ESPRESSO: A Framework for Empowering Search on Decentralized Web

Mohamed Ragab[1](✉), Yury Savateev[1], Reza Moosaei[2], Thanassis Tiropanis[1], Alexandra Poulovassilis[2], Adriane Chapman[1], and George Roussos[2]

[1] School of Electronics and Computer Science, University of Southampton, Southampton, UK
{ragab.mohamed,y.savateev,t.tiropanis,adriane.chapman}@soton.ac.uk
[2] School of Computing and Mathematical Sciences, Birkbeck, University of London, London, UK
{r.moosaei,a.poulovassilis,g.roussos}@bbk.ac.uk

Abstract. The centralization of the Web has led to significant risks to privacy, security, and user autonomy, prompting the need for decentralization. *Solid* is a set of standards, protocols, and technologies that seeks to enable Web *re-decentralization* based on the existing W3C recommendations. In *Solid*, users store their data in *personal online data stores* (pods) with full control and sovereignty over which individuals and applications get access to them. However, the current state of the Web and Web-based applications rely heavily on search functionality using *centralized* indices. This poses significant challenges when it comes to searching or querying large-scale data stored in *decentralized* user-controlled pods, where different individuals and applications have varied access to data. To address this gap, we propose the *ESPRESSO* framework, which aims to enable individuals or applications to search Solid pods at a large scale while pod owners maintain control over access to their data. *ESPRESSO* considers *access rights* and *caching needs* while facilitating the performance of distributed queries. Our framework offers a vision for empowering search utilities in the decentralized Web, utilizing the Solid framework, and opens up new research directions for future decentralized search applications.

Keywords: Web Search · Decentralized Web · Linked Data · Personal Online Data Stores (pods) · Solid Framework

1 Introduction

The Internet has become an integral part of our daily lives, enabling us to access vast amounts of information and connect with people all around the world [18]. However, the current state of the Web is highly centralized, with a few large corporations (e.g., Google, Facebook, Amazon etc.) controlling a significant portion of online activity and user data. This centralization poses significant risks to privacy, security, and user autonomy [7]. Moreover, data stored in such centralized data silos are not readily available to other application providers to offer further added-value services [3].

Given these concerns, there have been several proposals for decentralizing the Web [3,7]. Decentralization refers to the distribution of control and ownership of data and infrastructure, enabling individuals to manage access to their online activity and data. Decentralized technologies aim to create a more equitable and transparent online environment that empowers users to maintain control over access to their data, rather than relying on third parties to centrally store it and manage access to it. This will foster data-driven advancements such as sharing and synchronizing data across various applications [3,7]. Decentralization requires the nature of applications to evolve from data in silos to *shared views* of data by decentralized applications. Last but not least, web decentralization encourages healthy competition and diversity of development by application providers [16].

One such decentralized technology that has gained significant attention in recent years is the Solid[1] technology suite [7,8,17]. Solid has been proposed as a solution to enable Web *re-decentralization* by empowering users to maintain their data in personal online data stores, also known as pods. Solid incorporates elements of the W3C *Linked Data Platform* (LDP) recommendation to enable *read/write* access to pod-stored resources, with special provisions for managing Linked Data. Solid provides standards and technologies for deploying pods with full sovereignty of users (identified by *WebID* specifications[2]) over their data, rather than having it stored in several separated data silos each controlled by third parties, such as Web platforms and applications. Application providers are required to acquire the consent of pod owners before accessing and utilizing their data [17]. Specifically, a third-party application can access data in a user's pod only if its *WebID* is associated with specific access rights recorded by the user in the *Access Control Lists* (ACLs[3]) associated with each of their resources.

While the concept of a decentralized Web holds great promise, there are still significant challenges to overcome, particularly in the area of search and query processing [20]. The development of the Web has been significantly aided by *search* as a key component, allowing more engagement from ever-growing numbers of users [18]. Search engines have always been an essential part of the Web, enabling users to access relevant information quickly and efficiently. However, most modern search engines only offer *centralized* search services, while emergent decentralized search engines do not provide search over resources where different users and applications can have different access rights. Therefore, we argue that the data search over pods is a challenging and essential problem to investigate. Empowering applications that access personal data (subject to users' stated access control permissions) with search capabilities is crucial for a successful decentralized Web [20]. However, the current search utilities across Solid applications [8] and other current distributed, federated, or Linked Data query processing systems [15] do not yet provide adequate solutions to this problem. Indeed, there is a lack of an efficient system in decentralized Web ecosystems that enables search capabilities while ensuring the privacy and security of search queries and results.

[1] https://solidproject.org.

[2] WebID specification https://www.w3.org/2005/Incubator/webid/spec/identity/.

[3] Web Access Control specifications: https://www.w3.org/wiki/WebAccessControl.

