Data Science and Engineering (2024) 9:431–448
https://doi.org/10.1007/s41019-024-00263-w

RESEARCH PAPER

ESPRESSO: A Framework to Empower Search on the Decentralized Web

Mohamed Ragab[1] · Yury Savateev[3] · Helen Oliver[2] · Thanassis Tiropanis[1] · Alexandra Poulovassilis[2] · Adriane Chapman[1] · George Roussos[2]

Received: 7 March 2024 / Revised: 19 August 2024 / Accepted: 9 September 2024 / Published online: 26 November 2024

Abstract
The increasing centralization of the Web raises serious concerns regarding privacy, security, and user autonomy. In response, there has been a renewed interest in the development of secure personal information management systems and a movement towards decentralization. Decentralized personal online data stores (pods) represent a revolutionary example within this movement, built on the W3C's existing guidelines – an approach exemplified by initiatives such as `Solid` (https://solidproject.org). In the Solid paradigm, individuals store their personal data in pods and have absolute discretion when choosing to grant access to different users and applications. A barrier to the adoption of the pod approach is the predominant reliance on centralized indexes for search functionality in current Web and Web-based systems. This paper introduces the `ESPRESSO` framework, which is designed to facilitate this new paradigm of large-scale searches within personal data stores while respecting the individual pod owners' data access governance. The current ESPRESSO prototype integrates access control within pod indexes to enhance distributed keyword-based search. ESPRESSO's unique contribution not only enhances search capabilities on the decentralized Web but also paves the way for future explorations in decentralized search technologies.

Keywords Web search · Decentralized web · Distributed querying · Access control · Linked data · Personal online data stores · `Solid` framework · SPARQL · RDF

✉ Mohamed Ragab
ragab.mohamed@soton.ac.uk

Yury Savateev
y.savateev@herts.ac.uk

Helen Oliver
h.oliver@bbk.ac.uk

Thanassis Tiropanis
t.tiropanis@soton.ac.uk

Alexandra Poulovassilis
a.poulovassilis@bbk.ac.uk

Adriane Chapman
adriane.chapman@soton.ac.uk

George Roussos
g.roussos@bbk.ac.uk

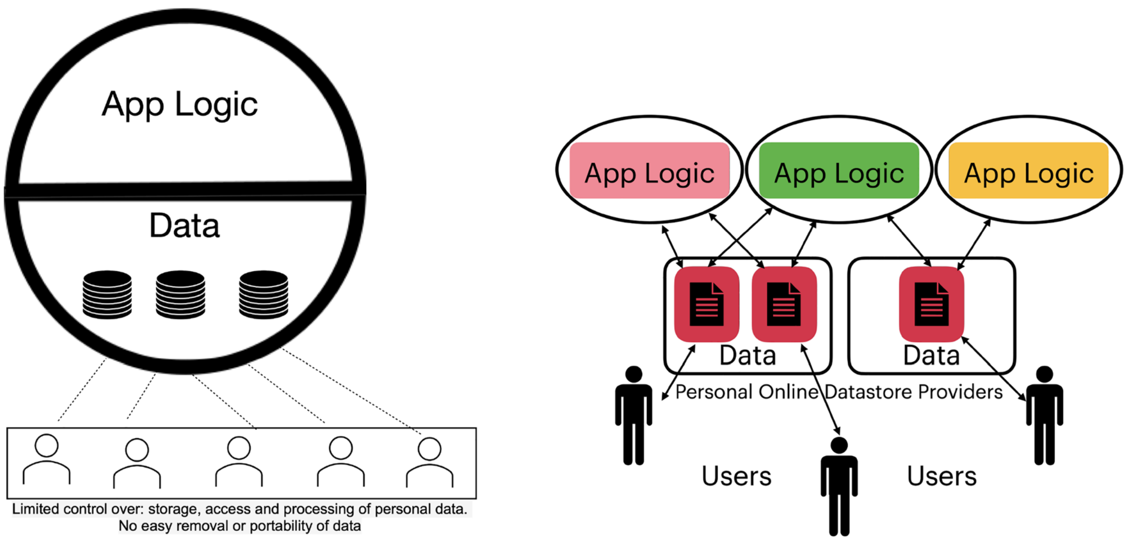
1 School of Electronics and Computer Science, University of Southampton, Southampton, UK

2 School of Computing and Mathematical Sciences, Birkbeck, University of London, London, UK

3 Department of Engineering and Computer Science, University of Hertfordshire, Hatfield, UK

1 Introduction

The usage of data-driven decision-making and the deployment of artificial intelligence (`AI`) applications present transformative opportunities with broad-ranging advantages, particularly in the rapid advancement of user-centric technologies within the realm of the World Wide Web (`WWW`). Nonetheless, these advancements are accompanied by significant concerns about data privacy [1, 2]. The original conception of the Web as an open and universally accessible platform for information exchange has given rise to substantial privacy challenges faced by its users. In the contemporary landscape, users contend with a loss of authority and control over their personal data, which is amassed and exploited by centralized online platforms [3]. Indeed, the current manifestation of the Web skews heavily towards centralization, dominated by a handful of major corporations (such as Google, Facebook, and Amazon), who wield substantial influence over online interactions and the management of user data [1]. This corporate domination not only engenders conspicuous disparities in information access and power dynamics but also frequently culminates in the


App Logic
Data
Limited control over: storage, access and processing of personal data.
No easy removal or portability of data
App Logic
App Logic
App Logic
Data
Data
Personal Online Datastore Providers
Users
Users

(a) Centralized Apps. Limited control over storage, access, and processing of personal data. No easy removal or portability of data.

(b) DeCentralized Apps — Data and App Logic Decoupling. Complete control over storage and access to personal data. Easy removal and portability of data.

Fig. 1 Centralized versus decentralized applications paradigms

violation of fundamental rights, thereby raising legitimate apprehensions about privacy breaches and the improper use of personal information [4]. This centralization of power raises profound concerns regarding privacy, security, and the autonomy of users [4, 5]. Additionally, the aggregation of data within these centralized repositories hinders its availability for other service providers, limiting the potential for enhanced value-added services [6].

In light of these concerns, various initiatives have been proposed to decentralize the Web [5–7]. Decentralization is characterized by distributing the control and ownership of data and infrastructure, granting individuals the power to govern their online activities and data. The goal of decentralized technologies is to establish a more balanced and transparent digital landscape, where users have direct control over their data, rather than depending on centralized entities for data storage and access management [8]. Such a shift is pivotal for enabling data-driven progress, including the sharing and synchronization of data across diverse applications [5, 6]. Decentralization necessitates a transformation in application design, moving from isolated data silos to *shared views* of data accessible by decentralized applications (see Fig. 1). Importantly, the move towards Web decentralization also promotes robust competition and fosters a variety of developments among application providers [9]. It also supports the increasing demand for using collective information to improve societal health and well-being [10], including initiatives such as improving our response to worldwide pandemics by compiling health and mobility records of individuals [11]. Last but not least, Web decentralization reduces risks of security and user autonomy by distributing data access and control across multiple nodes, reducing security risks like single points of failure and targeted attacks associated with centralized systems.

A prominent example of decentralized technology that has gained considerable attention is the Solid technology suite[1] [7], which proposes an approach for Web *re-decentralization*. Solid empowers users to manage their data in *personal online data stores*, or `pods` [8]. Pods are decentralized data stores that serve as secure personal web servers for user data. Solid leverages the *World Wide Web Consortium's* (`W3C`'s) *Linked Data Platform* (`LDP`) standards to facilitate *read/write* operations on data resources (text documents, *Resource Definition Framework* aka `RDF` files, etc.) stored in pods in a secure environment (organized in nested containers, similar to files in a `Unix` directory structure), with a special focus on managing *Linked Data*. Solid primarily specifies authenticated *Hypertext Transfer Protocol* (`HTTP`) access to those containers. Solid's framework ensures that users, identified through *WebID* specifications,[2] retain

[1] Solid is a set of specifications that can have several implementations, e.g., *Digita's* `use.id`https://get.use.id/.

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

complete control over their data. This approach contrasts with the traditional model of data being housed in separate, third-party controlled silos such as Web platforms and applications. Application providers, under this framework, must obtain explicit consent from pod owners to access and use their data [7]. Specifically, a third-party application is granted access to data within a user's pod only if its *WebID* has been granted specific access rights by the user, which are documented in the *Access Control Lists* (`ACLs`[3]) associated with each data resource. Thus, the Solid protocol and its specifications ensure robust privacy and security by allowing users to manage their data with an unprecedented level of control, from the creation of a Pod to detailed access and data interaction management, using standard, open data formats and protocols.

The vision of a decentralized Web is promising but faces considerable obstacles, especially in the realm of search and query operations [12]. The continuous production and publication of vast amounts of data necessitate some form of search functionality for effective navigation. Search functions have been a cornerstone in the Web's evolution, driving user engagement and growth [13]. While search engines have been integral to the Web, enabling rapid and efficient information retrieval, they predominantly operate on centralized models. Current decentralized search engines fall short in searching across resources with varied access rights for different users and applications. Thus, developing effective decentralized search capabilities for data stored in pods emerges as a critical challenge in the decentralized Web landscape [12]. Applications that access personal data, respecting user-defined access controls, must be equipped with robust search functionalities to thrive in a decentralized Web environment. However, existing search utilities in Solid applications [14], as well as in existing distributed, federated, or Linked Data query systems [15, 16], do not yet provide adequate solutions for distributed search over resources where different users, applications and search entities can have different access rights (see Sect. 7 for further discussion). An efficient system that facilitates search capabilities within decentralized Web ecosystems, while safeguarding the privacy and security of both search queries and results, is still missing.

In response to this research gap, our previous works [12, 17] have proposed a vision and architecture of *Efficient Search over Personal Repositories - Secure and Sovereign* (`ESPRESSO`). ESPRESSO is a framework that aims to enable large-scale search across Solid pods while respecting the data sovereignty of individuals, taking into account the varying *access rights* and *caching requirements* of users. The ESPRESSO framework is designed to meet the needs of different stakeholders in the Solid ecosystem, including pod users, pod providers, search issuers, and regulatory bodies [12]. Its primary objective is to facilitate *privacy-conscious* data exploration, particularly in sectors like healthcare, wellbeing, social networking, and education, with an initial focus on health and wellness applications. ESPRESSO provides a platform for researching, developing, and testing innovative indexing algorithms and query optimization methods tailored for decentralized environments. Furthermore, it aims to foster experimental analysis and benchmarking of diverse search scenarios within extensive, decentralized Solid pod networks.

