SolidSessionBench: Realistic Query Sequences for User-Oriented Decentralized Environments

Ruben Eschauzier[1 [0000−0002−6475−806X]] and Ruben Taelman[1 [0000−0001−5118−256X]]

Department of Electronics and Information Systems, Ghent University – imec

Abstract. To build user-facing applications on decentralized data, client-side decentralized query engines offer a natural integration point. However, optimizing queries in these environments is challenging due to a lack of prior knowledge regarding data distribution and availability. By repeatedly interacting with the same engine from the same user, applications allow the engine to overcome this knowledge gap by exploiting patterns in previously issued queries. However, evaluating these techniques accurately is difficult because public query logs are scarce, and existing benchmarks rely on static, template-based structures that fail to capture realistic behavioral variability. To fill this gap, we introduce an evidence-based simulation of client query usage patterns as a reusable benchmark enabling realistic and reproducible evaluation of client-side optimization techniques. We demonstrate its utility by evaluating client-side caching strategies for link traversal query processing. By comparing our sequence-based approach against traditional, non-sequence-based workloads, we highlight how realistic sequence simulation heavily influences performance evaluations. Ultimately, our benchmark reveals important performance dynamics, such as the impact of session switching and query refinement, that traditional workloads do not. We conclude that modeling realistic user interactions provides an essential foundation for accurately evaluating client-side optimization techniques in decentralized environments.

Keywords: Link Traversal-based Query Processing · Benchmarking · Workload Generation · Caching.

Resource Type: Benchmark
License: MIT
Persistent URI: https://doi.org/10.5281/zenodo.20079182
Source Code: https://github.com/SolidBench/
Raw Evaluation Data: https://doi.org/10.5281/zenodo.20056412
Canonical Citation: R. Eschauzier, R. Taelman., "SolidSessionBench: Realistic Query Sequences for User-Oriented Decentralized Environments", *ISWC*, 2026.

1 Introduction

Emerging web applications increasingly rely on decentralized architectures to give users control over their personal data, requiring query engines to dynam-

ically fetch and integrate information distributed across the web. While centralizing this data simplifies query processing, it often conflicts with privacy requirements and fine-grained access controls. Structured decentralized ecosystems such as Solid [51] and Mastodon [48] encourage users to keep their own data in small, independent stores that follow shared data models. This approach strengthens user autonomy, yet it makes query execution more difficult because engines must identify and query many heterogeneous data sources, each with its own access policies.

Traditional federated querying approaches [54,52,32] support decentralization of sources, but typically assume a small (10-100) set of well-described and consistently available endpoints [17]. These assumptions fail once data is spread across thousands of small units that enforce local access-control policies. In addition, enforcing fine-grained access control policies [21] on large sources with expressive interfaces introduces notable overhead [45], making standard approaches unsuitable for highly decentralized environments.

Newer decentralized environments, such as Solid, often expose data as lightweight HTTP resources instead of highly expressive query interfaces. Many of these resources are not known in advance and can only be discovered during query execution. Their simplicity lowers the cost of enforcing fine-grained controls. This setting requires a federation model that can operate over a very large number of small sources, many of which may be encountered at runtime, without relying on prior aggregation. Traditional federated approaches are insufficient in this setting due to the assumption of a small number of large endpoints.

Executing SPARQL queries using Link Traversal-based Query Processing (LTQP) [30] fits these needs. It discovers documents incrementally by following URIs encountered during execution, starting from an initial set of seed documents provided by either the user or the query. Its link-driven, asynchronous traversal supports fine-grained permissions and avoids the requirement to know the entire data distribution beforehand.

However, the fact that LTQP discovers documents incrementally creates significant challenges for efficient execution. Using the traditional optimize-then-execute paradigm is difficult, as the query itself determines which data will be accessed. As a result, the optimizer has no prior knowledge of the relevant data until the query's traversal has already begun.

Some approaches optimize query execution without prior knowledge by using adaptive query processing [26] and zero-knowledge query planning [28]. These yield modest but inconsistent performance gains, depending on the decentralized environment and the heuristics used [58]. However, other aspects of zero-knowledge LTQP optimization, such as link prioritization [31], are unlikely to improve query result arrival times in emerging decentralized environments [20].

Consequently, other works explore using precomputed server-side information as prior knowledge or relevance indexes to efficiently execute LTQP queries [59,61,25]. While these approaches offer greater performance benefits, they require the server to maintain these indexes. This raises the cost of serving resources and complicates privacy and access control.

To avoid server-side overhead, engines can derive prior knowledge directly on the client. In decentralized settings, LTQP is inherently client-side, with a single engine instance serving one or more applications. As these applications issue sequences of correlated queries, the engine identifies frequently accessed data and its characteristics. This allows engines to implement *personalized* query optimization algorithms that leverage local access patterns to build necessary prior knowledge. Benchmarking these approaches requires realistic query sequences. Traditional benchmarks (e.g., WatDiv [1], BSBM [10]) execute isolated or repeated query templates, which fails to capture client-side dynamics accurately. Furthermore, while centralized SPARQL optimization relies on real query logs [9,47,12,55], such logs do not yet exist for decentralized LTQP environments. Consequently, we need a standardized benchmark that realistically simulates user query sequences to prevent fragmented evaluation practices.

In this paper, we address this gap in existing benchmarks by introducing SolidSessionBench, a new benchmark that builds upon SolidBench [58] to generate realistic query sequences. These sequences capture three usage patterns identified in social network application literature, the domain simulated by SolidBench, alongside patterns observed in real-world SPARQL logs. Consequently, our benchmark enables researchers to evaluate personalized query optimization techniques against the actual user behaviors they will encounter in real client-facing applications.

To demonstrate our benchmark's utility, we implement three baseline client-side caching strategies in the LTQP engine Comunica [56,58]. We use these intentionally simple algorithms to establish a performance baseline and highlight the necessity of sequence-based evaluation.

2 Literature Review

2.1 Existing Benchmarking Approaches

For the evaluation of client-side benchmarking techniques that rely on preceding queries, we primarily investigate caching literature, as this is a well-studied problem relevant to our work.

In the literature, we find three distinct benchmarking approaches: simulated micro benchmarks, simulated end-to-end query benchmarks, and real-world query logs.

Simulated micro benchmarks isolate specific execution variables, such as cache size [27] or indirectly, cache hit rate [43]. While useful for analyzing isolated mechanisms, they provide an incomplete picture of performance under realistic, dynamic workloads.