To fill in this research gap, this paper presents *Efficient Search over Personal Repositories - Secure and Sovereign* (*ESPRESSO*), a novel framework that aims to enable large-scale data search across Solid pods while respecting individuals' data sovereignty, considering individuals' different *access rights* and *caching needs*. The ESPRESSO framework takes into consideration the requirements of the various Solid stakeholders, i.e., pod users, pod providers, search issuers, as well as information regulating entities. It aims to enable *privacy-respecting* data discovery for applications relating to domains such as healthcare, social networking, and education, with an initial emphasis on health and wellness. It will allow research, development, and testing of new indexing algorithms and query optimization techniques for search in decentralized ecosystems. Last but not least, it will foster experimentation and benchmarking of various search scenarios across large-scale decentralized Solid pods.

The remainder of the paper is organized as follows. Section 2 presents the design and implementation of the ESPRESSO framework. Section 3 provides a preliminary evaluation of the ESPRESSO system with a description of the experimental setup. Section 4 presents and discusses the results of the experiments. Section 5 provides an overview of related work. Section 6 discusses the prospective challenges ahead for the ESPRESSO project. Finally, Sect. 7 concludes the paper and discusses future research directions.

2 *ESPRESSO* Framework Architecture

This section discusses the design principles of the *ESPRESSO* framework together with implementation details for each core component of the framework. We believe that building an efficient search system within the Solid framework requires exploring the following challenges:

C.1 How to index users' data in a way that respects the access control requirements and does not jeopardize privacy.
C.2 How to effectively search a large amount of distributed indices across Solid servers' pods.
C.3 How to ensure that the search results only contain the data the search party is allowed to access.
C.4 How to efficiently route and propagate search queries across Solid servers.

ESPRESSO aims to establish a clear understanding of the challenges and requirements for enabling decentralized search across Solid pods. It also aims to provide an efficient distributed system that can facilitate the routing of both *keyword-based* searches and *declarative* more complex queries, e.g. SPARQL queries, across Solid pods for both *exhaustive* and *top-k* search scenarios while ensuring the privacy and security of the search queries and results.

The *ESPRESSO* framework needs to work with different deployment settings (ranging from online cloud-based Solid servers with thousands of pods, to a personal Solid server with a single pod) and to handle large-scale data. This requires designing distributed indexing mechanisms and query optimization techniques

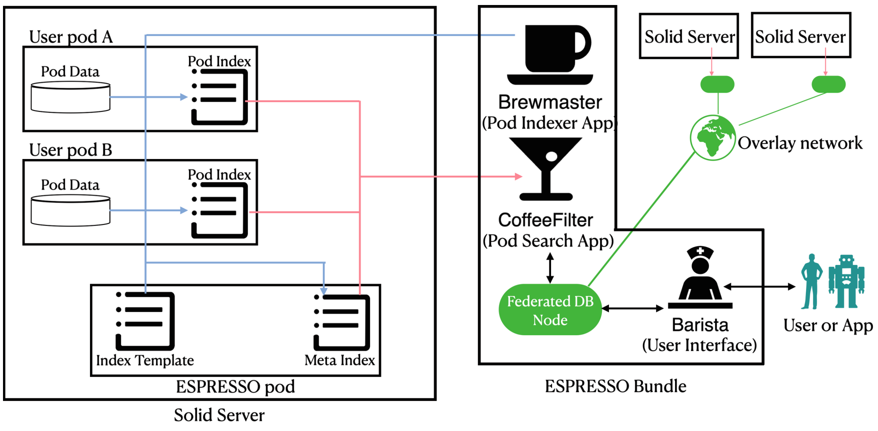

Fig. 1. *ESPRESSO* framework architecture.

for improving search performance. ESPRESSO also aims to build a decentralized search prototype implementation that enables experimentation and benchmarking on the search utilities, scenarios, and styles across Solid pods. This includes developing tools and frameworks for testing and validating the proposed search approach and ensuring its scalability and efficiency.

The *ESPRESSO* framework is built on several design principles that are summarized below alongside the specific challenges they address:

1. **Ensuring data sovereignty**. ESPRESSO should build and maintain *distributed* indices inside Solid pods rather than centralized ones, and maintain also Metadata indices to describe information on index access that can be used by search optimization algorithms (**C.1, C.3**).
2. **Respecting access control**. Enabling distributed keyword-based search and also supporting decentralized SPARQL querying while ensuring access control rights (**C.1, C.3**).
3. **Scalability**. Empowering search scenarios based on Solid servers that range from a few machines to hundreds/thousands of machines for implementing a diversity of applications (**C.2**).
4. **Decentralization**. We use a federated overlay network to propagate the queries and retrieve the search results. The search can be initiated from any node of the network (**C.4**).
5. **Privacy over efficiency**. Use of metadata for guiding query routing, and other search optimization techniques, should not compromise data privacy (**C.1, C.4**).