Our earlier research [17, 18] presented the ESPRESSO prototype's initial version and carried out preliminary proof-of-concept load testing experiments of the initial prototype implementation of the ESPRESSO framework. In this paper, we delve deeper into the ESPRESSO framework, highlighting the progress made in its architectural components. We also provide an extensive experimental analysis of the performance of ESPRESSO, conducting a large-scale decentralized search on a large number of pods hosted on multiple servers, simulating real-world settings. Specifically, we will showcase and examine the outcomes from testing and validating the ESPRESSO system's second prototype, using an illustrative scenario from the healthcare sector.

The contributions of this paper are as follows:

1. Introducing the ESPRESSO system architecture, and discussing its components' extensions, including optimizations in the keyword indexing structures and the search algorithm proposed in [17]. We show how this architecture can mitigate the challenges of decentralized keyword search, i.e., (C1-C4) in Sect. 3.1.
2. Demonstrating the effects of optimizations on keyword indexing and search algorithms by comparing the search performance of the initial ESPRESSO prototype [17] with that of the current prototype.
3. Validating the viability of the ESPRESSO architecture for undertaking scalable decentralized keyword-based search operations over personal online datastores distributed across several Solid servers. This entails providing a working implementation of ESPRESSO and exploring it in the context of a motivating healthcare scenario (see Sect. 2.2) to illustrate the advantages and the challenges posed by the practical implementation of such a system.

The structure of this paper is as follows: Sect. 2 presents details on the limitations of centralized Web Search and how the re-decentralization of the Web search mitigates those limitations. It also presents a motivating scenario of the decentralized Search in Sect. 2.2. Section 3.1 presents the design principles and implementation of the ESPRESSO

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

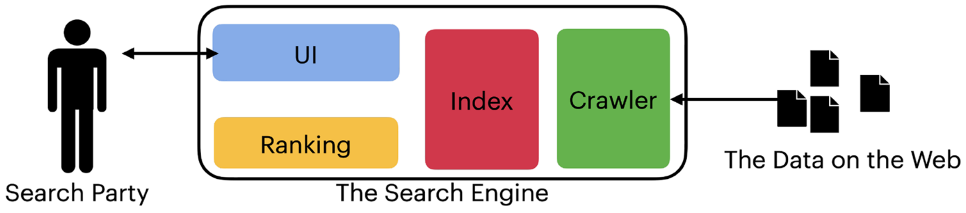


Fig. 2 Centralized search outline

framework along with the challenges it tackles. Section 4 provides an experimental evaluation of the current ESPRESSO prototype with a description of the experimental setup. Section 5 presents and discusses the results of the experiments. Section 6 discusses the prospective challenges ahead for the ESPRESSO project. Finally, Sect. 8 concludes the paper and discusses future research directions.

2 Search Dynamics & Motivating Scenario

2.1 Centralized Versus Decentralized Search

Centralized search engines, such as Google and Yahoo, function by crawling the web and creating an index of its content. This index, held by these corporations, is then searched when necessary (see Fig. 2). Centralized search engines have traditionally emphasized three core aspects: swift retrieval speeds, ensuring completeness of search results, and personalized ranking. This focus has encouraged the centralization of data storage and control, leading to a lack of transparency and potential compromises in user privacy [5]. Centralized search engines are not ideally equipped to handle data that necessitates stringent privacy measures [19]. For instance, medical data repositories, containing sensitive information, must often remain inaccessible to general search engines to maintain confidentiality. This necessitates the development of specialized search systems for such private data, which can lead to inefficiency and a lack of standardization. This fragmentation impedes comprehensive data analysis and could potentially restrict valuable research advancements in these fields. For instance, in the healthcare sector, the majority of citizens are willing to share their health data for scientific research aimed at benefiting society [20]. However, they also expect to have control over how their data is used. The non-transparent nature of data collection by centralized search engines is a growing concern for both data proprietors and users. The lack of clarity in these processes can erode trust and expose data to potential vulnerabilities. For certain types of data, the risks associated with centralized search engines' data management practices are deemed unacceptable.

While not all data should be universally accessible, individuals often require the capability to search within their accessible data domains without consolidating this data centrally. Current centralized search models offer limited, if any, support for integrating access control measures into their search functionalities. This limitation underscores the need for a more nuanced approach to search systems, one that respects privacy and access restrictions without compromising on efficiency and effectiveness.

Conversely, in a decentralized system, search results are tailored to each user's access permissions. This means two users might see different results for the same query, based on file access controls. Our aim is for the search to honor the access specifications set by the pod owner, ensuring that a user/application represented by a WebID only retrieves files it is authorized to access. The proposed decentralized search system for Solid pods (see Fig. 3) consists of: a general search app that verifies the user's WebID, processes search queries, and ranks results; and an overlay network disseminating queries across various Solid servers, where searches are conducted locally on server-specific pod indexes, with results sent back through the network. Local indexing software creates and maintains these pod indexes, ensuring sensitive pod data only leaves the server when accessed by authorized users.

2.2 Motivating Scenario

In this section, we present a hypothetical healthcare scenario that is inspired by the structure of the healthcare system in UK. The scenario demonstrates the use of our proposed decentralized search system over pods, involving confidential personal medical data stored in the pods, hosted by the *National Health Service* (`NHS`)[4] infrastructure. Such a scenario could be generalized to other domains besides healthcare. The decentralized approach aims to overcome the limitations inherent in centralized search systems, in which meaningful consent is problematic to obtain from the user [21], who faces a loss of control over what happens to the data they disclose [22], and for whom data rights are difficult to exercise in practice [23]. Under these conditions, it is difficult for users to share their data even when they want to [24].

[4] https://www.nhs.uk.

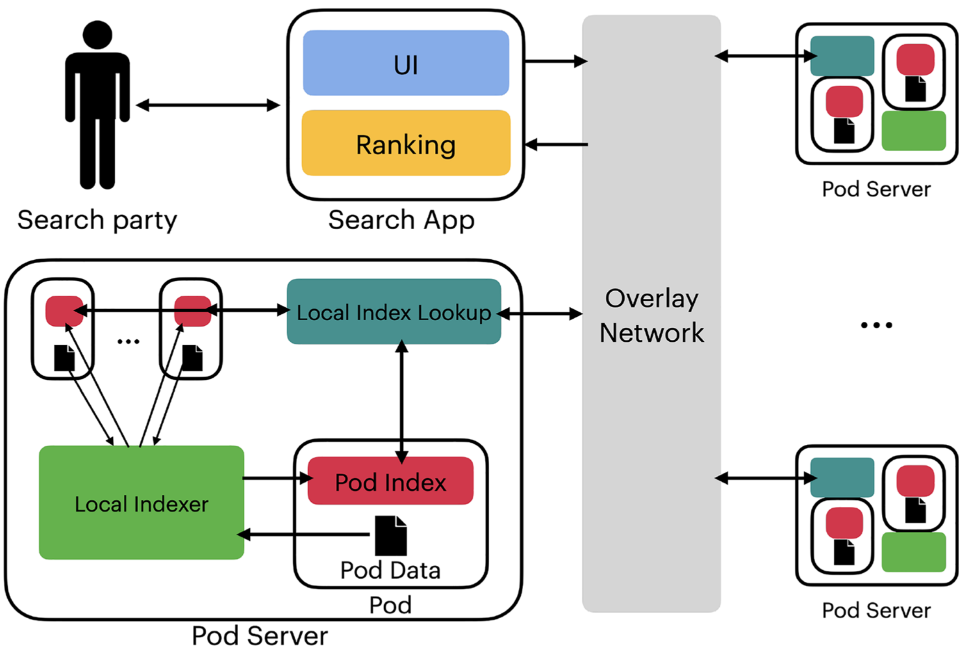


Fig. 3 Decentralized search outline

For this scenario, we assume that the search takes place in a Solid environment, where individuals store their medical data in pods hosted by the NHS. Individuals can import their medical records to those pods. These records may include General Practitioner (GP) notes, hospital visit reports, etc.

Alice is a medical researcher who is seeking potential participants for an approved initial-stage clinical trial of a novel treatment. She wishes to choose participants by reviewing their patient records. Alice has to request access from individuals to their pods in order to use their medical health records in the trial. Alice aims to search across these individuals' pods using keyword queries describing rare medical conditions or particular medications.

We therefore require solutions to the following problems:

- How can Alice search to find individuals with matching health records for her study, among those individuals who have given consent for their health data to be searched? And:
- How long would such a keyword search operation take to find such individuals in a metropolitan area comprising hundreds of thousands of patients?

To address these issues, individuals should first give consent to Alice's access request to their medical records, i.e., by adding Alice's WebID to the Access Control Lists of the health records they wish to share. This allows Alice to utilize her WebID to search within pods to which that WebID has been granted access, thereby allowing her to compile a list of individuals with a medical history of the condition/medication she is searching for. These individuals may also have consented to be contacted, i.e., may have made their contact information accessible to Alice's WebID. Consequently, Alice can directly reach out to these potential trial participants, inviting them to join the trial.

3 ESPRESSO Framework

This section discusses the design principles of the ESPRESSO framework, including specific implementation aspects for each essential part of the framework. We hold the view that the development of an effective search system within the Solid ecosystem necessitates an investigation into 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 indexes 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 is focused on the above challenges to enable decentralized searching over Solid pods which encompass diverse access permissions to pod data and constraints related to access control. Additionally, it aims to deliver an efficient distributed system supporting both *keyword-based* searches and more sophisticated *declarative* queries, for instance, `SPARQL` (*SPARQL Protocol And RDF Query Language*) queries, across Solid pods. This encompasses both *exhaustive* and *top-k* search situations, at all times preserving the privacy and security of search queries and their results. The ESPRESSO framework is designed to function in various deployment environments, from cloud-based Solid servers hosting thousands of pods to individual Solid servers with a single pod, while managing extensive data sets. This necessitates the creation of distributed indexing methods and query optimization strategies to enhance search efficiency. Additionally, ESPRESSO focuses on constructing a decentralized search prototype that facilitates experimental analysis and benchmarking of different search scenarios within Solid pods. Part of this process involves the development of tools and frameworks dedicated to the testing and validation of the proposed distributed search system, ensuring its scalability and effectiveness.

