}@kaust.edu.sa

ABSTRACT

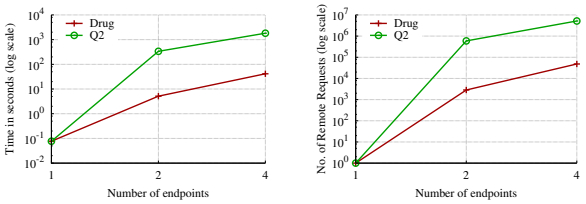
There has been a proliferation of datasets available as interlinked RDF data accessible through SPARQL endpoints. This has led to the emergence of various applications in life science, distributed social networks, and Internet of Things that need to integrate data from multiple endpoints.

We will demonstrate Lusail; a system that supports the need of emerging applications to access tens to hundreds of geo-distributed datasets. Lusail is a geo-distributed graph engine for querying linked RDF data. Lusail delivers outstanding performance using (*i*) a novel locality-aware query decomposition technique that minimizes the intermediate data to be accessed by the subqueries, and (*ii*) selectivity-awareness and parallel query execution to reduce network latency and to increase parallelism. During the demo, the audience will be able to query actually deployed RDF endpoints as well as large synthetic and real benchmarks that we have deployed in the public cloud. The demo will also show that Lusail outperforms state-of-the-art systems by orders of magnitude in terms of scalability and response time.

1. INTRODUCTION

Linked Data has led to new ways to publish, interlink and re-use information from remote Web data sources. There are now many publicly accessible interlinked datasets in different domains, such as media, government, and life sciences [7]. Each dataset is stored as RDF triples and is accessible via a SPARQL endpoint, which is a service published at a URI receiving SPARQL queries via HTTP requests. According to stats.lod2.eu, linked RDF data consist of more than 85 billions triples over more than 3400 datasets.

The availability of such a large amount of data has led to the emergence of several applications that need to query linked RDF data at scale, i.e., accessing tens to hundreds of geo-distributed linked RDF datasets. For example, in life sciences, Bio2RDF[1] has about 11 billions triples across 35 datasets, and the Internet of Things will connect billions of decentralized datasets [1] that will need to be accessed.



(a) response time (b) number of requests

Figure 1: Sensitivity of FedX to the number of endpoints while fully caching the results of source selection (best setting for FedX). The response time is proportional to the number of remote requests.

In our own work building a distributed social network [4] and collaborating with Bio2RDF, we found that the state-of-the-art systems, such as FedX [8], SPLENDID [2], and HiBISCuS [6], cannot meet the needs of these applications since they have a scalability limitation in terms of the number of datasets (endpoints) they can access. This limitation becomes more serious when these endpoints are geo-distributed, i.e., are not in the same cluster. A recent survey [5] shows that FedX [8] outperforms all existing systems. However, FedX was not able to process all queries of LargeRDFBench[2], a benchmark which has 13 different real datasets with up to 1 billion triples.

The main limitation of the state-of-the-art systems comes mainly from the way they decompose the query into subqueries that are then sent as HTTP remote requests to relevant endpoints. In addition, these systems utilize bound joins which limit the amount of parallelism a system can achieve. In fact, we conducted an experiment that checks the relationship between the size of intermediate results, the number of remote requests, and query response time using FedX [8] with two queries: a *Drug* query and query *Q2* from LUBM [3]. The *Drug* query finds medicines that target asthma and optionally gets information about these medicines. It uses up to 4 datasets, corresponding to 4 endpoints. Query *Q2* finds graduate students who are registered in courses delivered by their advisor. Each endpoint in the LUBM benchmark corresponds to a university, and adding more endpoints corresponds to adding more universities. By varying the number of endpoints and measuring the response time and the number of remote requests (see Figure 1), we

[1] http://bio2rdf.org/

[2] https://github.com/AKSW/LargeRDFBench

[3] http://swat.cse.lehigh.edu/projects/lubm/

SIGMOD'17, May 14-19, 2017, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/3035918.3058731

see a clear correlation between the query response time and the number of HTTP requests generated for remote queries, which limits scalability. Processing one triple pattern at a time while binding query variables to values from intermediate results causes a huge number of remote requests.