Figure 1 shows the abstract architecture of the ESPRESSO framework, depicting the core components that enable an end-to-end vision of how to empower search over pods. The online search functionality offered by the ESPRESSO system enables search for data stored in pods on Solid servers. First, a pod indexing application (called "*Brewmaster*") within the system creates a local index for each pod's data on a Solid server. These local indices are stored in the pods themselves

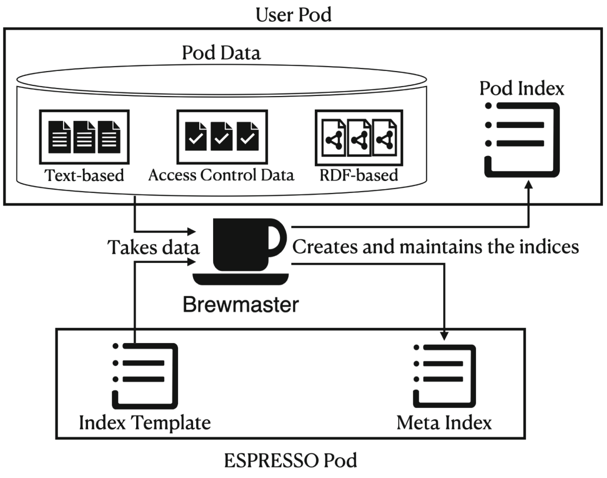

Fig. 2. Espresso pod indexing using the *Brewmaster* application.

and reflect not only the data but also users' access control assertions about it; that way, they differ from traditional indices since they include such access control information for each keyword and indexed resource. ESPRESSO allows users (or software applications/agents) to execute keyword-based search and SPARQL queries over indices through a querying user interface (called "*Barista*"). The system propagates a user query through an *overlay network* that connects the Solid servers, and returns aggregated and ranked search results retrieved from the Solid servers' pods to the user. This process of query answering passes by local indices filtering and a look-up process managed by another component (called "*CoffeeFilter*") that provides search capabilities over the indices in a pod.

Both the pod indexer application (Brewmaster) and the pod search application (CoffeeFilter) need to get access for their webIDs from the pod owners. Only if pod owners have given that consent for these two applications are the applications' *WebIDs* stored in the access control list of the pods. At this point, Brewmaster has access to all data while CoffeeFilter has access only to the index files. Thus, Brewmaster can index the data and CoffeeFilter can search the resulting indices. Without the pod owners' consent, Brewmaster and CoffeeFilter cannot access the pods' data or the indices, respectively.

Below, we describe the functionality of the core components of the ESPRESSO framework that aims to effectively support search across a large number of Solid-compliant servers. We follow the design of each component with the details of our current prototype implementation.

2.1 Brewmaster (Pod Indexer App)

ESPRESSO provides a Solid indexing application called *Brewmaster* that builds and maintains indices for the Solid pods. Figure 2 shows the process of indexing in the ESPRESSO framework. The indexing process works as follows: on a Solid server, a special pod is created called the *ESPRESSO pod* that contains (1)

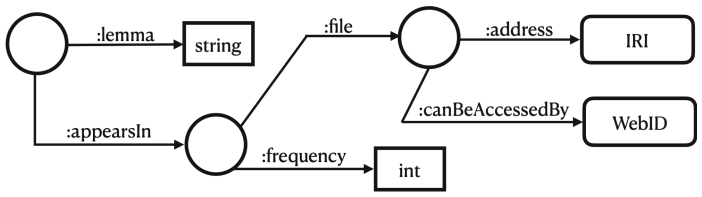

Fig. 3. Pod index format.

an *Index Template* which is an *RDF-graph* indexing template that serves as a blueprint for the pod indices, and (2) a *Meta Index* that is used for storing information about the pod indices such as index files' location and other metadata that is used by the search application for subsequent optimization and filtering purposes. Brewmaster accesses user pods for which permission is obtained by the owner for its WebID and creates a *local* index of the data stored there (pod data resources are organized hierarchically in containers) according to its type (e.g. RDF-based, or non-RDF text files) and the access control requirements provided, using the Index Template. The newly constructed pod index is saved in the pod itself. Then, Brewmaster adds the relevant information about the created local indices to the Meta Index. In the simplest case, the Meta Index maintains a list of all the addresses of the pod indices.

Figure 3 shows an example of an Index Template for *keyword-based* search and Fig. 4 shows a sample index built from this template. This RDF graph includes a node for each keyword and each of its appearances, that are connected to the relevant files (as *IRI*s), information about how many times it appears in the file, and who can access it (WebIDs). In this example, the word "*likes*" has the *lemma* "*like*". It *appearsIn* with a *frequency* of 15 times in a *file* with the *address* `http://solidserver/pod1/file1.txt` and can be accessed by one WebID: `https://bob.example/profile#me`. It also appears with a *frequency* of 3 times in a *file* with the *address* `http://solidserver/pod1/file2.txt` and can be accessed by two WebIDs: `https://alice.example/profile#me` and `https://bob.example/profile#me`.

2.2 Barista (User Interface)

Barista serves as the ESPRESSO *User Interface* (UI) application to facilitate end user search operations. Once an input query and valid login credentials are submitted, Barista leverages a *QueryBuilder* interface to construct and prepare the query for execution. Barista allows not just human users to conduct search operations, but also applications and services that can have their own WebIDs or can borrow a user's WebID when needed (through delegation [16]).