Simulated end-to-end benchmarks evaluate overall behavior using query sequences. However, these sequences are typically constructed by randomly grouping isolated queries [44,10,27,43,23], applying simple statistical distributions like Pareto [64,38,6], or relying on narrow application scenarios [15,38] with custom build query sequences. Each of these methods suffers from distinct limitations.

Random grouping ignores real-world behavior entirely. Simple statistical distributions are too basic to capture complex user habits. Finally, static scenarios generate too few queries to provide comprehensive benchmark coverage. Consequently, none of these approaches simulate how users actually explore data, missing how they group related queries into short sessions and constantly adjust their search parameters based on immediate results.

Real-world query logs provide the gold standard for evaluation. While extensive logs are widely used to benchmark centralized SPARQL endpoints [50,55,9,65,46,35], none currently exist for decentralized LTQP environments.

To avoid the fragmented evaluation practices of current simulated approaches, the next best option is a unified simulated benchmark. A shared benchmark would provide a common basis for comparison, improve reproducibility, and prevent each paper from having to create its own evaluation methodology.

2.2 Real-world Query Usage Patterns

To inform the benchmark design on what patterns should be simulated, we will outline three relevant client query usage patterns observed in the literature. The relevance is determined for our social network application use case.

User Interest-Based Query Behavior Work on social networks shows users form communities based on shared interests [40,16,7,34]. These networks exhibit power-law degree distributions, small-world structures, and scale-free properties. This topology creates a dense core of high-degree nodes that connect many smaller, strongly clustered groups [40]. Simulation studies confirm that this interest-driven group formation is stable and reliably modeled [2,24].

Furthermore, empirical research formally establishes that shared interests directly drive a higher degree of explicit and implicit interactions between users [41]. These findings justify our simulation of user-relatedness and interest-driven behavior when generating query sequences.

Logical Query Sessions Analysis of web browsing behavior shows that activity is best grouped into *logical sessions*: sequences of user interactions focused on specific goals, rather than purely time-bound intervals [39,49,53]. These task-oriented sessions frequently involve non-linear navigation, including backtracking. New sessions begin in approximately 15% of clicks, following a log-normal distribution with average size and depth means of 6.1 and 3, respectively.

These patterns also appear in application navigation. In a decentralized social media application, for example, a user viewing a feed might click a creator's username, view comments, or like a post. Because the parameters of each new query derive directly from the preceding query's results, these queries chain together into structures mirroring logical web sessions. Additionally, direct searches for specific posts or topics reflect standard session-switching behavior.

Consequently, interactive applications generate sequences of interdependent requests [63]. This dynamic informs the design of Facebook's TAO data store, which is explicitly optimized for workloads driven by the user's step-by-step traversal of the social graph within the interface [14].

Query Refinement DBpedia SPARQL logs show users frequently submit related queries in streaks. In these streaks, queries share an underlying goal and are iteratively refined to achieve this objective [13,66]. While early session studies focused on robotic traffic [66,13], our benchmark models organic sequences generated through software interactions [66,37]. Because application design, rather than underlying data organization, drives these organic queries, we expect these patterns to also exist in decentralized environments. We exclude robotic queries from our benchmark, as these rely heavily on templated query executions that traditional benchmarks can already simulate [58,10,1]. Finally, across all query logs, analysis of total usage and reformulations shows that adding or removing FILTER, UNION, and OPTIONAL operators drives the vast majority of reformulations.

Recent analyses of Wikidata query logs further demonstrate that query development is an iterative process of design, debugging, and maintenance [5]. Based on this empirical data, exploratory querying encompasses six core behaviors [5]: **result refinement** to reduce records via filters or selective triple patterns, **result generalization** to retrieve more entities by broadening parameters, such as adding superclasses or increasing limits, **result extension** to fetch additional attributes by appending new triple patterns, **bug fixing** to correct structural errors like invalid URIs, **query refinement** to adjust complex language features, and **shifting query intent** to switch topic by changing core predicates or URIs in the query.

These findings confirm that real-world querying relies on dynamic, step-by-step modifications rather than the static execution of independent templates. Consequently, existing evaluation frameworks fail to capture how users actively interact with data systems. Evaluating system performance for these exploratory query sequences remains a critical research challenge, explicitly requiring new benchmarks to validate progress [5]. Therefore, modeling these empirical sequence patterns is essential to accurately assess client-side query optimization techniques.

3 Query Sequence Benchmark

In this section, we detail our simulation methodology and the resulting hyperparameters. We base our sequence generator on SolidBench [58] to simulate a highly distributed network of a decentralized social media application within the Solid ecosystem, using data described by a single ontology.

SolidBench builds upon the Social Network Benchmark (SNB) dataset [19,3] and applies the Linked Data Benchmark Council (LDBC) choke point methodology [4]. This methodology is a design strategy that explicitly targets query execution and optimization bottlenecks in current database management systems [11]. By first identifying critical execution challenges, such as estimating cardinality during traversal or determining optimal join orders [19], the benchmark designs queries that rigorously test these specific limitations.

To represent individual Solid pods, the benchmark utilizes a four-step sequential architecture: it generates the underlying SNB dataset via the LDBC Datagen, introduces non-query noise triples and other data enhancements, fragments the data so that all information related to a specific user resides in a single pod, and instantiates the SPARQL queries. Under default settings, this process generates 158,233 RDF files across 1,531 data vaults, totaling 3,556,159 triples.

Its workload combines standard SNB *interactive* queries with custom *discover* templates, which explicitly target Link Traversal Query Processing (LTQP) choke points in decentralized environments. To produce the final workload, queries are grouped by template and repeatedly instantiated with various values.

To support sequence-based evaluation workload, we extend three core components of the SolidBench ecosystem:

- **Query Generator:** Replaced to produce correlated query sequences that reflect empirical usage patterns. We only reuse the generator architecture as its static instantiation logic has almost no overlap to our simulation-based methodology.
- **Dataset Generator:** We enhance the data generation process to model interest-based similarities between simulated users.
- **Benchmark Runner:** We integrate sequence execution logic into the existing runner to facilitate straightforward, reproducible sequence-based experiments and allow easy generation of default configuration files.

3.1 Simulation Methodology

Because real-world query logs for highly decentralized environments are unavailable, SolidSessionBench ensures realism by combining established empirical usage patterns. We maintain topological realism by utilizing the proven LDBC social network data generator. To generate realistic query sequences, we base our simulation methodology directly on the empirical research summarized in Section 2.2. We validate this methodology by benchmarking caching algorithms. Because cache performance depends heavily on data locality and sequence continuity, this evaluation serves as a proxy to demonstrate that our simulated sequences capture essential realistic dynamics that static or random workloads miss.

Our methodology models three interdependent behaviors within each query sequence: task-oriented logical sessions, user interests for data locality, and refinement patterns for iterative query development.