In this demo, we highlight the architectural design choices of Lusail as a geo-distributed graph engine for querying linked RDF data. Section 3 outlines the main novel ideas for our locality-aware query decomposition and selectivity-awareness and parallel execution. In Section 4, we show how Lusail will be demonstrated using real data and synthetic benchmarks, with datasets of total sizes of billions of triples. For real datasets, we will show Lusail's performance in answering real queries on the 35 Bio2RDF endpoints. We will also deploy the 13 datasets of LargeRDFBench over 7 different regions in the USA and Europe of the Microsoft Azure cloud. For the synthetic benchmark, we will show query performance against 256 endpoints of the LUBM benchmark. The audience will be able to query these datasets. We will also give a glimpse on how Lusail outperforms state-of-the-art systems by up to three orders of magnitude and scales-up to 256 endpoints, whereas other systems cannot scale beyond 4 endpoints.

2. FEDERATED SPARQL QUERIES - THE STATE-OF-THE-ART

A federated SPARQL query cannot be answered by a single endpoint as it requires joining partial results from multiple endpoints. For example, the query in Figure 2 finds US presidents, their parties, and associated news articles.

A federated SPARQL query does not necessarily specify the endpoints against which it is processed. Therefore, the first step in federated SPARQL query processing is *source selection*; selecting the endpoints to be queried. There are two approaches for source selection: either use a list of endpoints [8], such as the list maintained by W3C[4], or discover relevant endpoints on the fly by looking up URIs and live exploration [3]. For a list of endpoints and for each triple pattern, the federated query processor asks each endpoint whether it can answer the given triple pattern. For example, the triple patterns in lines 2 and 4 (Figure 2) can be answered by the DBpedia endpoint while the one in line 5 (Figure 2) can be answered by the NYTimes endpoint. Any endpoint, which can answer at least one of the query triple patterns, is considered relevant and must also be queried.

Several systems, such as SPLENDID [2], HiBISCuS [6], and FedX [8] adopt a naïve query processing approach by iterating over all triple patterns in the query to: (*i*) evaluate the triple pattern against the corresponding endpoints, (*ii*) fetch intermediate data, and (*iii*) utilize bound join with the fetched intermediate data to solve the next triple patterns. Figure 3 demonstrates this approach for the query in Figure 2. The first iteration starts with the first triple pattern on line 2, and gets the names of US presidents as intermediate solutions. Then, each of these solutions binds the query variable *?president* to one of these names. After that, remote requests are sent to get solutions for the variable *?x*. The federated query processor then outputs the required query answers. These systems attempt to optimize the naïve approach in different ways. One obvious optimization is based on analyzing the endpoints' schemas. If

[4]http://www.w3.org/wiki/SparqlEndpoints

```
1  SELECT ?president ?party ?page WHERE {
2          ?president a dbpedia-yago:PresidentsOfUSA .
3          ?x <http://www.w3.org/2002/07/owl\#sameAs> ?president .
4          ?president <http://dbpedia.org/ontology/party> ?party.
5          ?x <http://data.nytimes.com/elements/topicPage> ?page. }
```

Figure 2: A federated SPARQL query to find US presidents and associated news articles.

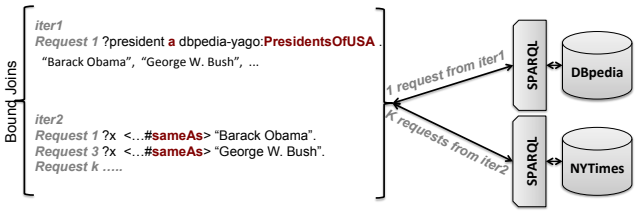


Figure 3: Traditional federated query processing.

a set of triple patterns has a solution exclusively at a single endpoint, these patterns are sent together to that endpoint as one request. We call this *schema-based decomposition*. Another optimization uses block nested loops joins to send multiple triple patterns to an endpoint as one request (e.g., the k patterns sent to NYTimes in Figure 3 can be sent as one request). Other optimizations include gathering offline statistics about ontologies or cardinalities of triples at different endpoints to reduce the online cost of query processing.

The effectiveness of schema-based decomposition is limited by the fact that endpoints belonging to the same domain (e.g., universities, government, or hospitals) utilize similar ontologies. Hence in many cases, a triple pattern can be answered by multiple endpoints. This is also the case for common predicates such as *owl:sameAs* and *rdfs:seeAlso*, which appear in most endpoints [7]. If a triple pattern can be answered by multiple endpoints, it cannot be part of an exclusive group and needs to be sent to the endpoints as an individual triple. This leads to many queries being processed one triple pattern at a time. These triple patterns retrieve a large amount of data, which is then bound to other variables and sent to other endpoints to compute the query results. Therefore, current federated RDF systems retrieve unnecessary data from the endpoints and make numerous round trips to the endpoints, which leads to poor scalability and poor query response time. In addition, the nature of the bound join operation and the binding process limit the available parallelism since only one join step is processed at a time, and the federated query processor has to wait for the results of this join step before issuing the next join.