To this end, the ESPRESSO framework is built on several design principles that are summarized below as follows:

1. *Ensuring data sovereignty*. ESPRESSO should build and maintain *distributed* indexes inside Solid pods rather than centralized ones, and maintain Metadata indexes to describe information on index access that can be used by search optimization algorithms.
2. *Respecting access control*. Enabling distributed keyword-based search and also supporting decentralized SPARQL querying while ensuring access control rights.
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.
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.
5. *Privacy over efficiency*. Use of metadata for guiding query routing, and other search optimization techniques, should not compromise data privacy.

Table 1 presents a summary of the mentioned design principles behind ESPRESSO. It links each principle to the specific challenges it is intended to address and details the set of solutions provided by each principle to tackle these challenges.

For instance, the principle of ensuring data sovereignty addresses challenges C.1 and C.3 by securely storing data in user-controlled pods, building and maintaining distributed (Meta)indexes, and minimizing data disclosure during searches. Respecting access control, also addressing challenges C.1 and C.3, involves integrating data with access control rights into the indexes and ensuring that distributed queries access indexes rather than the data directly. Scalability, targeting challenge C.2, is achieved through diverse large-scale search scenarios suitable for various applications. Decentralization, which addresses challenge C.4, is facilitated by a federated overlay network for query routing and propagation. Finally, the principle of privacy over efficiency, covering challenges C.1 and C.4, mandates that search optimizations must preserve data privacy [25].

3.1 ESPRESSO Architecture Overview

Figure 4 illustrates the conceptual design of the ESPRESSO framework, highlighting its fundamental components that contribute to a comprehensive understanding of search enhancement across pods. We outline the functionality of the key components within the ESPRESSO framework, which are structured to facilitate efficient search across a large number of Solid-compliant servers. This is followed by a detailed description of our current prototype implementation for each component.

(1) *Pod indexing app – Brewmaster* ESPRESSO includes a Solid pod indexing application known as *Brewmaster* which creates and maintains local indexing files for the data inside pods. The created indexes reflect not only the pods' data but also access control assertions about it. Also, the Brewmaster app maintains relevant information about the created local indexes in a *MetaIndex* file for the Solid server (in a special pod called *ESPRESSO Pod*). The MetaIndex stores information

Table 1 Summary of ESPRESSO design principles mapping challenges along with their solutions

Design principle	Challenges	ESPRESSO solutions
Ensuring data sovereignty	C.1, C.3	Data is securely stored in pods controlled by users. Building and maintaining distributed (Meta)indexes. Indexes minimize data disclosure during searches
Respecting access control	C.1, C.3	Indexes integrate data with access control rights. Distributed queries access indexes, not data directly
Scalability	C.2	Diverse large-scale search scenarios for various applications
Decentralization	C.4	Federated overlay network for query routing & propagation
Privacy over efficiency	C.1, C.4	Search optimizations must preserve data privacy

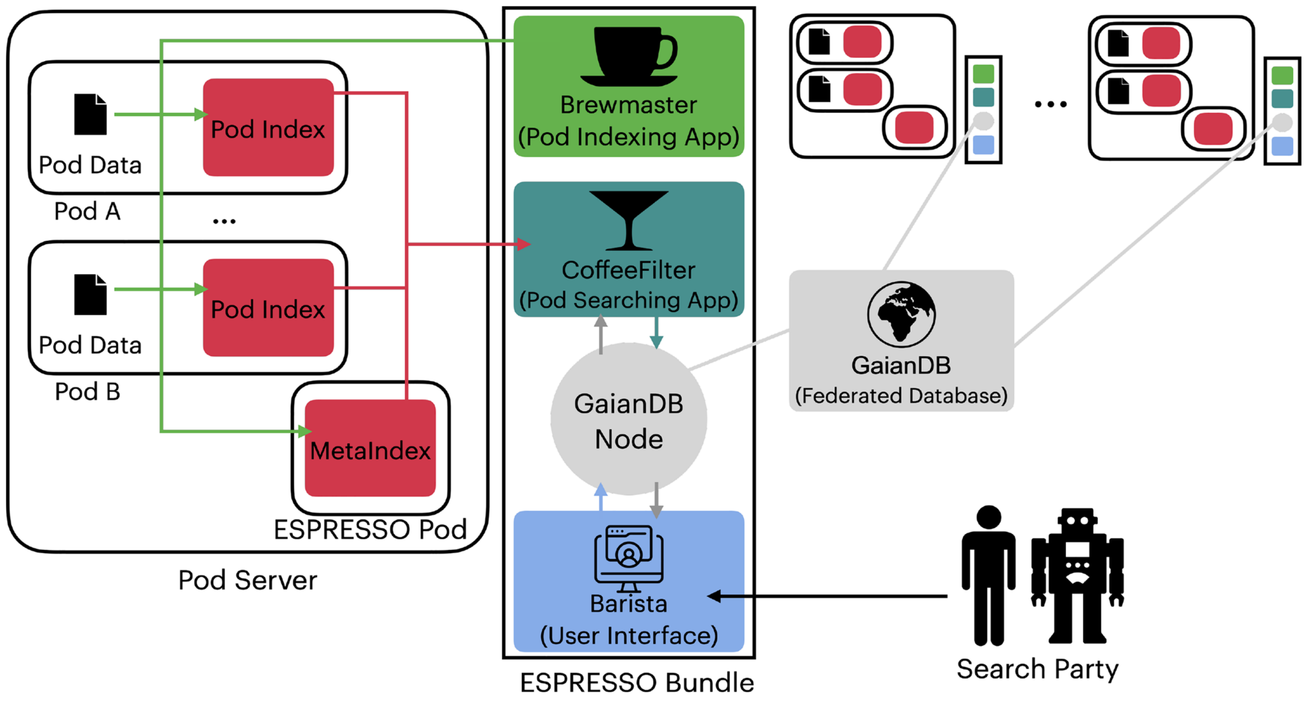


Fig. 4 ESPRESSO framework architecture

such as indexing files' addresses and other metadata that is used for search optimization and filtering purposes. To ensure privacy, the ESPRESSO system ensures a pod's index does not leave the pod when a search party performs search operations.

(2) *User interface – Barista Barista* acts as the *User Interface* (UI) application for the ESPRESSO framework, streamlining the search process for end users. Upon receiving a search query along with valid login credentials, Barista employs a *QueryBuilder* interface to formulate and prepare the query for execution. It facilitates searches not only by human users but also by applications and services, which can either use their own WebIDs or utilize a user's WebID through delegation, as described in [9]. Once the query is constructed, Barista forwards it to a federated database (DB) node. This node then disseminates the query to other federated DB nodes within the overlay network. The gathered results are aggregated across these nodes and relayed back to the Barista interface via the federated DB node that initiated the query. Barista then displays the results in a user-friendly and comprehensible format. In the initial prototype of ESPRESSO, the search query is converted into a GaianDB query, specifically an *SQL* query. This conversion occurs in *step (1)* "Query Preparation", as depicted in Fig. 5.

(3) *Overlay network*: The ESPRESSO framework employs an overlay network to disseminate end-user queries to pertinent data resources located on various Solid servers. In this setup, every Solid server within the ESPRESSO system is linked to a federated DB node within the overlay network. The primary goal of this query propagation strategy is to conduct user queries efficiently while reducing the amount of data exchanged between different data sources.

The initial prototype of ESPRESSO incorporates a data federation platform named *GaianDB*[5] to facilitate the propagation of user queries and the collection of search results from Solid servers, as referenced in [26] and depicted in *step (2)* in Fig. 5. GaianDB is a dynamic and distributed federated database system that integrates concepts from extensive distributed databases, database federation, and network topology. It functions as a *peer-to-peer* (P2P) overlay network compatible with a broad array of devices, focusing on optimizing query propagation and managing endpoint access limitations.

GaianDB offers the capability for data to be stored and retrieved from multiple locations, embodying the principle of *Store Locally and Query Anywhere*. This approach enhances the system's resilience and mitigates the risk of data loss. Furthermore, GaianDB facilitates the propagation of queries across its nodes,

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

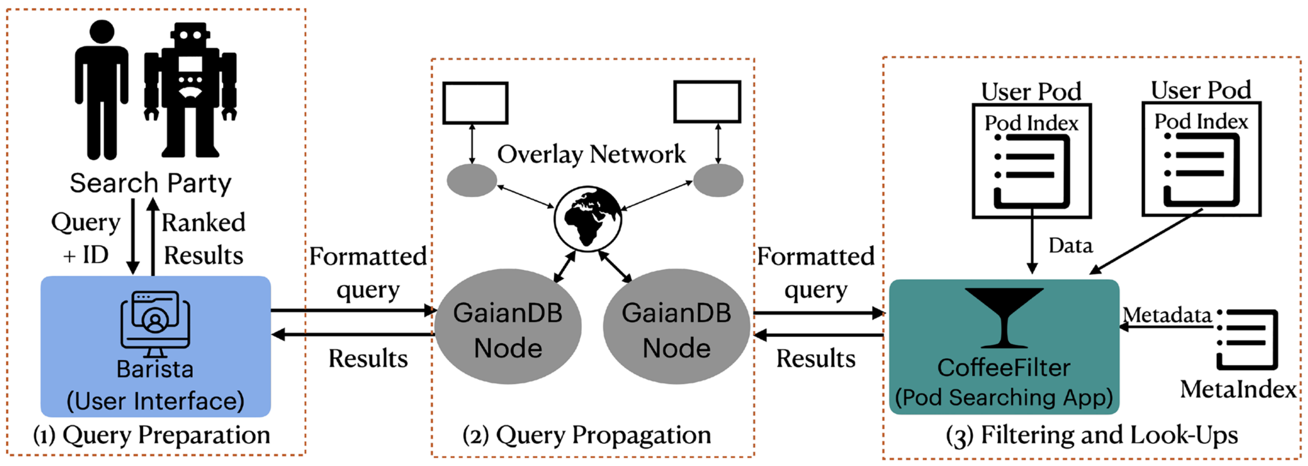
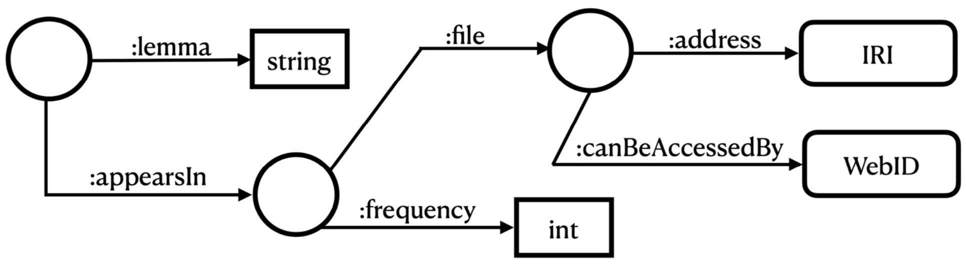