After constructing the query, Barista passes it on to a federated database (DB) node, which in turn propagates the query to other federated DB nodes within the overlay network. The results are then collected and aggregated across these nodes before being sent back to the Barista interface through the federated DB node where the query was initiated. Barista then presents the results to the user in a format that is easy to read and understand.

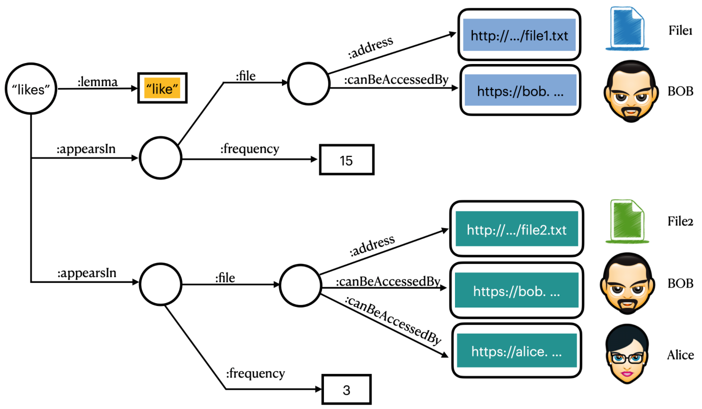

Fig. 4. An example of a pod index.

In our first prototype implementation of ESPRESSO, the search query is transformed into a GaianDB query (details on GaianDB are given in the following section), which is typically an *SQL* query. This transformation takes place in *step (1)* "Query Preparation" as shown in Fig. 5.

2.3 The Overlay Network

ESPRESSO uses an overlay network to propagate user queries to relevant data resources across Solid servers. Each solid sever in the ESPRESSO framework is connected to a federated DB node in the overlay network. The objective of query propagation is to execute the user's query while minimizing the volume of data transferred between various data sources.

The first prototype of ESPRESSO uses a data federation platform called *GaianDB*[4] for propagating user queries and aggregating search results from Solid servers [2] (as shown in *step (2)* in Fig. 5). *GaianDB* is a dynamic, distributed federated database that combines the principles of large distributed databases, database federation, and network topology. It works as a P2P overlay network that runs on a wide range of devices, addressing query propagation optimization and endpoint access restrictions. GaianDB also allows for data to be stored and accessed from multiple locations (i.e., *Store Locally and Query Anywhere*), increasing resilience and reducing the risk of data loss. Additionally, GaianDB supports query propagation across the GaianDB nodes, and result sets will be received over the *shortest paths* from nodes that can satisfy the query. Therefore, the query can be triggered at any of the GaianDB nodes and automatically propagated to the other nodes in the network. GaianDB then handles the composition of the results and sends the aggregated result back to the initiating GaianDB node. In the interests of search optimization, GaianDB also provides cache mechanisms for the preprocessed results.

[4] IBM GaianDB https://github.com/gaiandb/gaiandb.

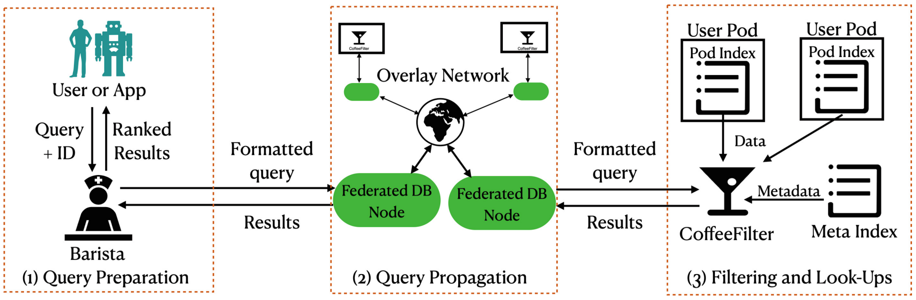

Fig. 5. Search, index look-ups, and query propagation pipeline in ESPRESSO.

In our ESPRESSO prototype, we have built a *Solid-to-GaianDB* connector component that establishes a connection between a GaianDB node and a Solid server. This connector stores the data obtained from the Solid server within a CSV file. Then, the connector generates a *logical table* (named *LTSOLID*) and maps the data from the CSV file onto the LTSOLID logical table. The Solid-to-GaianDB connector utilizes the concept of logical tables [2] for building an abstract federation layer in the GaianDB network for all data sources across solid servers' pods. Users can access all relevant data distributed across pods in different Solid servers through GaianDB nodes according to their access rights.

2.4 CoffeeFilter (Pod Search App.)

ESPRESSO provides a pod search application called *CoffeeFilter* to be installed on each Solid server. After receiving a query from the federated DB node, the *CoffeeFilter* accesses the Meta Index (created by the *Brewmaster* application) and gets the addresses of the pod indices and the relevant metadata. Then, CoffeeFilter performs a search against the relevant pod indices and sends the results back to the federated DB node.