3. SYSTEM ARCHITECTURE

Lusail's architecture is shown in Figure 4. Lusail optimizes federated SPARQL query processing via two strategies: (*i*) at compile time, Lusail employs a novel locality-aware decomposition technique. Lusail's decomposition is based not on the schema but rather on the actual locations of data instances satisfying the query triple patterns. This decomposition increases parallelism in the query execution and minimizes the retrieval of unnecessary data from the endpoints. (*ii*) At run time, Lusail employs a selectivity-aware and parallel query execution technique that orders subqueries based on their selectivity. It delays subqueries expected to return large results, and chooses the join order of the subqueries that achieve a high degree of parallelism.

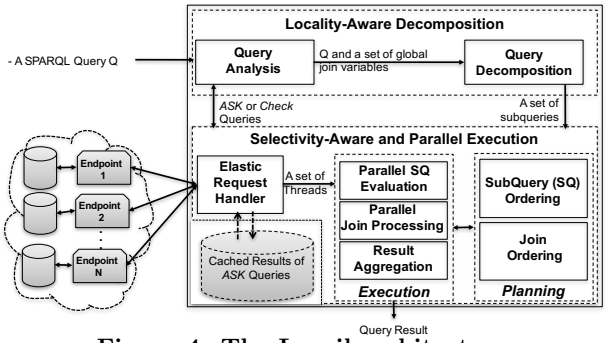


Figure 4: The Lusail architecture.

3.1 Locality-Aware Decomposition

To avoid the processing of one triple pattern at a time, we utilize the knowledge of the locations of the actual RDF triple instances matching a query variable. Consider an example from the LUBM benchmark, where each endpoint represents a university. Knowing the actual location of these instances leads to two different cases of processing triple patterns: (*i*) local instances, e.g., the instances matching the variable `?Stud` in ⟨`?Stud`, ub:advisor, `?Prof`⟩ and ⟨`?Stud`, ub:takesCourse, `?Course`⟩ are located in the same endpoint; i.e., all advised students are taking courses. (*ii*) remote instances, e.g., one of the instances matching the variable `?Univ` in a university endpoint, whose triples are ⟨`?Prof`, ub:PhDDegreeFrom, `?Univ`⟩ and ⟨`?Univ`, ub:address, `?Adrs`⟩, is located in different endpoints, i.e., there is a professor working in a given university that is different from the university he graduated from. In the former case, the triple patterns can be processed as one unit by the same endpoint while in the latter, they have to be sent separately and then joined by the federated query processor. In contrast to using exclusive groups [8] to find groupings of triple patterns based on schema information, we are proposing to find groupings based on instance information.

Existing federated SPARQL systems, such as SPLENDID [2], HiBISCuS [6], and FedX [8], cannot determine the location of triples in a general and accurate way. Hence, they retrieve unnecessary data from the data sources, leading to poor scalability and response time. In contrast to other systems, Lusail takes the additional step of checking, for each pair of triple patterns with a common (or join) variable, whether the pair can be evaluated as one unit by the relevant endpoints. The result of this check determines a group of triple patterns, i.e., a subquery, that can be sent together to an endpoint.

3.2 Selectivity-Aware and Parallel Execution

Our set of independent subqueries can be submitted concurrently for execution at each of the relevant endpoints. The results of these subqueries will then need to be joined by Lusail. Unlike other systems, each subquery is treated as an independent task, thus allowing Lusail to utilize more threads as needed. This decision is made by the Elastic Request Handler. The simplest approach is to simultaneously submit the subqueries to the relevant endpoints and wait for their results to start the joining phase. Some subqueries may affect the query evaluation time by overwhelming the network, the endpoints, and Lusail, with irrelevant data. Examples include: (*i*) generic subqueries that are relevant to the majority of the endpoints, e.g., common predicates such as *owl:sameAs*, *rdf:type*, *rdfs:label* and *rdfs:seeAlso*. (*ii*) Simple subqueries that have one triple pattern with two or three variables, e.g., ⟨`?s`, ?p, `?o`⟩ or ⟨`?s`, owl:sameAs, `?o`⟩, and (*iii*) optional subqueries.