Fig. 5 Search, index look-ups, and query propagation pipeline in ESPRESSO

Fig. 6 RDF pod index format



ensuring that result sets are obtained via the *shortest paths* from nodes capable of fulfilling the query. As a result, a query can be initiated at any GaianDB node and will be seamlessly distributed to other nodes in the network. GaianDB then undertakes the task of compiling these results, sending the consolidated outcome back to the node where the query originated. For the purpose of optimizing search efficiency, GaianDB also incorporates caching mechanisms for storing preprocessed results.

In the ESPRESSO prototype, we have developed a *Solid-to-GaianDB* connector component. This component creates a link between a GaianDB node and a Solid server. The connector is responsible for storing data retrieved from the Solid server in a CSV file format. Following this, it creates a *logical table*, and maps the data from the CSV file onto this logical table. The Solid-to-GaianDB connector leverages the concept of logical tables, as detailed in [26], to construct an abstract federation layer within the GaianDB network. This layer encompasses all data sources from pods across various Solid servers. Consequently, users can access all relevant data spread across different pods in various Solid servers through GaianDB nodes, in accordance with their access permissions.

(4) *Pod searching app – CoffeeFilter* conducts local search operations on the pods of every Solid server on which it is installed. When it receives a query from a federated DB node, *CoffeeFilter* consults the MetaIndex, which has been generated by the *Brewmaster* application. This provides the locations of the pod indexes along with pertinent metadata. Subsequently, CoffeeFilter executes a search within the relevant pod indexes and forwards the search results back to the federated DB node. The details of the search operation process are described in Sect. 3.2.

The *ESPRESSO Bundle*, comprising the components illustrated in Fig. 4, is distributed as an open-source software suite[6]. For its functionality, the Pod Indexer Application (*Brewmaster*) requires access to the content of each individual pod, provided by the pod owners, in order to index the text files. Subsequently, the Pod Searching Application (*CoffeeFilter*) utilizes the generated indexes for searching purposes.

[6] *ESPRESSO Search System*: https://github.com/espressogroup/ESPRESSO.

3.2 Indexing & Search Over Pods

In our preliminary investigation of implementing an inverted index for pod data, a *single* inverted index is created for the pods' data [17] on a Solid server. The created index is materialized as an RDF graph and follows an indexing template. Figure 6 shows an example of an Index Template for *keyword-based* search and Fig. 7 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 `IRIs`, or *Internationalized Resource Identifiers*), 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://solid-server/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`.

That indexing technique has several drawbacks: (1) The Pod Searching App has to access it every time a query is received. This leads to increased search times, especially if the size of the pod index is large. Also, (2) the Pod Searching App requires GETting/downloading the index file to undertake search operations against it. This violates privacy because it exposes more data than is necessary to answer a query.

Thus, we developed an enhanced alternative indexing scheme that takes into account the limitations of the previous indexing scheme regarding index size; moreover, it places a strong emphasis on privacy and considers different users' and applications' access rights [27]. This new indexing scheme considers the Solid server-side search processing limitations [8] and builds the indexing files to be accessed according to Solid *LDP* principles via simple HTTP GET requests [15]. Also, it dedicates high attention to optimizing the size and the number of indexing files required for efficient search operations while respecting users' data sovereignty. Below, we describe how the proposed indexing scheme allows Solid applications to search for data across pods to which they have access, without having to extract and aggregate such data into a central repository for search purposes. Additionally, the current ESPRESSO indexing method limits data exposure to only what is necessary for the query and within access permissions, thus enhancing the system's data privacy and reduce security risks.

Figure 8 shows how the Brewmaster Pod Indexing Application constructs local indexes using the new indexing schema. Consider a scenario with a Solid server at `https://server/` and a corresponding pod at `https://server/pod/`. In this setup, the Brewmaster Pod Indexing Application assigns a unique *fileID* to each pod file, following a simple naming pattern such as *F1, F2, F3*,.... It then establishes a directory for indexing purposes, say `https://server/pod/espressoindex/`. This URL is recorded in the MetaIndex within the *ESPRESSO Pod* on the same server. The indexing application arranges the index files in the pod's indexing directory using a

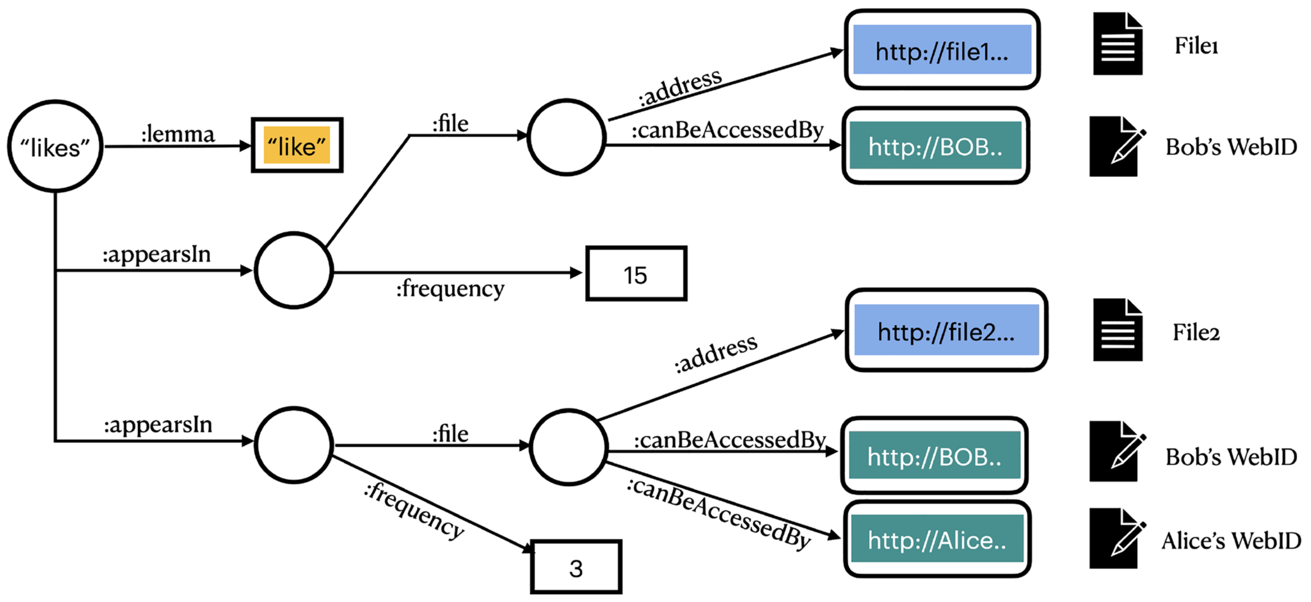


Fig. 7 An example of an RDF pod index

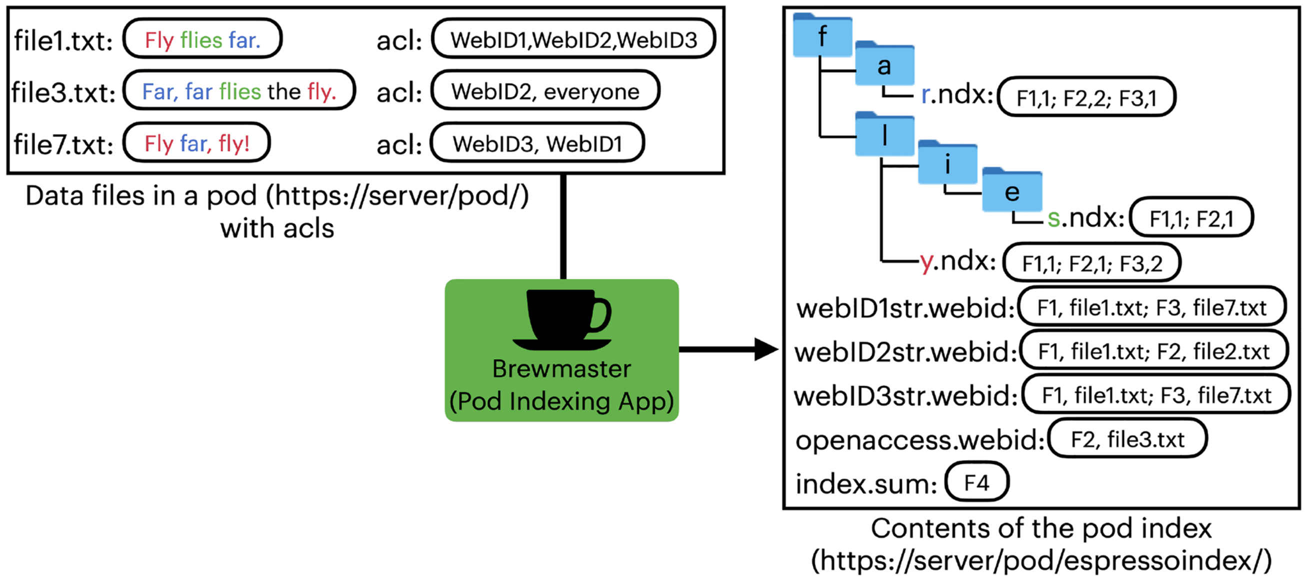

file1.txt: Fly flies far. acl: WebID1,WebID2,WebID3
file3.txt: Far, far flies the fly. acl: WebID2, everyone
file7.txt: Fly far, fly! acl: WebID3, WebID1
Data files in a pod (https://server/pod/) with acls
Brewmaster (Pod Indexing App)
f
a
r.ndx: F1,1; F2,2; F3,1
l
i
e
s.ndx: F1,1; F2,1
y.ndx: F1,1; F2,1; F3,2
webID1str.webid: F1, file1.txt; F3, file7.txt
webID2str.webid: F1, file1.txt; F2, file2.txt
webID3str.webid: F1, file1.txt; F3, file7.txt
openaccess.webid: F2, file3.txt
index.sum: F4
Contents of the pod index (https://server/pod/espressoindex/)

Fig. 8 Pod index structure. On the left, Pod files with corresponding ACLs. On the right, generated Pod indexing files

trie-like structure to optimize index file retrieval, a method expounded upon in [28].