Logical Query Sessions Formally, a logical query session [39,49,53] is a sequence $S = (q_1, \ldots, q_n)$ initiated by a goal G, representing the user's initial intention (e.g., exploring a forum) and determining only the initial query q_1. Subsequent queries are selected through a directed graph of valid template transitions, where the output type of one query determines which templates can be instantiated next. When possible, instantiation values for q_{i+1} are drawn from the result set $R(q_i)$.

To model logical sessions in our query sequences, we categorize queries into four topics: person-specific messages, forum information, person attributes, and message attributes. We create a directed graph to govern session flow by mapping each template's output types to valid subsequent templates. Within a session, a template dynamically derives its instantiation values from the previous query's results. We simulate this process by executing centralized LDBC SNB queries via QLever [8] and randomly sampling the results to instantiate the next template, defaulting to interest-based instantiation if the result set is empty.

For example, consider a dataset where Alice knows Bob, and Bob has published a post. The sequence generator initiates a session by executing Q1 to discover Bob's posts. The engine returns a specific post URI (e.g., `<.../bob/posts/1>`). To simulate a user clicking this post, the generator selects a subsequent query template (Q2) that specifically expects a 'post' instantiation value and dynamically injects the URI as the subject to fetch the post's content. This strict directional dependency ensures structured logical sessions emerge naturally.

We simulate the session switching behavior from Section 2.2 using a log-normal probability distribution (mean 15%) [39,53]. A switch randomly triggers either a new session (interest-based) or resumes a previous one (using its last query's results). This means that each *query sequence* can contain multiple *logical sessions* concurrently. To prevent repetition and ensure uniform template distribution, we prohibit backtracking and scale down a template's selection probability based on its execution frequency.

Unlike web browsing, which often transitions to disjoint resources, such as entirely different websites, our session switches operate over a shared graph, creating structural data overlap. This overlap, combined with session resumption, requires client-side optimizations to manage both short-term (intra-session) and long-term (cross-session) temporal locality.

User Interest Modeling Individuals interact more frequently with users sharing similar interests [41]. To simulate this behavior, we use the underlying LDBC dataset utilized by SolidBench [19], which inherently models realistic user preferences. The LDBC generator, Datagen, produces person profiles containing specific interests, educational backgrounds, and workplaces. It links users based on these demographic and interest overlaps, yielding a non-random network topology with a realistic degree distribution.

We use this interest-driven topology to generate template instantiation values for the first query of each logical session. Specifically, we construct a binary feature vector v_k for each user k: $v_k = (x_{k,1}, \ldots, x_{k,n}, y_{k,1}, \ldots, y_{k,m})$, where $x_{k,j} = 1$ if user k holds stated interest j, and $y_{k,l} = 1$ if user k completed study l.

For each sequence, we randomly select a base person k and compute the cosine similarity between v_k and the feature vectors of all other users. For example, if base person "Alice" explicitly lists "Wildlife" as an interest, her feature vector

aligns perfectly with another user who shares this interest, but not with a user who solely lists "Pottery".

Let N be the total number of users. To maintain efficiency at large scale factors, we restrict our candidate pool to $C_{k,M}$, defined as the set of the top $M < N$ users ranked by similarity to k. Template instantiation values are then selected probabilistically using a softmax function with temperature T:

$$P(i \mid k) = \frac{e^{s_{k,i}/T}}{\sum_{u \in C_{k,M}} e^{s_{k,u}/T}}, \tag{1}$$

where $s_{k,i}$ is the similarity score between base person k and candidate $i \in C_{k,M}$. Using this softmax, our simulation prioritizes instantiating queries for "Alice" using other wildlife enthusiasts. For templates requiring non-person values (e.g., posts or forums), selection probabilities are derived from the similarity scores of the content's creators, ensuring Alice is routed to "Wildlife" related resources.

Refinement Patterns The benchmark simulates query refinement by randomly applying composable transformations to an instantiated query template. Following our literature review, we specifically focus on transformations, and their reciprocal counterparts, that add or modify BGPs (Basic Graph Patterns), OPTIONALs, UNIONs, and query instantiation values.

To ensure transformations meaningfully change query execution dynamics rather than superficially appending triples, we apply the choke point-based design methodology [4,11]. The generator sequentially refines a base SolidSessionBench template, translating empirical exploratory usage patterns [5] (Section 2.2) into concrete execution bottlenecks via planning (P) and traversal (T) choke points. Planning choke points challenge query plan generation, while traversal choke points change web data discovery and traversal dynamics.

Specifically, planning choke points force engines to adapt execution plans: **P.1** tests selectivity estimation through `FILTER` modifications; **P.2** complicates filter push-downs and non-bgp query planning through `UNION` or `OPTIONAL` clauses; and **P.3** and **P.4** challenge plan optimization through additions of selective and non-selective triple patterns, respectively. Traversal choke points expand upon SolidBench [58,57]: **T.1** resizes the reachable sub-web via predicate additions; **T.2** shifts the sub-web by changing the query instantiation value and thus the traversal seed documents; **T.3** modifies the required traversal hops to obtain relevant data(SolidBench L.1–L.6 [57]); and **T.4** impacts structural index usability (SolidBench L.8 [57]).

We map these choke points directly to exploratory query behaviors:

- **Result Refinement:** Narrows results via `FILTER` constraints (P.1) and selective triple patterns (P.3).
- **Result Generalization:** Expands queried classes by adding selective (P.3) or non-selective (P.4) triple patterns to Basic Graph Patterns (BGPs).

- **Result Extension:** Fetches attributes by appending triple patterns to BGPs (P.3, P.4) or introducing `OPTIONAL` and `UNION` blocks (P.2).
- **Query Refinement:** Modifies query structure by altering `OPTIONAL` and `UNION` blocks (P.2).
- **Shifting Query Intent:** Substitutes instantiation values to start traversal from different seed documents (T.1, T.2).

The generator supports variable instantiation within refinements, enabling users to bind pattern variables to specific values randomly sampled from configuration files. For example, a `FILTER` refinement can dynamically restrict posts to a specific month using instantiation values within a FILTER clause, identical to standard query template instantiation.

Users can define custom refinement patterns via JSON files to add to the default configurations. These defaults, alongside documentation and pattern descriptions, are available on (https://github.com/SolidBench/SolidBench.js/tree/master/templates/refinements).

3.2 Hyperparameters

The sequence generation process introduces several new hyperparameters, which are detailed in Table 1 together with their default values. To guarantee reproducibility despite extensive random sampling, a single seeded pseudo-random number generator (PRNG) is used throughout the entire pipeline.