In our first prototype of ESPRESSO, *CoffeeFilter* consists of two components, a *RESTful*[5] API and a query processing component. The API component receives the user query from a federated DB node (a GaianDB node), validates it, and then forwards it to the query processing component. The API component is also responsible for reformatting search results and returning them to that federated DB node. The query processing component accesses the Meta Index, transforms the user query into a SPARQL query, and runs it on the relevant pod indices (as shown in *step (3)* in Fig. 5). For the purpose of running a query against the indices, CoffeeFilter uses the *Comunica*[6] [19] search engine library. Comunica is a flexible SPARQL and GraphQL JavaScript library for querying decentralized knowledge graphs on the Web. Finally, the CoffeeFilter returns a ranked list of search results in JSON format to the federated DB node.

[5] https://aws.amazon.com/what-is/restful-api/.
[6] https://comunica.github.io/comunica/.

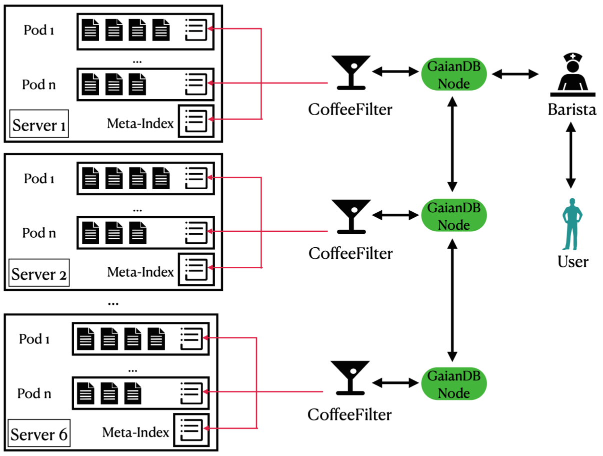

Fig. 6. Experimental architecture.

3 Preliminary Evaluation and Experimental Setup

This section outlines our experimental methodology for stress testing of the initial prototype implementation of the ESPRESSO framework. Our objective is to assess the viability and efficiency of ESPRESSO's underlying components in delivering efficient search utilities. The stress testing aimed to identify potential performance bottlenecks of our system at this early stage of development. Specifically, we evaluated the following:

– **The suitability of GaianDB** as an overlay network for propagating and routing the search queries across Solid servers. Specifically, we focused on testing the default GaianDB *query flooding* mechanism in the initial prototype implementation.
– **The efficiency of query processing** of the *CoffeeFilter* search application, which is powered by the *Communica* search engine. Our evaluation concentrated on the current implementation of keyword-based search, using a diverse set of keywords with varying frequencies across the dataset.
– **The scalability of the system** by increasing the dataset size, altering the data distribution over pods, and adjusting the number of data pods for each Solid server.

We now provide an overview of the dataset and experimental setup employed in our experiments, as illustrated in Fig. 6.

Dataset: In our evaluation, we used the *Medical Text*[7] *text-based* dataset from *Kaggle*. This dataset comprises medical abstracts that provide descriptions of various patient conditions. The original dataset has a size of 36 MB. We split

[7] https://www.kaggle.com/datasets/chaitanyakck/medical-text.

it into 100 files of roughly equal size, which we refer to as *Dataset-1*. For the scalability stress tests, we duplicated this dataset, resulting in a *72-MB* dataset of 200 files of roughly equal size (*Dataset-2*).

Environment Setup: Our experiments were executed on a cluster of 6 virtual machines with a Red-Hat Enterprise Linux 8.7 OS, running on 2.4 GHz per node processor, and 8 GB of memory per node, alongside a high-speed 125 GB drive for each node. The CoffeeFilter App. uses the Communica search engine V2.0.1 to support SPARQL query capabilities on RDF indices. We also used GaianDB V2.1.8.

Experiment Setup: We ran our experiments both on a single server and on the 6-machine cluster. For the experiments that involve multiple pods, we distributed the dataset files evenly (i.e., a uniform distribution) to reflect environments where each party has comparable contributions, or with a *Power-law* distribution (i.e., according to the *Zipf's* law) to reflect real-world scenarios where the majority of the data is contributed by few parties. The Brewmaster removes *stop words* and creates an inverted index of the dataset files with the keywords and their frequencies in each pod (Section 2.1).

For our tests, we chose *three* keywords – *Kwd1: disease*, *Kwd2: corona*, and *Kwd3: hemoproteins* – which appear in the dataset files *frequently*, *moderately* and *rarely*, respectively. We submit a keyword query from one machine and wait for the search results (c.f. Figure 6).

Performance Evaluation Measure (Response Time): We used the response time as a metric to measure the *latency* of query searches. Specifically, we recorded the time taken by the slowest search application instance in the cluster and the total time of the search operation. Thus, we were able to estimate the time taken to federate the query and results across the servers by GaianDB. For each experiment, we conducted tests using the three different keywords. For each experiment, we ran the query *five* times. We excluded the *first* run to avoid *warm-up* bias and computed an average of the other *four* run times.