For every keyword in the indexed text files, such as '*fly*', the indexing application creates a distinct index file at `https://server/pod/espressoindex/f/l/y.ndx`, containing the *fileIDs* of all occurrences of '*fly*' and their respective frequency. Moreover, for each WebID with read permissions, e.g. *WebID1*, a `WebID1str.webid` file is generated in the index. This file lists the *fileIDs* and actual names (e.g., `file1.txt`) of accessible files for that WebID. A file named `openaccess.webid` is also created for files accessible to all, containing their fileIDs and names. Lastly, an `index.sum` file is produced for administrative purposes, detailing information such as the list of deleted fileIDs and the subsequent available fileID.

This enhanced indexing scheme improves efficiency by generating concise, inverted micro-indexes for each keyword in text files. It also compiles a list of WebIDs that have authorization to access files containing these keywords. This method accelerates the retrieval of specific keyword queries while ensuring controlled access to pertinent information only.

Search with old indexing scheme: In the initial version of the ESPRESSO prototype, *CoffeeFilter* is structured into two distinct components: a `RESTful` (*representational state transfer*) `API`[7] (*application programming interface*) and a query processing segment. The API component is responsible for receiving queries from a federated DB node (a GaianDB node), verifying them, and then directing them to the query processing segment. This component also takes charge of reshaping the search results and sending them back to the federated DB node. The query processing segment accesses the MetaIndex, converts the user query into a SPARQL query, and runs this query on the relevant pod indexes, as illustrated in *step (3)* in Fig. 5. To execute queries against the indexes, CoffeeFilter utilizes the *Comunica*[8] search engine library, a flexible JavaScript library for SPARQL and GraphQL, adept at querying decentralized knowledge graphs on the Web. Ultimately, the Pod Searching application returns a *JavaScript Object Notation* (`JSON`)-formatted ranked list of search results to the federated DB node.

Search with new indexing scheme: Fig. 9 illustrates the keyword search mechanism employed by the Pod Searching (CoffeeFilter) App in the current ESPRESSO system. Consider a scenario where the CoffeeFilter Pod Searching App receives a query for the keyword '*fly*', originating from a certain *WebID*. The initial step involves fetching the *MetaIndex* from the *ESPRESSO Pod* to gather all pod index URLs. For each pod index URL, denoted as `podindex`, the CoffeeFilter executes a series of operations:

1. GET the file `podindex/f/l/y.ndx` and obtain the list of fileIDs with the corresponding frequencies.
2. GET the files `podindex/openaccess.webid` and `podindex/WebIDstr.webid` and combine them to get the list of all accessible files.

[7] https://aws.amazon.com/what-is/restful-api/.

[8] https://comunica.github.io/comunica/.

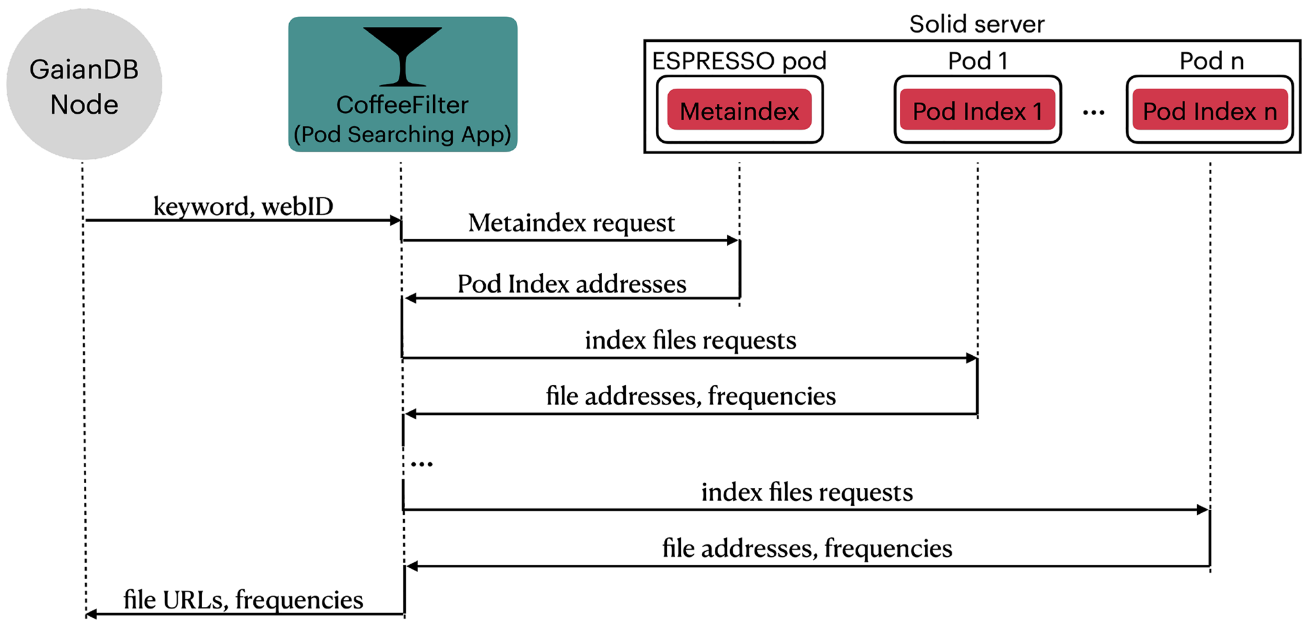


Fig. 9 Search process in the CoffeeFilter pods search application

Table 2 Summary of the experimental parameters of the motivating scenario

Parameter	Description
Number of solid servers	50 Servers
Number of pods/server	~9500 Pods
Number of files/pod	1 to ~350 files (*Pareto* distribution, $\alpha = 1$)
Data size per pod	~5KB to ~750KB (based on number of files)
Access control	10% of pods grant access to their files (~850 pods)

3. Intersect the two lists of fileIDs and obtain the files' URLs by concatenating the file names with the base URL for the pod.
4. Return the list of URLs with the corresponding frequencies.

Upon completing these steps for all pods on the server, the CoffeeFilter Pod Searching App aggregates the results. These are then relayed back to the overlay network.

4 Evaluation & Experimental Setup

This section outlines our experimental methodology for load testing of the current prototype implementation of the ESPRESSO framework. The goal is to assess the viability of the decentralized search in our ESPRESSO prototype by exploring its performance in real-word settings like the aforementioned motivating scenario in Sect. 2.2.

4.1 Experimental Environment & Setup

Our experiments extend previous recent decentralized search systems studies that utilized a single Solid server for experimentation [16, 29]. We established an initial cluster comprising 50 Solid servers designed to exemplify a network of general health practitioners in a metropolitan area, where each practitioner utilizes a Solid server. On these servers, patients' medical records can be securely maintained in individual pods. Each Solid server is installed on a *Virtual Machine* (`VM`); each VM is outfitted with the *Red Hat Enterprise Linux* operating system, version 8.7, and is powered by 2.4GHz processors with 8GB of RAM. Moreover, each VM is equipped with a high-speed 125GB storage drive and operates a single instance of *Community Solid Server* (version 6.0) with a file-based storage backend, operating in *multithreaded* mode utilizing multiple workers (scaling to the number of processor cores) .[9] Due to current computational resource constraints, one of the VMs is equipped with an *8-core* processor, whereas the remaining 49 VMs utilize only *dual-core* processors. For search operations across Solid Servers, the Pod Searching App, *CoffeeFilter*, employs a *Node-js Axios* library for performing HTTP requests to index files in the pods. Additionally, our experiments utilized a customized version of GaianDB (version 2.1.8), i.e., plugged in with our developed *Solid-to-GaianDB* connector (details of the connector in Sect. 3.1).

For our experiment, we used the following procedure and set of experimental parameters (see summary of parameters in Table 2):

1. *Number of servers and pods*: With each server representing a *GP practice*, our 50 servers represent an area of considerable size (e.g. *Cornwall* has 57 GP practices[10]). The average number of patients in a GP practice in the UK is ~9500 people[11], so on each server, we store ~9500 pods.
2. *Dataset*: The pods are populated with sample medical history files with texts taken from the *Medical Transcriptions Samples* [12] text-based dataset. This dataset includes medical transcription samples from different medical specialties.
3. *Data preparation*: The original dataset is in CSV. So, we developed a simple script that extracts the transcription text from each medical visit record into a separate text file. The size of each file is approximately 5KB.
4. *Selection of search keywords*: We selected *four* keywords for search: a commonly used keyword appearing in approximately 10% of the files, a moderately frequent word occurring in about 2% of the files, and two seldom-used keywords found in roughly 0.2% and 0.1% of the files, respectively.
5. *Data distribution logic*: The literature indicates that the rate of patients' participation in medical trials stands at 10%, c.f. the global study by Anderson et. al. [30]. Thus, we aimed to ensure that around 10% of the pods grant access to their files through the selected WebID for this experiment. To this end, on each Solid server, we created a first group of pods – normally distributed with a mean of 8500 and standard deviation of 85, representing the pods of people not consenting to the use of their data, and a second group of pods – normally distributed with a mean of 850 and standard deviation of 162.5, representing the people who do.