Our cost model delays subqueries expected to return large results. Lusail assigns a thread per endpoint to collect the result of the subquery. The result is seen as a relation, whose data is partitioned among different threads. Lusail decides the join order based on the number of partitions and the actual size of the result per subquery. Thus, we can achieve a high degree of parallelism while minimizing the communication cost at two levels: (*i*) globally, by getting results from different endpoints simultaneously, and (*ii*) internally, by utilizing different threads in joining results.

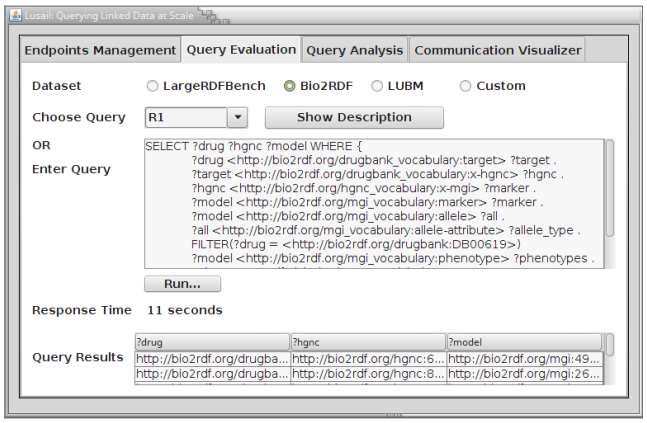


Figure 5: Demonstration interface.

Table 1: Datasets used in the demo.

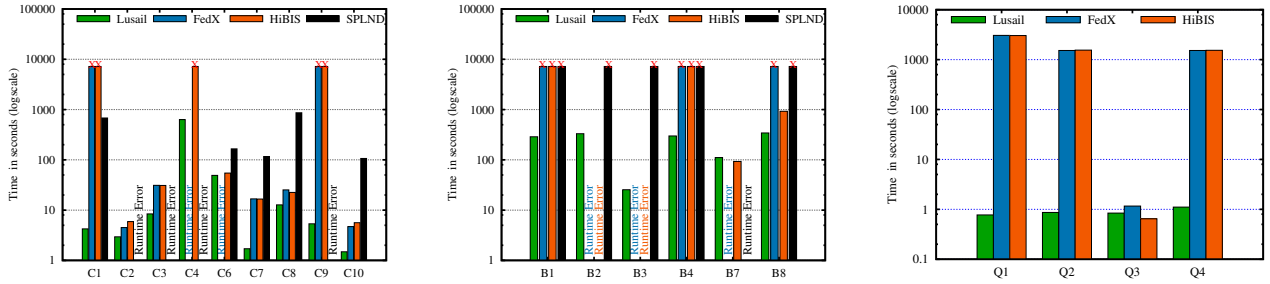
Benchmark	Endpoint	Triples(M)	Type
LUBM	256 Universities	35	Synthetic
LargeRDFBench	13 Datasets	1,004	Real
Bio2RDF	35 Datasets	11,895	Real

4. DEMONSTRATION OVERVIEW

The objective of our demonstration is to show the ease of finding and integrating data from different data sources in a large set of data sources. In real applications, such as Bio2RDF or data.gov, data scientists have to first identify relevant data sources to the task under investigation. Then, they either manually query the endpoints one-by-one and compose the results, or instead use the SERVICE keyword to name an endpoint from which a specific triple pattern has to be answered[5]. For example, one may be interested in evaluating the disparity between the amount of research and the burden of a specific disease, e.g., tuberculosis. To do so, the data scientist has to know that the relevant data sets are PubMed, GHO and LinkedCT. Similarly, getting the phenotypes of knock-out mouse models for the targets of a selected drug requires joining data from four Bio2RDF datasets; namely DrugBank, HGNC, MGI and BioPortal. These relevant datasets are out of a list of tens or even thousands of datasets, such as in data.gov.

Instead of explicitly specifying the relevant endpoints, users submit queries to Lusail against a large set of potential endpoints and get the final results in a reasonable time, as shown Figure 5. We will demonstrate how our system can work with any set of SPARQL endpoints without any preprocessing. Participants can add/delete to the list of

[5]https://github.com/bio2rdf/bio2rdf-scripts/wiki/Query-repository



(a) LargeRDFBench: Complex Queries (b) LargeRDFBench: Large Queries (c) LUBM Queries: Two Endpoints

Figure 6: Geo-distributed federation: endpoints are deployed in 7 different regions of the Azure cloud. Communication cost affects all systems, but Lusail can execute all queries and outperforms other systems.

endpoints/datasets as needed. Note that for other systems, such as SPLENDID [2], and HiBISCuS [6], the data must be preprocessed and indexed offline. Participants can write federated SPARQL queries, or choose one of the predefined queries. The predefined queries can be selected either from a real query log (Bio2RDF), or from benchmark queries (LargeRDFBench and LUBM). The benchmarks provide a mixture of queries with varying structural characteristics and selectivity classes. Participants will be able to see our query analysis, decomposition and subquery order. Moreover, our demo provides a visualization of the amount of communication incurred during query evaluation between the relevant endpoints and Lusail.