Table 1. LogT-normal distribution parameters (μ, σ) control the queries per sequence, queries per logical session, and the probability of switching logical sessions. Refinement parameters define the likelihood (`patternProbability`) and bounds (`min/maxRefinementLength`) of generating a refinement sequence for a given query. Instantiation relies on a softmax `temperature` and a `maxLogits` cutoff for similarity-based entity selection.

Category	Parameter	Value
Distributions (μ, σ)	Sequence Length	3.0, 0.2
	Session Length	1.7, 0.5
	Transition Probability	-2.0, 0.25
Refinements	`patternProbability`	0.1
	`min/maxRefinementLength`	2, [3–5]
Instantiation	`temperature`	0.1
	`maxLogits`	5–10

4 Caching in Link Traversal-based Query Processing

Unlike traditional query engines that evaluate queries against a centralized, pre-built index, LTQP evaluates queries directly over the live Web [57,29] using

the "follow-your-nose" principle. Starting from a seed URI, the engine fetches data through a *link queue* governed by *reachability criteria* [30] and stores retrieved triples in an in-memory *active local store*, typically using dictionary encoding [56]. Caching in this paradigm has been shown to mitigate the high network latency of LTQP by retaining recently accessed Web documents [27].

We use the Comunica LTQP engine [56,58]. As a JavaScript-based engine, it asynchronously interleaves traversal and local query execution through the event loop. We integrate three Least Recently Used (LRU)-based caching approaches, all of which cache dereferenced triples in-memory to reduce latency:

Unindexed Cache This baseline caches raw triple arrays with source metadata and indexes them in the active local store upon a cache hit.

Indexed Cache This approach pre-indexes triples before caching and, upon a hit, extracts only triples matching the query patterns. This reduces local store memory and indexing overhead, at the cost of higher initial ingestion costs.

Indexed Cache + Cardinality Estimation This approach uses the indexed cache to estimate quad pattern cardinalities across cached documents, replacing zero-knowledge heuristics [28] with Comunica's cost-based optimizer [56] to generate more accurate query plans.

We evaluate small (350k quads, ∼10% of the dataset), medium (700k, ∼20%), and large (1.4M, ∼40%) cache capacities. Our implementations adhere to RFC 9111 [22] and interpret standard HTTP headers such as `Cache-Control`. Because the benchmark dataset is static, entries require no revalidation and never become stale, so observed hit rates are likely higher than in a live environment.

Comparing these strategies establishes caching baselines for realistic user-centric workloads and provides a basis for future research into advanced caching approaches.

5 Evaluation

We evaluate our benchmark using a scale 0.1 SolidBench dataset comprising 158,233 RDF files, 1,531 vaults, and 3,556,159 triples. Applying the default hyperparameters from Table 1, we generated 8 sequences totaling 192 interactive-workload queries, encompassing both discover and short templates. Experiments were conducted on an Ubuntu 20.04 machine (Intel Core i5-9400) with 16 GB RAM and a 180-second timeout. To minimize variance and simulate realistic networks, we repeated each sequence 10 times with 70–90 ms injected latency to simulate realistic network conditions. Caches were scoped to individual sequences, allowing reuse across interleaved or resumed logical sessions while ensuring no state was shared between repetitions.

These experiments aim to validate our sequence generation by assessing how our design choices influence query correlation, measured via cache hit rates and

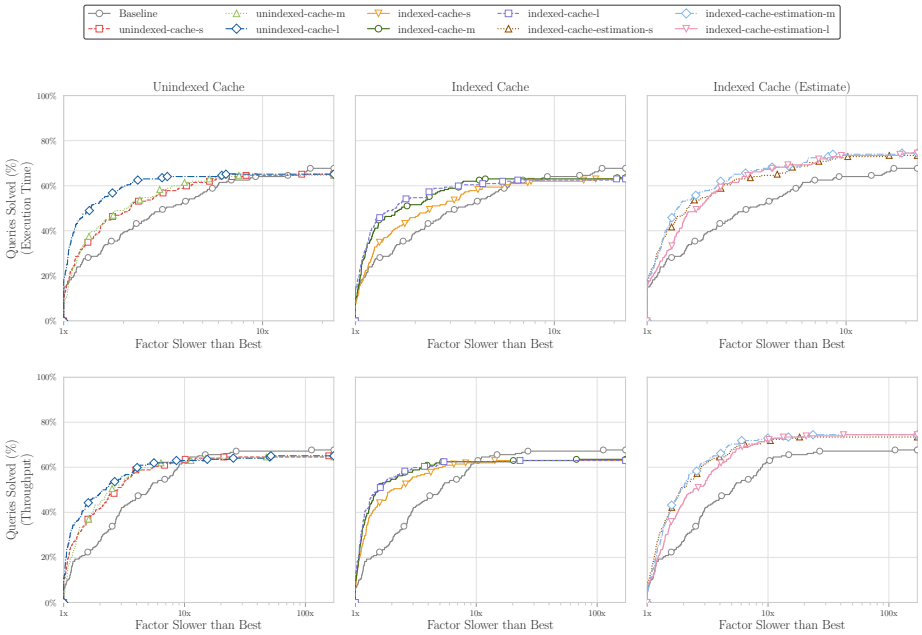

Fig. 1. Performance profiles of the tested cache algorithms. The y-axis shows the percentage of queries executing within a factor (x-axis) of the fastest algorithm. Curves closer to the top-left indicate better overall performance. The y-intercept ($x = 1$) shows how often an algorithm is the outright fastest, while the rightward trajectory demonstrates robustness (e.g., $x = 2, y = 80\%$ means 80% of queries execute within twice the fastest time). From the figure, we observe that all caching-based approaches outperform the no-cache baseline.

evictions, and to establish a caching performance baseline for a Link Traversal-based Query Processing (LTQP) engine. While we report speedups to demonstrate utility, the primary focus is benchmark validation rather than exhaustive algorithm evaluation.

5.1 Global Cache Performance

Figure 1 compares performance profiles [18] for each approach against the no-cache baseline. As expected, all caching methods outperform the baseline.

For the unindexed cache, small (350k triples) and medium (700k triples) sizes perform similarly. The large cache (1,400k triples) significantly improves speed by retaining enough entries across the sequence. However, the baseline resolves more total queries, suggesting cache management overhead causes timeouts for longer queries.

The indexed cache has a higher failure rate, as initial indexing costs slow execution when the eviction rate outweighs the increased efficiency of cache hits. Still, it accelerates successful queries more than the unindexed cache. It

also reaches critical capacity sooner, with medium and large sizes performing similarly.