   In each pod is placed a random sample of files selected from the dataset, which comprises medical visit records. Research from [31] and [32] indicates that, on average, individuals visit their GP about *once* per year. However, this frequency is primarily due to a subset of frequent visitors. To accurately represent this distribution in our study, we have chosen to employ a *Pareto distribution* with a parameter of $\alpha = 1$ for allocating the number of files in each pod.
6. *Access control*: We made each file accessible to one WebID from a list of 250 to simulate the doctors having access to the patients' files. We also made all the files in the second group of pods open to a special WebID, that represents Alice's WebID, from the scenario described in Sect. 2.2, which we will use for the search.
7. *Index construction*: We created the pod indexes based on the newly established indexing scheme (see Sect. 3.2). Each pod's files and indexes were compressed and stored in a specific directory assigned to their corresponding Solid server.
8. *Creation of solid pods and metaIndexes*: For each experiment, we set up the designated pods on the selected servers and implemented MetaIndexes in the *ESPRESSO Pod* located on each server (refer to Fig. 4).
9. *Deployment of data and indexes*: The compressed files were transferred to the respective VMs hosting the servers. These files were then decompressed into the pods following the previously described logical structure.
10. *Search execution*: In each experiment, we performed search queries for the four pre-selected keywords across the 50 Solid servers. To reduce any *warm-up* effects, each query was executed *five* times, disregarding the first run. The average time was then calculated from the subsequent *four* runs.

Experiments & evaluation: To evaluate the system's performance, we focused on the following parameters: the search time — how long the pod searching app takes to perform a query on its server, and the routing time — the time taken by the overlay network to distribute the query and collect the results from all the servers. To measure the search time we use the time taken by the pod searching app on the 8-core processor VM (the fast machine). To measure the routing time we take the difference between the

[9] Community Solid Server: https://github.com/CommunitySolidServer/CommunitySolidServer.

[10] Data taken fromhttps://cios.icb.nhs.uk/health/primary-care/.

[11] Data taken from https://www.gponline.com/fifth-gp-practices-closed-merged-nhs-england-formed/article/1790429.

[12] https://www.johnsnowlabs.com/marketplace/medical-transcription-samples/.

total execution time of the whole search operation and the time taken by the pod searching app of the slowest VM.

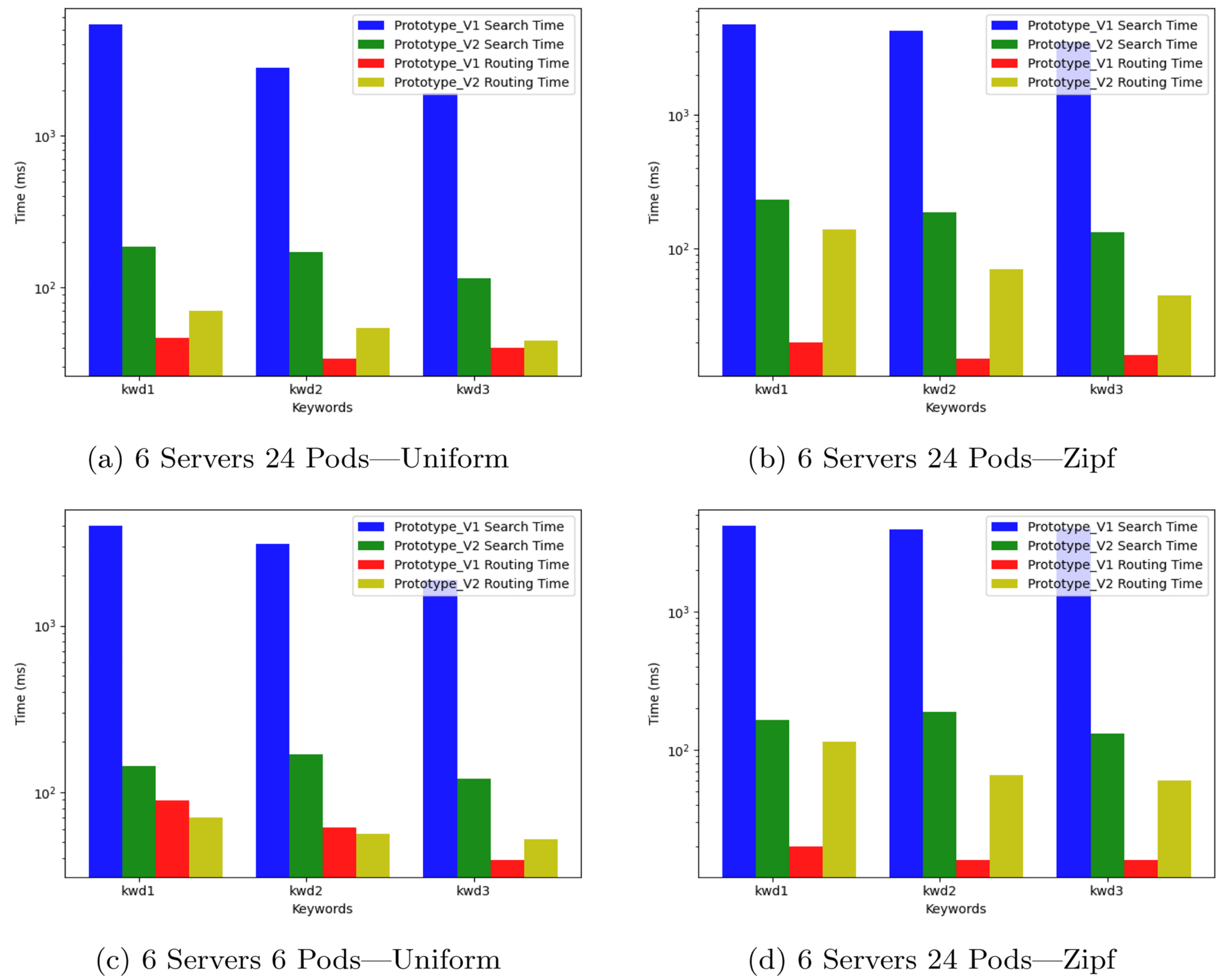
Table 3 Experiment results for search on scenario's dataset with the chosen four keywords. Search & Routing runtimes are in ms. Number of rows represent search results rows number

Keyword	Search time	Routing time	Number of rows
Keyword 1	9053	1560	32,691
Keyword 2	8203	393	6640
Keyword 3	8114	151	625
Keyword 4	8186	138	319

5 Experimental Results and Discussion

In this section, we present and discuss the results of our experiments. We report the results of performing keyword search operations on a large number of decentralized personal datastores located in multiple different Solid servers, in order to check ESPRESSO's viability in a real-world setting.

The preliminary experimental results in [17], stress testing the initial ESPRESSO prototype, showed that most of the total search time is taken by the local pod searching app that fetches results at each Solid server. The extensive experiment conducted in this paper on our motivating scenario's dataset still confirms that initial finding (see *Search Time*



(a) 6 Servers 24 Pods—Uniform

(b) 6 Servers 24 Pods—Zipf

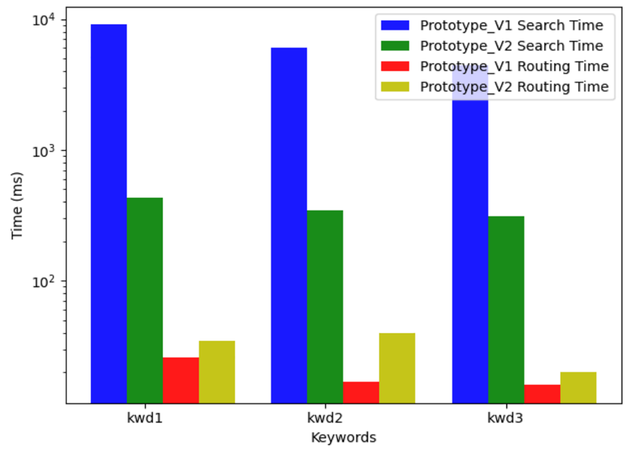
(c) 6 Servers 6 Pods—Uniform

(d) 6 Servers 24 Pods—Zipf

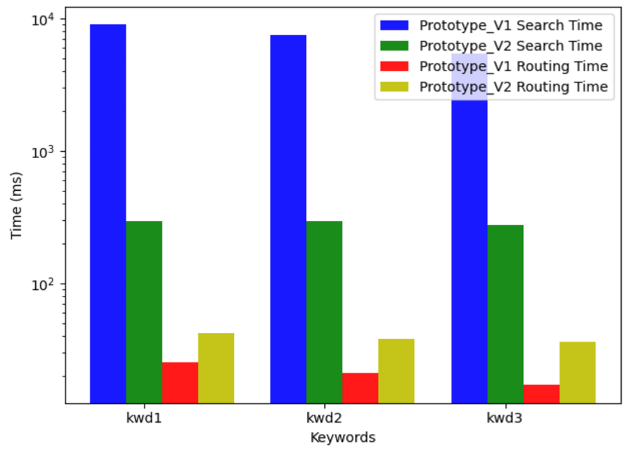
Fig. 10 The impact of the new indexing scheme (Part 1— multiple-servers experiments), comparing the performance of ESPRESSO prototypes (Version 1 (V1) in [17] and the current prototype Version 2 (V2)) across various settings (number of servers and pods), and different data distribution (`Uniform` and `Zipf`), with three various frequency keywords (`Kwds`). The *Search* and *Routing* Times are in ms. A logarithmic scale is set for the y-axis

results in Table 3). The *Routing Time* results in Table 3 also confirm the initial finding that the time taken by *GaianDB* to propagate and route the query and aggregate the search results is negligible by comparison. This also unveils the consistent stable performance of the GaianDB overlay network for routing the queries and aggregating the results even with scaling the number of Solid servers (from 6 Solid servers in [17] to 50 servers in our current evaluation). Also in this regard, the keyword's frequency is shown to have a notable impact on the search run times in all the experiments, especially on the Routing Times. Indeed, GaianDB has to aggregate and return back all the search results from all the Solid server nodes to the node that issued the query – the more frequent the keyword, the longer it takes to retrieve and aggregate all the search results.