4 Experiment Results and Discussion

In this section, we present and discuss the experimental results of our preliminary evaluation of the current implementation of ESPRESSO. We also share our experience dealing with querying decentralized indices in the Solid ecosystem.

Tables 1 and 2 show the results of our experiments on *Dataset-1* and *Dataset-2*, respectively. For each experiment, the tables show the search *keyword* (*Kwd*) and the number of rows fetched (*Res*). For each setup, we show the number of servers used (S), the average total response time (TT), including the CoffeeFilter search time (ST), and GaianDB routing time (RT). The upper part in each table addresses the results of the single-server experiments (S=1), while the lower part shows the 6-servers results (S=6). Within each part in each table, we show the results of two different distributions, i.e., Zipf and Uniform. Notably, *N/A*s shown in the tables represent the non-applicability of distributing the data on a single server using Zipf distribution.

Table 1. Experiment results on Dataset-1. Runtimes are in *ms*.

Dataset-1			**1pod/server**						**24 pods total**					
			Zipf N/A			**Uniform Largest index 76.8 MB**			**Zipf Largest index 21.5 MB**			**Uniform Largest index 3.5 MB**		
S	**Kwd**	**Res**	**ST**	**RT**	**TT**	**ST**	**RT**	**TT**	**ST**	**RT**	**TT**	**ST**	**RT**	**TT**
	Kwd1	100				9898	28	9926	8964	25	8989	9045	26	9071
	Kwd2	12				9372	19	9390	7562	21	7583	6026	17	6043
1	*Kwd3*	1		N/A		9463	17	9480	5431	17	5448	4468	16	4483
			Largest index 34.1 MB			**Largest index 14.5 MB**			**Largest index 21.5 MB**			**Largest index 3.5 MB**		
	Kwd1	100	4132	20	4152	3957	89	4047	4726	20	4746	5363	47	5410
	Kwd2	12	3928	16	3944	3087	61	3147	4277	15	4291	2780	34	2813
6	*Kwd3*	1	3988	16	4004	1883	39	1921	3482	16	3498	1911	40	1951

The results show that the single-server experiments (i.e., S=1, upper part of Tables 1 and 2) have longer runtimes compared to the decentralized distributed indices experiments with 6 servers (i.e., S=6, lower part of the tables). We can also observe that in most cases the search becomes faster when data is spread across multiple pods on the same server (i.e., *24 pods total*) than when it is stored in just one pod (i.e., *1pod/server*). Indeed, the slowest search performance is seen in the case of a single server with one pod (i.e., S=1 and *1pod/server*). The reason behind this is that the more files a pod has, the bigger the pod index is, requiring a longer time for CoffeeFilter to load and query. To show that, we give information about the pod's *"Largest index"* size in the tables. This finding is supported also by the fact that in cases with the Zipf distribution (that puts more files into some pods, resulting in bigger index files) the search tends to perform slower compared to the cases with the uniform distribution. Comparing the results of the Dataset-1 (Table 1) and Dataset-2 (Table 2) experiments shows that the runtimes scale predictably with the dataset size.

Last but not least, it is evident that in all cases more than 99% of the total response time of ESPRESSO is taken by the search application (ST) and less than 1% is taken by the GaianDB routing (RT).

4.1 Results Discussion

In all cases, we have seen that in the keyword-based queries, most of the time for end-to-end search in the ESPRESSO prototype is taken by the search component that fetches results at each server. The time taken by GaianDB to propagate and route the query and aggregate the results is negligible by comparison. Indeed, the GaianDB overlay network shows a consistent stable performance for routing the queries and aggregating the results in our preliminary evaluations. On the other hand, the CoffeeFilter search application employs the *Communica* engine

Table 2. Experiment results on Dataset-2. Runtimes are in *ms*.

Dataset-2			**1pod/server**						**24 pods total**					
			Zipf			**Uniform**			**Zipf**			**Uniform**		
			N/A			**Largest index 76.8 MB**			**Largest index 21.5 MB**			**Largest index 3.5 MB**		
S	**Kwd**	**Res**	**ST**	**RT**	**TT**	**ST**	**RT**	**TT**	**ST**	**RT**	**TT**	**ST**	**RT**	**TT**
	Kwd1	200				23655	20	23675	19082	33	19115	20114	45	20159
	Kwd2	24				25064	11	25075	17085	18	17103	15599	19	15618
1	*Kwd3*	2	N/A			23160	12	23171	11785	15	11800	10078	18	10097
			Largest index 68.2 MB			**Largest index 28.3 MB**			**Largest index 44.6 MB**			**Largest index 7 MB**		
	Kwd1	200	8013	22	8035	7957	141	8098	10056	23	10079	4980	77	*5057*
	Kwd2	24	8306	18	8324	5921	88	6010	9903	16	9918	5971	83	6055
6	*Kwd3*	2	8085	16	8101	5294	44	5337	7467	18	7485	3439	48	3487

that requires loading index files into memory to run SPARQL queries on them. The size of the index files has a high impact on total response time.