Overall, the indexed cache with cardinality estimation performs best, increasing speed and solving more queries. Notably, larger cache sizes degrade performance. A possible mechanism for this is that the estimator evaluates all cache entries, including irrelevant unreachable data from past queries. This accumulated noise likely can degrade estimates, suggesting the estimator favors smaller, fresher caches.

5.2 Logical Sessions

To evaluate the effect of logical sessions on client-side caching, we measure the cache hit rate deviation (Δ) from the overall average for queries that start a new session, switch to a previous session, or remain within the current session.

Web browsing workloads exhibit strong temporal locality. Since LRU retains recently accessed objects, its hit rate follows this locality: it is highest within a session, lowest for new sessions that introduce a different working set, and intermediate when returning to a previous session [36,33,42,60].

Table 2 confirms this behavior. Within-session queries consistently achieve above-average hit rates, while new sessions incur the largest performance penalty. Returning to a previous session causes a smaller reduction, indicating that part of the previous working set remains cached. These effects are strongest for small and medium cache sizes.

Table 2. Impact of logical session status on algorithm performance. Values denote cache hit rate deviations from the global average across new, existing, and within-session queries. Negative values indicate reduced performance.

Cache Algorithm	**New Session**		**Existing Session**		**Within Session**	
	Abs.	**%**	**Abs.**	**%**	**Abs.**	**%**
unindexed-l	-0.066	-12.278	-0.014	-2.474	0.011	1.931
unindexed-m	-0.056	-15.649	-0.031	-7.900	0.024	5.486
unindexed-s	-0.071	-24.275	-0.021	-6.551	0.017	4.582
indexed-estimate-l	-0.053	-10.573	-0.015	-2.796	0.012	2.127
indexed-estimate-m	0.028	7.174	-0.000	-0.064	-0.000	-0.035
indexed-estimate-s	-0.044	-14.305	-0.013	-3.921	0.010	2.924
indexed-l	-0.026	-5.011	-0.003	-0.571	0.003	0.467
indexed-m	-0.087	-22.782	-0.021	-5.098	0.017	3.682
indexed-s	-0.033	-11.036	-0.015	-4.470	0.012	3.153

The observed behavior matches the expected response of LRU to temporally local workloads, indicating that our simulated logical sessions capture a key characteristic of real browsing behavior. This also suggests that cache sizing

and eviction policies should account for session-switching behavior. Future work should investigate retaining entries shared across sessions or maintaining session-aware caches to reduce misses when users alternate between tasks.

5.3 Refinement Patterns

To evaluate iterative query refinement simulation, we analyze cache behavior across refinement depths. Figure 2 shows that hit ratios increase progressively as queries undergo sequential refinement. Across all algorithms, hit rates increase at depths 2 and 3, confirming that refinement heavily reuses prior data.

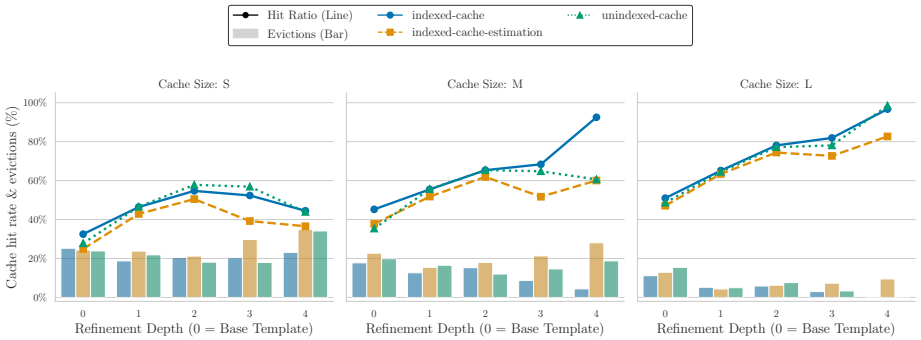

Fig. 2. Cache hit ratios and percentage of cache capacity evicted across refinement depths for small, medium, and large cache configurations.

At higher refinement depths, performance depends heavily on cache capacity. Large caches sustain high hit rates through depth 4, whereas small caches experience significant thrashing, and degraded hit ratios.

Surprisingly, the indexed cache without estimation maintains high hit rates at small capacities. Different caching mechanisms alter how the Node.js event loop schedules traversal and join executions, changing traversal order and rate. We suspect this variance increases temporal locality, though further analysis is needed to confirm the exact mechanism and explain why the indexed cache outperforms the unindexed baseline.

The observed cache hit rate and eviction differences directly translate to measurable execution speedups. Figure 2 illustrates that the geometric mean speedup generally scales positively with refinement depth, with a more pronounced effect for the throughput. The indexed-cache algorithm proves particularly effective for prolonged exploratory sequences, achieving substantial throughput gains at depth 4 compared to the unindexed alternatives.

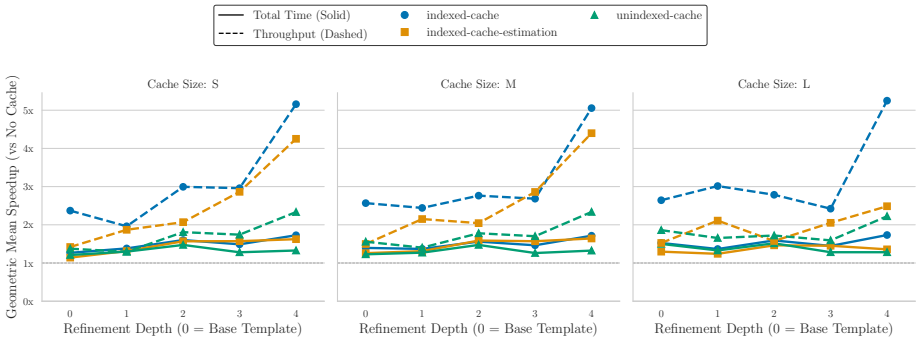

Fig. 3. Geometric mean of observed speedups and increases respectively for total execution time and query throughput relative to the no caching baseline across refinement depths.

5.4 Comparison To Other Evaluation Methods

Microbenchmarks - Theoretical Hit Rate Analysis Cache literature extensively uses microbenchmarks of various forms to analyze caching performance [43,27]. To compare the conclusions of such a benchmark to the conclusions of the benchmark introduced in this paper we investigate theoretical caching speedup without management costs, by executing each query template twice with simulated cache miss probabilities. We excluded the estimation-based approach, as using the full cache for estimation would skew the results for miss rates above zero.