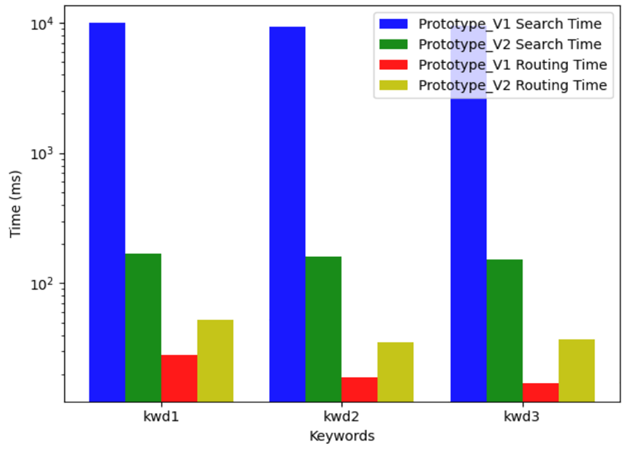
To show the impact of the new enhanced indexing scheme, we replicated the same setup as in [17] (i.e., 6 Solid servers, and up to 24 pods) and tested the new indexing scheme using the same dataset as in that paper. Figure 10a–d, with multiple-server experiments, and Fig. 11a–c, with single-server experiments, show how the new indexing significantly enhances the performance of the decentralized search in ESPRESSO. It shows how the search and routing time dramatically improved with the new indexing schema. The enhanced indexing scheme we developed and implemented, as detailed in Sect. 3.2, has also a notable impact on the evaluation of the current experiment. Scaling up to our real-world motivating scenario, the performance of the current prototype, with the enhanced indexing scheme, with 50 servers and approximately ~475,000 pods in total, notably



(a) 1 Server 24 Pods—Uniform



(b) 1 Server 24 Pods—Zipf



(c) 1 Server 1 Pod—Uniform

Fig. 11 The impact of the new indexing scheme (Part 2— single-server experiments), comparing the performance of ESPRESSO prototypes (Version 1 (V1) in [17] and the current prototype Version 2 (V2)) across various settings (number of servers and pods), and different data distribution (`Uniform` and `Zipf`), with three various frequency keywords (`Kwds`). The *Search* and *Routing* Times are in ms. A logarithmic scale is set for the y-axis

outperforms the performance of the initial search prototype experimenting with only 6 Solid servers and 24 pods in total. Besides the reasons mentioned in Sect. 3.2 of how the current indexing scheme has enhanced performance, an additional factor in the initial pod search application being too slow was that it employed the *Communica* engine, which requires loading the RDF index files into memory in order to execute *SPARQL* queries on them. Thus, the performance of the initial stress testing in [17] mainly rested on the size of the *single RDF-based* index file in each pod whereas the current Pod Searching app (CoffeeFilter) requires a few HTTP GET requests against the indexing files relating to the queried keyword.

It is worth mentioning that in our work [33], we perform other extensive experiments to investigate the individual impact of several controllable parameters on the decentralized search performance. In particular, we investigate a range of factors characterizing data distribution across Solid servers, such as the *number of servers*, the *number of pods*, the *number and size of the files*, and how many of those files the search party has access to. We report the details and results of the experiments in that paper.

6 Limitations & Challenges Ahead

Index structure refinement Currently, the index structure is composed of smaller files due to a high frequency of uncommon terms, resulting in considerable disk space usage. To mitigate this, we suggest exploring alternative file system partitioning strategies or refining the index structure. An optimized index structure would ideally maintain the privacy-preserving aspects of the current scheme while being more space-efficient. Enhancing the indexing to include more nuanced relevance measures beyond mere word frequency could also improve the quality of search results.

Advancing search capabilities Expanding ESPRESSO's search functionalities beyond simple keyword queries is a significant undertaking. This entails incorporating *Natural Language Processing* (`NLP`) operations, multi-word queries, and structured queries (e.g., SPARQL) specifically tailored for RDF data in Solid pods. Investigating novel indexing structures suitable for RDF data, as well as mechanisms that consider RDF data access control within the ESPRESSO framework, is crucial. Such advancements will enable ESPRESSO to be compared more directly with other decentralized search systems, potentially leading to a more robust and versatile search framework.

Optimized query handling and propagation The existing query propagation mechanism within the GaianDB network relies on the default "query flooding" technique. We anticipate that the adoption of advanced routing algorithms and more sophisticated query propagation methods can significantly enhance search efficacy across the GaianDB federated network. The development of additional metadata could further optimize this process by directly routing queries to pertinent federated nodes, thereby alleviating the load on the overlay network.

7 Related Work

The decentralized search problem has been tackled from various perspectives and research areas, including distributed databases [34], P2P search and query routing [35], and SPARQL distributed querying and link-following [36–38].

A substantial body of research has delved into developing mechanisms of empowering privacy ind distributed database systems [25, 34, 39]. Distributed search methodologies within federated database systems across various autonomous entities [5]. These studies have introduced distributed indexing approaches to facilitate searches across disparate databases [40]. However, these methods often rely on the availability of query endpoints and results caching, assumptions that may not hold in decentralized environments.

The realm of P2P data management systems has provided insights into decentralized search processes [41]. Notably, protocols such as the *InterPlanetary File System* (`IPFS`) utilize *Distributed Hash Tables* (`DHTs`) for keyword-based search [42]. However, these approaches often overlook the nuances of varying access controls and the complexities of managing distributed indexes and query endpoints.

In the Semantic Web and Linked Data contexts, decentralized SPARQL querying presents its own set of challenges [43, 44]. It necessitates the establishment of endpoint metadata for each search entity and the implementation of stringent access controls and caching constraints during SPARQL link-following [15, 37]. Adhering to these requirements significantly escalates storage, network, and computational demands.

Thus the task of performing decentralized searches across Solid pods introduces complexities not fully addressed in the existing literature. The variability in access privileges among search entities and the constraints imposed by caching limitations on search result dissemination are crucial considerations [12, 17]. While recent studies have begun to explore querying RDF data within Solid pods from a decentralized Knowledge Graphs [45, 46] perspective, using approaches like *Link Traversal Query Processing* (`LTQP`), they often fall short in addressing access control [16] or decentralized indexing [15].

Therefore, the ESPRESSO project represents an initial effort in enabling large-scale decentralized keyword search over Solid pods. It emphasizes data sovereignty and compliance with user access controls through decentralized indexes on pod contents. Our focus is initially on keyword-based search as a foundational step towards understanding the requirements and performance factors needed for advanced structured distributed queries, such as SPARQL, or decentralized keyword search in structured/semi-structured personal datastores [47].

8 Conclusion

ESPRESSO provides search functionalities for personal online data repositories while adhering to users' access control requirements and maintaining data sovereignty. The architecture of the system minimizes the revelation of irrelevant data during searches. Our testing confirms that ESPRESSO is an effective and scalable option for keyword searches, making it suitable for diverse practical situations. The system demonstrates significant potential, especially in handling large volumes of sensitive information that require strict access controls, such as health data [48]. This is in stark contrast to traditional search methods that often rely on data centralization or centralized indexing, which heightens the risk of data breaches and unauthorized access by third parties.

In future work, we aim to tackle the challenges articulated in Sect. 6 and investigate further ESPRESSO's practicality in fields that benefit from decentralized, privacy-centric search solutions. This includes performing extensive benchmarking experiments (using frameworks like [49–51]), testing various search scenarios and styles (e.g., exhaustive and top-k for both keyword-based and SPARQL queries), and comparing different query propagation techniques, and exploring the feasibility of performing federated learning on Solid servers empowered with search [52]. ESPRESSO shows potential in various applications, including contact tracing where user location data is securely stored in personal data storage spaces. This method ensures privacy while allowing for alerts to be sent to individuals who have been in locations shared with those diagnosed with diseases like *COVID-19*. We also aim to enable ESPRESSO in other scenarios such as users recommendations, Linked Data streams [53], and neighbor-aware review predictions [54, 55].

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s41019-024-00263-w) contains supplementary material, which is available to authorized users.

Acknowledgements The authors of this work like to thank *Lance Draper* and Dr. *Ruben Taelman* for their support in this work.

Author Contributions Mohamed Ragab, the corresponding author, wrote the main manuscript text, conducted all sets of experiments, and analyzed the results. Yury Savateev prepared figures, designed the motivating scenario's initial design, and helped with the experiments. The rest of the authors reviewed the manuscript and gave valuable feedback, also helped with all discussions that led to the manuscript.

Funding This work was supported by the EPSRC Espresso Project (EP/W025868/1 and EP/W024659/1) and the NIHR Southampton Biomedical Research Centre (NIHR203319).

Availability of Data and Materials In this paper, we use a anonymized publicly available medical transcriptions dataset. To ensure integrity, we make sure that the dataset does not include any personal information (names, contacts, etc), but generated medical GP visits prescriptions, and we keep the data on our local servers with no sharing to the public, we share only performance search results in our paper. - Dataset Source: This data was scraped from John Snow Labs; `MTSample.com` can be publicly found here: https://www.johnsnowlabs.com/marketplace/medical-transcription-samples/. We also aim to provide the dataset on our GitHub repository. - Source License Requirements: N/A - Source Citation: N/A - Expected update frequency: Annual.

Declarations

Conflict of interest No conflict of interest.