4.1 Demonstration Setup

We will showcase Lusail with real and synthetic datasets (see Table 1). LargeRDFBench is a recent federated benchmark of 13 interlinked real datasets pertaining to different domains including DBpedia (wikipedia infoboxes), LinkedMDB (movie database), GeoNames (geographical data), Drugbank and Cancer Genome Atlas. Bio2RDF is a major provider of linked data in life sciences. It contains 11 billion triples connecting 35 different biological datasets, including UniProt (protein database), KEGG (Genes and Genomes), PubMed (medical publications), and the Gene Ontology. Using LUBM, we generate data for 256 universities, each with around 138K triples. The data includes links between the different universities through students and professors. We use Virtuoso 7.1 as the SPARQL engine at the endpoints.

The audience will experience Lusail through two different settings: Queries to Bio2RDF will be answered by directly accessing the actual endpoints while LargeRDFBench and LUBM will be deployed on our own infrastructure. More specifically, we simulate a real setting where Lusail and endpoints are deployed on the Azure cloud in 7 different regions in the USA and Europe. We will use 18 instances of 16 cores and 28 GB memory. These instances will host the 13 LargeRDFBench and the 256 LUBM endpoints. Lusail and its competitors are deployed on a D5_V2 instance (16 cores, 56 GB memory) in Central USA, while none of the 18 instances is located in Central USA.

4.2 Experimental Evaluation

We now show some results of our experiments to evaluate Lusail by simulating a real scenario on the cloud as well as using real endpoints, as described earlier. Figures 6(a) and 6(b) show the query response times of both complex and large queries on LargeRDFBench. For complex queries, FedX timed out on two queries and gave runtime errors in two others. HiBISCuS timed out on three queries but did reasonably well in the rest. SPLENDID was able to evaluate only five out of the ten complex queries. Lusail outperformed all other systems in almost all complex queries, in some cases by up to two orders of magnitude (*C1* and *C9*). Large queries show the same behavior. Lusail is the only system that returns results (no time out or runtime errors). Figure 6(c) shows results on two endpoints of the LUBM dataset. All queries finished in around 1 second. In contrast, both FedX and HiBISCuS require more than 1,000 seconds; an order of magnitude compared to their performance on the local cluster. This shows their sensitivity to the communication overhead since they tend to communicate large volumes of data. With four endpoints, FedX and HiBISCuS were able to evaluate only Q3 and ran out of memory or timed out in the rest. Lusail managed to process the 4 LUBM queries in less than 8 seconds on 256 endpoints.

5. ACKNOWLEDGEMENT

We would like to thank Michel Dumontier, scientific director for Bio2RDF project, for the fruitful discussions regarding using Lusail as a federated engine for querying Bio2RDF datasets and for providing us with the query logs.

6. REFERENCES

[1] M. I. Ali, N. Ono, M. Kaysar, K. Griffin, and A. Mileo. A semantic processing framework for IoT-enabled communication systems. In *Proc. of ISWC*, 2015.

[2] O. Görlitz and S. Staab. SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In *Proc. COLD*, 2011.

[3] O. Hartig. SQUIN: A traversal based query execution system for the Web of linked data. In *Proc. SIGMOD*, 2013.

[4] E. Mansour, A. V. Sambra, S. Hawke, M. Zereba, S. Capadisli, A. Ghanem, A. Aboulnaga, and T. Berners-Lee. A demonstration of the solid platform for social web applications. In *Proc. WWW*, 2016.

[5] M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, and A.-C. Ngonga Ngomo. A fine-grained evaluation of SPARQL endpoint federation systems. *Semantic Web Journal*, 7(5), 2015.

[6] M. Saleem and A. N. Ngomo. HiBISCuS: Hypergraph-based source selection for SPARQL endpoint federation. In *Proc. ESWC*, 2014.

[7] M. Schmachtenberg, C. Bizer, and H. Paulheim. Adoption of the linked data best practices in different topical domains. In *Proc. ISWC*, 2014.

[8] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization techniques for federated query processing on linked data. In *Proc. ISWC*, 2011.