Figure 4 shows representative performance profiles for the workload. Most templates (8 out of 12) show reliable improvement that scales with the hit rate for both algorithms. However, certain long-running queries (e.g., interactive-discover-6, 7, and short-3) show no speedup, and some even experience slowdowns. These templates traverse many solid pods but do not require all traversed data. We hypothesize that rapidly serving documents from an in-memory cache overwhelms the system with data, slowing result production. Conversely, HTTP network latency gives the JavaScript event loop time to process the join execution pipeline, yielding initial results faster since only the first few requests are needed.

Both caching approaches perform similarly overall, but the indexed cache significantly reduces time-to-first-result across most templates. As expected, indexed caching is highly advantageous without evictions. However, the initial indexing cost creates a trade-off: it amplifies both speedups and slowdowns relative to an unindexed cache. While this micro-benchmark approach aligns with prior literature [43], our mixed findings show that micro-benchmarks alone fail to thoroughly evaluate client-side caching optimizations. This limitation demonstrates the clear need for the comprehensive benchmark we introduce in this work.

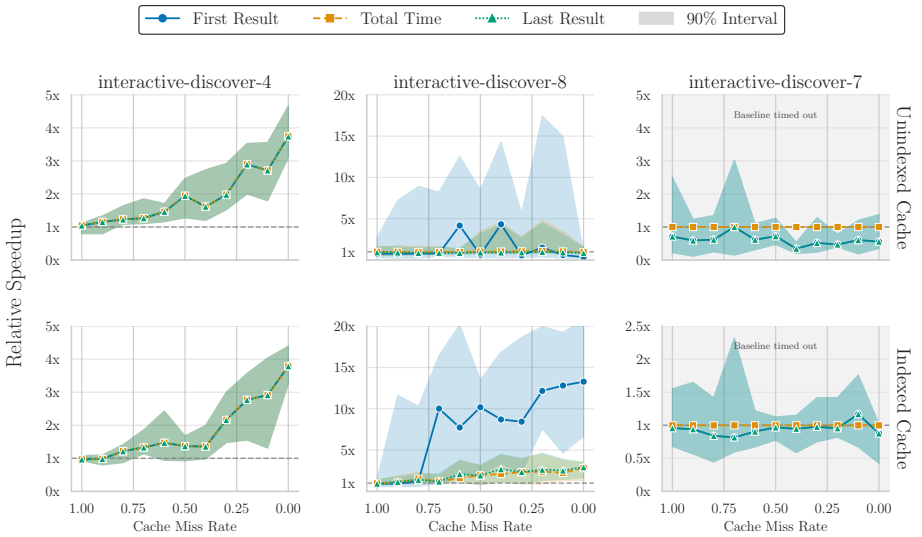

Fig. 4. Performance of unindexed and indexed caches across representative query templates. The y-axis shows the mean speedup relative to a baseline with a 0% hit rate. Shaded regions denote the 90th percentile

Random Query Sequences Another common approach is to use random query sequences from an existing benchmark [44,10,27,43,23]. Figure 5 shows execution times for sequences created by randomly grouping default SolidBench.js queries.

For these random sequences, only large caches perform well, while smaller ones degrade below the baseline. These results misleadingly suggest that large caches are vital, whereas our benchmark demonstrates significant performance benefits even with smaller sizes. As expected, the estimation cache performs well in both scenarios because cardinality estimation does not require cache hits.

Scope and Limitations We evaluate SolidSessionBench against microbenchmarks and random sequences, but explicitly exclude hand-crafted scenarios and simple distributions. Hand-crafted methods lack scalability, while simple distributions fail to capture complex user habits. By automating logical sessions and user interests, our benchmark functions as a scalable superset of both approaches, making direct comparison redundant. By contrast, the compared methods do not incorporate these sequence attributes, and our results show this omission risks drawing inaccurate conclusions.

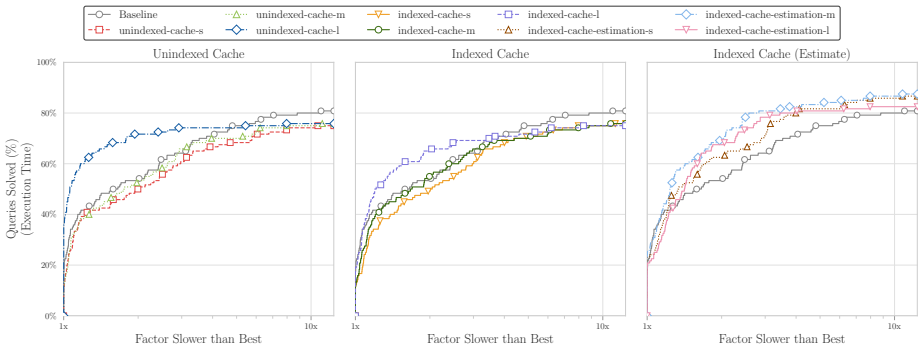

Fig. 5. Performance profile of the tested cache algorithms on random query sequences. The y-axis shows the percentage of queries that execute within a given performance factor (x-axis) of the fastest algorithm for that query.

6 Conclusion

We introduced SolidSessionBench, an evidence-based benchmark for generating realistic query sequences in decentralized environments. By modeling logical sessions, user interests, and iterative refinement, it captures client-side performance dynamics missed by traditional methods. While primarily designed for Solid, our underlying concepts apply to emerging standards like Linked Web Storage (LWS), which we will support once W3C specifications are finalized.

The simulation methodology is highly configurable. Adapting to new data schemas (e.g., a Mastodon network) only requires mapping the fragmentation configuration to the ActivityStreams vocabulary, not a structural overhaul. Extending the query workload is also straightforward: researchers can introduce new exploratory behaviors by defining a query template and mapping its dependencies in the session configuration graph.

To ensure longevity, we integrated SolidSessionBench into the core SolidBench ecosystem and documented a sustainability plan (https://github.com/SolidBench/SolidBench.js/wiki/Sustainability-Plan). As SolidBench is a standard community resource for evaluating decentralized query execution [20,62,26,59], this integration encourages broad adoption for evaluating client-specific optimizations.

The modular, MIT-licensed framework and reproducible experimental setups are available at https://github.com/RubenEschauzier/simulated-query-sequence-paper. This extensible architecture allows researchers to easily modify components for future requirements without disrupting the evaluation pipeline.

Acknowledgements This research was supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10). Ruben Taelman is a postdoctoral researcher at the Research Foundation – Flanders (FWO).

7 Declaration of Generative AI Use

In this work, we used Gemini to:

- Generate self-contained code snippets, which we validated through manual and automated testing.
- Generate code to produce figures.
- Identify and resolve bugs.
- Rewrite and condense paragraphs to improve clarity.