The current implementation of the Solid ecosystem does not support *server-side* SPARQL executions [5]. Solid currently only provides authorization and LDP-style querying at the document level. Solid project maintainers encourage client-side SPARQL query engines for processing RDF pod data, such as Communica [19]. Thus, querying RDF resources on pods using SPARQL entails loading the entire document and delegating the query execution to the client-side engine. We believe that implementing a server-side querying paradigm will significantly enhance search component performance (i.e., only filtered search results will be transferred to client-side applications).

One more aspect to consider is that the current implementation of LDP-style querying in Solid may also violate the Solid principle of data sovereignty, retrieving and revealing more data than is necessary to answer the query. This opens up future research directions for the ESPRESSO project on how to search RDF resources (such as RDF indices), allowing only the necessary data to be retrieved to enhance search performance while respecting users' data sovereignty. This would further support application development on Solid, allowing apps to search data across pods to which they have access, without having to extract and aggregate such data on a central repository for search purposes.

Also, we are currently investigating alternative indexing schemes that do not require server-side SPARQL executions while enforcing data sovereignty. Our preliminary tests of a simple indexing scheme, where we create an index file for each keyword, and access only this small index file while querying, show at least *four-fold* improvement in search run-time.

5 Related Work

The decentralized web has received considerable attention in the academic and research literature [3,7]. The challenge of performing a search within a decentralized data ecosystem is complex because it requires handling keyword or database queries that are dispersed across a vast network of thousands of data stores, each with varying accessibility and unique data governance constraints related to storage and migration. Different search parties may have distinct access rights, further complicating the search process [8,17]. A real example of the requirement to maintain decentralized data but facilitate queries arises in the health domain, in which health service providers must protect their patient information, yet wish to support research and cohort generation [11]. In this section, we review existing research efforts related to search over the decentralized web.

There has been extensive research in the domain of Distributed Database Systems. These works have explored various *distributed querying* techniques that enable querying of databases administered by different organizations with varying levels of autonomy [7]. Additionally, distributed indexing techniques have been proposed to support search across multiple databases, and distributed information retrieval techniques have been investigated to select datasets for search based on metadata [4].

Peer-to-Peer (P2P) data management and query routing have also been studied as potential solutions for decentralized search [9,12]. For instance, P2P protocols such as IPFS [1] leverage *Distributed Hash Tables* (DHTs) to enable keyword-based search. *Socio-aware* P2P search has also been explored for content sharing and community detection, with a distributed index used to support search within communities [12]. In personal data management, researchers have investigated distributed querying techniques to address security concerns related to storing all data in a single location. Additionally, access control policy enforcement has been studied in P2P environments.

Despite its potential benefits, decentralization poses substantial performance challenges when it comes to searching or querying *large-scale* data stored in decentralized pods. The current state-of-the-art research efforts in distributed database querying assume that the querying party has access to query endpoints, indices, and results caching options, which may not be the case in decentralized environments. Moreover, to support decentralized SPARQL querying, endpoint metadata would need to be created and maintained for each search party, access control would need to be implemented for resource selection, and caching control would need to be enforced for SPARQL *link-following* [6]. Therefore, the techniques required for data search over Solid pods go beyond the literature in distributed keyword-based search and distributed SPARQL Linked Data querying [10,14]. This is because *access rights* to pod data can vary for different search parties, and *caching restrictions* can apply to the propagation of search results across the network [20].

6 Challenges Ahead

Further development of the ESPRESSO framework must also address the following challenges:

1. The diverse nature of data stored in user-controlled pods makes efficient search challenging. Thus, ESPRESSO requires robust data indexing, meta-

data, and an efficient distributed indexing strategy that can accommodate the diversity of data types and access levels.
2. ESPRESSO must consider the access rights of users and applications as well as users' sovereignty over their data. This impacts performance since it restricts the use of standard optimization techniques such as caching.
3. ESPRESSO employs an overlay network that federates and propagates search queries. However, larger-scale experiments are required to analyze the need for optimizations on the level of query propagation, routing, and aggregation techniques.
4. ESPRESSO must address questions related to the scalability of Solid-based search applications and their ability to handle large-scale data.
5. To enable *top-k* search in ESPRESSO, decentralized ranking algorithms must be developed.

7 Conclusion and Future Directions

Solid proposed a set of standards and tools for building decentralized Web applications based on Linked Data principles. The development of search utilities in the decentralized Web is a vital step towards realizing the full potential of decentralized technologies such as Solid. The current distributed search techniques, including distributed and federated databases, as well as distributed SPARQL query processing, do not fully cover decentralized search requirements. In this paper, we present the *ESPRESSO* framework architecture that aims to enable distributed search across decentralized Solid pods. Specifically, *ESPRESSO* aims to address the challenges of search in decentralized environments, including varying access rights to pod data and caching restrictions that can apply to the propagation of search results across a network of multiple solid servers.