References

1. Abiteboul S, André B, Kaplan D (2015) Managing your digital life. Commun ACM 58(5):32–35
2. Ge Y-F, Wang H, Cao J, Zhang Y (2022) An information-driven genetic algorithm for privacy-preserving data publishing. In: International Conference on Web Information Systems Engineering, 340–354. Springer
3. Kollnig K, Binns R, Van Kleek M, Lyngs U, Zhao J, Tinsman C, Shadbolt N (2021) Before and after gdpr: Tracking in mobile apps. arXiv preprint arXiv:2112.11117
4. Mazeh I, Shmueli E (2020) A personal data store approach for recommender systems: enhancing privacy without sacrificing accuracy. Expert Syst Appl 139:112858
5. Kahle B (2015) Locking the web open: a call for a decentralized web. Brewster Kahle's blog
6. Berners-Lee T (2010) Long live the web. Sci Am 303(6):80–85
7. Sambra AV, Mansour E, Hawke S, Zereba M, Greco N, Ghanem A, Zagidulin D, Aboulnaga A, Berners-Lee T (2016) Solid: a platform for decentralized social applications based on linked data. MIT CSAIL & Qatar Computing Research Institute, Tech. Rep
8. Dedecker R, Slabbinck W, Hochstenbach P, Colpaert P, Verborgh R (2022) What's in a pod?–a knowledge graph interpretation for the solid ecosystem
9. Sambra A, Guy A, Capadisli S, Greco N (2016) Building decentralized applications for the social web. In: Proceedings of the 25th international conference companion on world wide web, pp. 1033–1034
10. Zhao R, Goel N, Agrawal N, Zhao J, Stein J, Verborgh R, Binns R, Berners-Lee T, Shadbolt N (2023) Libertas:

privacy-preserving computation for decentralised personal data stores. arXiv preprint arXiv:2309.16365
11. Troncoso C, Payer M, Hubaux J-P, Salathé M, Larus J, Bugnion E, Lueks W, Stadler T, Pyrgelis A, Antonioli D, et al. (2020) Decentralized privacy-preserving proximity tracing. arXiv preprint arXiv:2005.12273
12. Tiropanis T, Poulovassilis A, Chapman A, Roussos G (2021) Search in a redecentralised web. In: Computer science conference proceedings: 12th international conference on internet engineering; Web Services (InWeS 2021)
13. Morris M, Teevan J (2009) Collaborative web search: who, what, where, when, and why. https://doi.org/10.2200/S00230ED1V01Y200912ICR014
14. Mansour E, Sambra AV, Hawke S, Zereba M, Capadisli S, Ghanem A, Aboulnaga A, Berners-Lee T (2016) A demonstration of the solid platform for social web applications. In: Proceedings of the 25th international conference companion on world wide web, 223–226
15. Taelman R, Verborgh R (2023) Link traversal query processing over decentralized environments with structural assumptions. In: Payne TR, Presutti V, Qi G, Poveda-Villalón M, Stoilos G, Hollink L, Kaoudi Z, Cheng G, Li J (eds) The semantic web - ISWC 2023. Springer, Cham, pp 3–22
16. Vandenbrande M, Jakubowski M, Bonte P, Buelens B, Ongenae F, Van den Bussche J (2023) POD-QUERY: schema mapping and query rewriting for solid pods, p. 5
17. Ragab M, Savateev Y, Moosaei R, Tiropanis T, Poulovassilis A, Chapman A, Roussos G (2023) ESPRESSO: a framework for empowering search on decentralized web. In: Zhang, F., Wang, H., Barhamgi, M., Chen, L., Zhou, R. (eds.) Web information systems engineering - WISE 2023 - 24th international conference, Melbourne, VIC, Australia, October 25-27, 2023, Proceedings. Lecture notes in computer science, vol. 14306, pp. 360–375. Springer. https://doi.org/10.1007/978-981-99-7254-8_28
18. Ragab M, Savateev Y, Oliver H, Moosaei R, Tiropanis T, Poulovassilis A, Chapman A, Roussos G (2024) A demonstration of decentralized search over solid personal online datastores. In: Companion proceedings of the ACM on web conference 2024, 1055–1058
19. Raza A, Han K, Hwang SO (2020) A framework for privacy preserving, distributed search engine using topology of dlt and onion routing. IEEE Access 8:43001–43012
20. Mayeur C, Hoof W (2021) Citizens' conceptions of the genome: related values and practical implications in a citizen forum on the use of genomic information. Health Expect 24(2):468–477
21. Luger E, Moran S, Rodden T (2013) Consent for all: revealing the hidden complexity of terms and conditions. In: Proceedings of the SIGCHI conference on human factors in computing systems. CHI '13, pp. 2687–2696. Association for computing machinery, New York, NY, USA. https://doi.org/10.1145/2470654.2481371
22. Das S (2023) Nhs data breach: trusts shared patient details with facebook without consent. The Observer
23. Hudig AI, Singh J, Binns R, Cloete R, Haddadi H, Mandalari AM (2023) Transparency in the consumer internet of things: data flows and data rights. Information Commissioner's Office, Technical report
24. Iacobucci G (2023) Data privacy: Gp surgery withdraws from kidney screening pilot after patients voice concerns. BMJ 380:157. https://doi.org/10.1136/bmj.p157
25. Ge Y-F, Bertino E, Wang H, Cao J, Zhang Y (2023) Distributed cooperative coevolution of data publishing privacy and transparency. ACM Trans Knowl Discov Data 18(1):1–23
26. Bent G, Dantressangle P, Vyvyan D, Mowshowitz A, Mitsou V (2008) A dynamic distributed federated database. In: Proceedings of 2nd annual conference international technology alliance
27. Vechtomova O (2009) Introduction to information Retrieval Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze (Stanford University, Yahoo! Research, and University of Stuttgart) Cambridge: Cambridge University Press, 2008, xxi+ 482 pp; hardbound, ISBN 978-0-521-86571-5. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info
28. Mudgil P, Sharma AK, Gupta P (2013) An improved indexing mechanism to index web documents. In: 2013 5th International conference and computational intelligence and communication networks
29. Taelman R, Van Herwegen J, Vander Sande M, Verborgh R (2018) Comunica: a modular SPARQL query engine for the web. In: The semantic web–ISWC 2018: 17th international semantic web conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II 17, pp. 239–255. Springer
30. Anderson A, Borfitz D, Getz K (2018) Global public attitudes about clinical research and patient experiences with clinical trials. JAMA Netw Open 1(6):182969
31. Tai-Seale M, Olson CW, Li J, Chan AS, Morikawa C, Durbin M, Wang W, Luft HS (2017) Electronic health record logs indicate that physicians split time evenly between seeing patients and desktop medicine. Health Aff (Millwood) 36(4):655–662
32. Vedsted P, Christensen MB (2005) Frequent attenders in general practice care: a literature review with special reference to methodological considerations. Public Health 119(2):118–137. https://doi.org/10.1016/j.puhe.2004.03.007
33. Ragab M, Savateev Y, Oliver H, Tiropanis T, Poulovassilis A, Chapman A, Taelman R, Roussos G (2024) Decentralized search over personal online datastores: architecture and performance evaluation. In: International conference on web engineering, pp. 49–64. Springer
34. Lindeborg A, Ödquist K (2024) Enhancing privacy in social matching through mixnets and solid pods: a comparative study of privacy enhancing technologies
35. Lua EK, Crowcroft J, Pias M, Sharma R, Lim S (2005) A survey and comparison of peer-to-peer overlay network schemes. IEEE Commun Surv Tutor 7(2):72–93. https://doi.org/10.1109/COMST.2005.1610546
36. Ragab M, Tommasini R, Eyvazov S, Sakr S (2020) 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
37. Hartig O (2013) An overview on execution strategies for linked data queries. Datenbank-Spektrum 13:89–99
38. Moaawad MR, O Mokhtar HM, Al Feel HT (2017) On-the-fly academic linked data integration. In: Proceedings of the international conference on compute and data analysis, pp. 114–122
39. Ge Y-F, Orlowska M, Cao J, Wang H, Zhang Y (2022) Mdde: multitasking distributed differential evolution for privacy-preserving database fragmentation. VLDB J 31(5):957–975
40. Crestani F, Markov I (2013) Distributed information retrieval and applications. In: Advances in information retrieval: 35th European conference on IR research, ECIR 2013, Moscow, Russia, March 24-27, 2013. Proceedings 35, pp. 865–868. Springer
41. Nordström E, Rohner C, Gunningberg P (2014) Haggle: opportunistic mobile content sharing using search. Comput Commun 48:121–132
42. Balakrishnan H, Kaashoek MF, Karger D, Morris R, Stoica I (2003) Looking up data in p2p systems. Commun ACM 46(2):43–48
43. Sakr S, Bonifati A, Voigt H, Iosup A, Ammar K, Angles R, Aref W, Arenas M, Besta M, Boncz PA et al (2021) The future is big graphs: a community view on graph processing systems. Commun ACM 64(9):62–71
44. Ragab M, Tommasini R, Sakr S. Comparing schema advancements for distributed rdf querying using sparksql

45. Ragab M (2022) Towards prescriptive analyses of querying large knowledge graphs. In: European conference on advances in databases and information systems, 639–647. Springer
46. Ragab M, Tommasini R, Awaysheh FM, Ramos JC (2021) An in-depth investigation of large-scale rdf relational schema optimizations using spark-sql
47. Chen Y, Wang W, Liu Z, Lin X (2009) Keyword search on structured and semi-structured data. In: Proceedings of the 2009 ACM SIGMOD international conference on management of data, pp. 1005–1010
48. Ragab M, Savateev Y, Oliver H, Tiropanis T, Poulovassilis A, Chapman A, Roussos G (2024) Unlocking the potential of health data with decentralised search in personal health datastores. In: Companion proceedings of the ACM on web conference 2024, pp. 1154–1157
49. Ragab M, Awaysheh FM, Tommasini R (2021) 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
50. Ragab M, Adidarma AS, Tommasini R. Papaya: A library for performance analysis of sql-based rdf processing systems. Semantic Web (Preprint), 1–19
51. Ragab M, Tommasini R, Sakr S (2019) Benchmarking spark-sql under alliterative rdf relational storage backends
52. Arana N, Ragab M, Tiropanis T (2024) An investigation into the feasibility of performing federated learning on social linked data servers. In: Companion proceedings of the ACM on web conference 2024, 1712–1714
53. Tommasini R, Ragab M, Falcetta A, Valle ED, Sakr S (2020) A first step towards a streaming linked data life-cycle. In: The Semantic Web–ISWC 2020: 19th International semantic web conference, Athens, Greece, November 2–6, 2020, Proceedings, part II 19, pp. 634–650. Springer
54. Du J, Rong J, Wang H, Zhang Y (2021) Neighbor-aware review helpfulness prediction. Decis Support Syst 148:113581
55. Moawad MR, Maher MMMZA, Awad A, Sakr S (2019) Minaret: a recommendation framework for scientific reviewers. In: the 22nd International conference on extending database technology (EDBT)