The authors reviewed all AI-generated content and take full responsibility for the final manuscript and codebase.

References

1. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of rdf data management systems. In: International Semantic Web Conference. pp. 197–212. Springer (2014)
2. Amblard, F., Bouadjio-Boulic, A., Gutiérrez, C.S., Gaudou, B.: Which models are used in social simulation to generate social networks? a review of 17 years of publications in jasss. In: 2015 Winter Simulation Conference (WSC). pp. 4021–4032. IEEE (2015)
3. Angles, R., Antal, J.B., Averbuch, A., Birler, A., Boncz, P., Búr, M., Erling, O., Gubichev, A., Haprian, V., Kaufmann, M., et al.: The ldbc social network benchmark (2020)
4. Angles, R., Boncz, P., Larriba-Pey, J., Fundulaki, I., Neumann, T., Erling, O., Neubauer, P., Martinez-Bazan, N., Kotsev, V., Toma, I.: The linked data benchmark council: a graph and rdf industry benchmarking effort. vol. 43, pp. 27–31. ACM New York, NY, USA (2014)
5. Arenas, M., Franconi, E., Hammerer, J., Hartig, O., Hose, K., Koesten, L., Konstantinidis, G., Libkin, L., Martens, W., Sasaki, Y., et al.: Exploring exploratory querying. Proceedings of the VLDB Endowment **18**(13), 5731–5739 (2025)
6. Arnold, B.C.: Pareto distribution. Wiley StatsRef: Statistics Reference Online pp. 1–10 (2014)
7. Backstrom, L., Huttenlocher, D., Kleinberg, J., Lan, X.: Group formation in large social networks: membership, growth, and evolution. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 44–54 (2006)
8. Bast, H., Buchhold, B.: Qlever: A query engine for efficient sparql+ text search. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. pp. 647–656 (2017)
9. Berendt, B., Hollink, L., Luczak-Rösch, M., Möller, K., Vallet, D.: Usewod2013 3rd international workshop on usage analysis and the web of data. 10th ESWC-Semantics and Big Data, Montpellier, France (2013)
10. Bizer, C., Schultz, A.: The berlin sparql benchmark. In: Semantic Services, Interoperability and Web Applications: Emerging Concepts, pp. 81–103. IGI Global Scientific Publishing (2011)
11. Boncz, P., Neumann, T., Erling, O.: Tpc-h analyzed: Hidden messages and lessons learned from an influential benchmark. In: Technology Conference on Performance Evaluation and Benchmarking. pp. 61–76. Springer (2013)

12. Bonifati, A., Martens, W., Timm, T.: DARQL: Deep Analysis of SPARQL Queries. In: Companion Proceedings of the The Web Conference 2018. pp. 187–190. WWW '18, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE (Apr 2018). https://doi.org/10.1145/3184558.3186975, https://dl.acm.org/doi/10.1145/3184558.3186975
13. Bonifati, A., Martens, W., Timm, T.: An analytical study of large sparql query logs. The VLDB Journal **29**(2), 655–679 (2020)
14. Bronson, N., Amsden, Z., Cabrera, G., Chakka, P., Dimov, P., Ding, H., Ferris, J., Giardullo, A., Kulkarni, S., Li, H., et al.: {TAO}:{Facebook's} distributed data store for the social graph. In: 2013 USENIX annual technical conference (USENIX ATC 13). pp. 49–60 (2013)
15. Chatzopoulos, S., Vergoulis, T., Skoutas, D., Dalamagas, T., Tryfonopoulos, C., Karras, P.: Atrapos: Evaluating metapath query workloads in real time. CoRR (2022)
16. Clark, A.: Understanding community: A review of networks, ties and contacts (2007)
17. Dang, M.H., Aimonier-Davat, J., Molli, P., Hartig, O., Skaf-Molli, H., Le Crom, Y.: Fedshop: A benchmark for testing the scalability of sparql federation engines. In: International Semantic Web Conference. pp. 285–301. Springer (2023)
18. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Mathematical programming **91**(2), 201–213 (2002)
19. Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.D., Boncz, P.: The ldbc social network benchmark: Interactive workload. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 619–630 (2015)
20. Eschauzier, R., Taelman, R., Verborgh, R.: Revisiting link prioritization for efficient traversal in structured decentralized environments. In: International Semantic Web Conference. pp. 575–593. Springer (2025)
21. Esteves, B., Pandit, H.J., Rodríguez-Doncel, V.: Odrl profile for expressing consent through granular access control policies in solid. In: 2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 298–306. IEEE (2021)
22. Fielding, R., Nottingham, M., Reschke, J.: Rfc 9111: Http caching (2022)
23. Folz, P., Skaf-Molli, H., Molli, P.: Cyclades: a decentralized cache for triple pattern fragments. In: European semantic web conference. pp. 455–469. Springer (2016)
24. Hamill, L., Gilbert, G.: Social circles: A simple structure for agent-based social network models. Journal of Artificial Societies and Social Simulation **12**(2) (2009)
25. Hanski, J., Taelman, R., Verborgh, R.: Observations on bloom filters for traversal-based query execution over solid pods. In: European Semantic Web Conference. pp. 228–233. Springer (2024)
26. Hanski, J., Van Braeckel, S., Taelman, R., Verborgh, R.: Link traversal over decentralised environments using restart-based query planning. In: International Conference on Web Engineering (2025)
27. Hartig, O.: How caching improves efficiency and result completeness for querying linked data. In: LDOW (2011)
28. Hartig, O.: Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In: Extended Semantic Web Conference. pp. 154–169. Springer (2011)
29. Hartig, O.: Sparql for a web of linked data: Semantics and computability. In: Extended Semantic Web Conference. pp. 8–23. Springer (2012)