The roadmap of ESPRESSO includes the following future research directions:

1. Validating the architecture, proposing and implementing different real-world search scenarios that incorporate user-defined access control and data governance constraints in several application domains (e.g., healthcare, education, and well-being scenarios).
2. Performance and scalability evaluation via comprehensive experiments on the ESPRESSO framework against several baselines (e.g., fully centralized approaches, and other decentralized optimizations and caching mechanisms).
3. Performing extensive benchmarking experiments (using frameworks like [13]), testing various search scenarios and styles (i.e., exhaustive and top-k, for both keyword-based and SPARQL queries) as well as comparing different query propagation techniques.
4. Developing new decentralized search algorithms, metadata structures, indexing and query optimization techniques in existing or emergent ecosystems.
5. Investigating the integration of ESPRESSO with existing decentralized Web technologies, such as *IPFS*, and exploring the use of machine learning techniques to enhance search relevance and accuracy.

Achieving ESPRESSO's vision within this roadmap represents a significant step towards the development of a more decentralized and user-controlled Web.

Acknowledgements. This work was funded by EPSRC (EP/W024659/1).

References

1. Benet, J.: IPFS-content addressed, versioned, P2P file system. arXiv preprint arXiv:1407.3561 (2014)
2. Bent, G., Dantressangle, P., Vyvyan, D., Mowshowitz, A., Mitsou, V.: A dynamic distributed federated database. In: Proceedings of the 2nd Annual Conference International Technology Alliance (2008)
3. Berners-Lee, T.: Long live the web. Sci. Am. **303**(6), 80–85 (2010)
4. Crestani, F., Markov, I.: Distributed information retrieval and applications. In: Serdyukov, P., et al. (eds.) ECIR 2013. LNCS, vol. 7814, pp. 865–868. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36973-5_104
5. Dedecker, R., Slabbinck, W., Hochstenbach, P., Colpaert, P., Verborgh, R.: What's in a Pod?-A knowledge graph interpretation for the solid ecosystem (2022)
6. Hartig, O.: An overview on execution strategies for linked data queries. Datenbank-Spektrum **13**, 89–99 (2013)
7. Kahle, B.: Locking the Web open: a call for a decentralized web. Brewster Kahle's Blog (2015)
8. Mansour, E., et al.: A demonstration of the solid platform for social web applications. In: Proceedings of the 25th International Conference Companion on World Wide Web, pp. 223–226 (2016)
9. Mislove, A., Gummadi, K.P., Druschel, P.: Exploiting social networks for internet search. In: 5th Workshop on Hot Topics in Networks (hotnets06). Citeseer, p. 79. Citeseer (2006)
10. Moaawad, M.R., Mokhtar, H.M.O., Al Feel, H.T.: On-the-fly academic linked data integration. In: Proceedings of the International Conference on Compute and Data Analysis, pp. 114–122 (2017)
11. Mork, P., Smith, K., Blaustein, B., Wolf, C., Sarver, K.: Facilitating discovery on the private Web using dataset digests. In: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, pp. 451–455 (2008)
12. Nordström, E., Rohner, C., Gunningberg, P.: Haggle: opportunistic mobile content sharing using search. Comput. Commun. **48**, 121–132 (2014)
13. Ragab, M., Awaysheh, F.M., Tommasini, R.: Bench-ranking: a first step towards prescriptive performance analyses for big data frameworks. In: 2021 IEEE International Conference on Big Data (Big Data), pp. 241–251. IEEE (2021)
14. Ragab, M., Tommasini, R., Eyvazov, S., Sakr, S.: Towards making sense of spark-SQL performance for processing vast distributed RDF datasets. In: Proceedings of The International Workshop on Semantic Big Data, pp. 1–6 (2020)
15. Sakr, S., et al.: The future is big graphs: a community view on graph processing systems. Commun. ACM **64**(9), 62–71 (2021)
16. Sambra, A., Guy, A., Capadisli, S., Greco, N.: Building decentralized applications for the social Web. In: Proceedings of the 25th International Conference Companion on World Wide Web, pp. 1033–1034 (2016)
17. Sambra, A.V., et al.: Solid: a platform for decentralized social applications based on linked data. MIT CSAIL & Qatar Computing Research Institute, Technical report (2016)
18. Spink, A., Jansen, B.J.: Web Search: Public Searching of the Web. Springer, Dordrecht (2004)

19. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: a modular SPARQL query engine for the web. In: Vrandečić, D., et al. (eds.) ISWC 2018. LNCS, vol. 11137, pp. 239–255. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00668-6_15
20. Tiropanis, T., Poulovassilis, A., Chapman, A., Roussos, G.: Search in a redecentralised web. In: Computer Science Conference Proceedings: 12th International Conference on Internet Engineering; Web Services (InWeS 2021) (2021)