30. Hartig, O., Bizer, C., Freytag, J.C.: Executing sparql queries over the web of linked data. In: International semantic web conference. pp. 293–309. Springer (2009)
31. Hartig, O., Özsu, M.T.: Walking without a map: Ranking-based traversal for querying linked data. In: International Semantic Web Conference. pp. 305–324. Springer (2016)
32. Heling, L., Acosta, M.: Federated sparql query processing over heterogeneous linked data fragments. In: Proceedings of the ACM Web Conference 2022. pp. 1047–1057 (2022)
33. Jin, S., Bestavros, A.: Temporal locality in web request streams: Sources, characteristics, and caching implications (1999)
34. Kalaitzakis, A., Papadakis, H., Fragopoulou, P.: Evolution of user activity and community formation in an online social network. In: Proceedings of the 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. pp. 1315–1320. IEEE (2012)
35. Lorey, J., Naumann, F.: Caching and prefetching strategies for sparql queries. In: Extended Semantic Web Conference. pp. 46–65. Springer (2013)
36. Mahanti, A., Eager, D., Williamson, C.: Temporal locality and its impact on web proxy cache performance. Performance Evaluation **42**(2-3), 187–203 (2000)
37. Malyshev, S., Krötzsch, M., González, L., Gonsior, J., Bielefeldt, A.: Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In: The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II 17. pp. 376–394. Springer (2018)
38. Martin, M., Unbehauen, J., Auer, S.: Improving the performance of semantic web applications with sparql query caching. In: Extended Semantic Web Conference. pp. 304–318. Springer (2010)
39. Meiss, M., Duncan, J., Gonçalves, B., Ramasco, J.J., Menczer, F.: What's in a session: tracking individual behavior on the web. In: Proceedings of the 20th ACM conference on Hypertext and hypermedia. pp. 173–182. ACM, Torino Italy (Jun 2009). https://doi.org/10.1145/1557914.1557946, https://dl.acm.org/doi/10.1145/1557914.1557946
40. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement. pp. 29–42 (2007)
41. Mitzlaff, F., Atzmueller, M., Benz, D., Hotho, A., Stumme, G.: User-relatedness and community structure in social interaction networks. arXiv preprint arXiv:1309.3888 (2013)
42. Mookerjee, V.S., Tan, Y.: Analysis of a least recently used cache management policy for web browsers. Operations Research **50**(2), 345–357 (2002)
43. Nicholson, H., Chrysogelos, P., Ailamaki, A.: Hpcache: memory-efficient olap through proportional caching revisited. The VLDB Journal **33**(6), 1775–1791 (2024)
44. O'Neil, P., O'Neil, E., Chen, X., Revilak, S.: The star schema benchmark and augmented fact table indexing. In: Technology Conference on Performance Evaluation and Benchmarking. pp. 237–252. Springer (2009)
45. Padia, A., Finin, T., Joshi, A., et al.: Attribute-based fine grained access control for triple stores. In: 3rd Society, Privacy and the Semantic Web-Policy and Technology workshop, 14th International Semantic Web Conference (2015)
46. Papailiou, N., Tsoumakos, D., Karras, P., Koziris, N.: Graph-aware, workload-adaptive sparql query caching. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 1777–1792 (2015)

47. Picalausa, F., Vansummeren, S.: What are real SPARQL queries like? In: Proceedings of the International Workshop on Semantic Web Information Management. pp. 1–6. SWIM '11, Association for Computing Machinery, New York, NY, USA (Jun 2011). https://doi.org/10.1145/1999299.1999306, https://dl.acm.org/doi/10.1145/1999299.1999306
48. Raman, A., Joglekar, S., Cristofaro, E.D., Sastry, N., Tyson, G.: Challenges in the decentralised web: The mastodon case. In: Proceedings of the internet measurement conference. pp. 217–229 (2019)
49. Sadagopan, N., Li, J.: Characterizing typical and atypical user sessions in clickstreams. In: Proceedings of the 17th international conference on World Wide Web. pp. 885–894 (2008)
50. Saleem, M., Ali, M.I., Hogan, A., Mehmood, Q., Ngomo, A.C.N.: Lsq: the linked sparql queries dataset. In: International semantic web conference. pp. 261–269. Springer (2015)
51. Sambra, A.V., Mansour, E., Hawke, S., Zereba, M., Greco, N., Ghanem, A., Zagidulin, D., Aboulnaga, A., Berners-Lee, T.: Solid: a platform for decentralized social applications based on linked data. MIT CSAIL & Qatar Computing Research Institute, Tech. Rep. **2016** (2016)
52. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: Fedbench: A benchmark suite for federated semantic data query processing. In: The Semantic Web–ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I 10. pp. 585–600. Springer (2011)
53. Schneider, F., Feldmann, A., Krishnamurthy, B., Willinger, W.: Understanding online social network usage from a network perspective. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement. pp. 35–48 (2009)
54. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: Fedx: Optimization techniques for federated query processing on linked data. In: The Semantic Web–ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I 10. pp. 601–616. Springer (2011)
55. Stadler, C., Saleem, M., Mehmood, Q., Buil-Aranda, C., Dumontier, M., Hogan, A., Ngonga Ngomo, A.C.: Lsq 2.0: A linked dataset of sparql query logs. Semantic Web **15**(1), 167–189 (2024)
56. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: a modular sparql query engine for the web. In: International Semantic Web Conference. pp. 239–255. Springer (2018)
57. Taelman, R., Verborgh, R.: Evaluation of link traversal query execution over decentralized environments with structural assumptions (2023)
58. Taelman, R., Verborgh, R.: Link traversal query processing over decentralized environments with structural assumptions. In: International Semantic Web Conference. pp. 3–22. Springer (2023)
59. Tam, B.E., Taelman, R., Colpaert, P., Verborgh, R.: Opportunities for shape-based optimization of link traversal queries. arXiv preprint arXiv:2407.00998 (2024)
60. Traverso, S., Ahmed, M., Garetto, M., Giaccone, P., Leonardi, E., Niccolini, S.: Unravelling the impact of temporal and geographical locality in content caching systems. IEEE Transactions on Multimedia **17**(10), 1839–1854 (2015)
61. Umbrich, J., Hose, K., Karnstedt, M., Harth, A., Polleres, A.: Comparing data summaries for processing live queries over linked data. World Wide Web **14**(5), 495–544 (2011)
62. Vandenbrande, M., Taelman, R., Bonte, P., Ongenae, F.: Incremunica: Web-based incremental view maintenance for sparql. In: European Semantic Web Conference. pp. 297–315. Springer (2025)

63. Vögele, C., van Hoorn, A., Schulz, E., Hasselbring, W., Krcmar, H.: Wessbas: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems. Software & Systems Modeling **17**(2), 443–477 (2018)
64. Williams, G.T., Weaver, J.: Enabling fine-grained http caching of sparql query results. In: International Semantic Web Conference. pp. 762–777. Springer (2011)
65. Zhang, W.E., Sheng, Q.Z., Qin, Y., Yao, L., Shemshadi, A., Taylor, K.: Secf: improving sparql querying performance with proactive fetching and caching. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. pp. 362–367 (2016)
66. Zhang, X., Wang, M., Zhao, B., Liu, R., Zhang, J., Yang, H.: Characterizing robotic and organic query in sparql search sessions. In: Web and Big Data: 4th International Joint Conference, APWeb-WAIM 2020, Tianjin, China, September 18-20, 2020, Proceedings, Part I 4. pp. 270–285. Springer (2020)