ILX: Intelligent "Location+X" Data Systems (Vision Paper)

Walid G. Aref[1], Ahmed M. Aly[2], Anas Daghistani[3], Yeasir Rayhan[1], Jianguo Wang[1], Libin Zhou[1]
[1]Purdue University, West Lafayette, IN, USA
[2]Meta Platforms, Menlo Park, CA, USA
[3]Umm Al-Qura University, Mecca, Saudi Arabia
[1]{aref, yrayhan, csjgwang, zhou822}@purdue.edu, [2]aaly@fb.com, [3]ahdaghistani@uqu.edu.sa

ABSTRACT

Due to the ubiquity of mobile phones and location-detection devices, location data is being generated in very large volumes. Queries and operations that are performed on location data warrant the use of database systems. Despite that, location data is being supported in data systems as an afterthought. Typically, relational or NoSQL data systems that are mostly designed with non-location data in mind get extended with spatial or spatiotemporal indexes, some query operators, and higher level syntactic sugar in order to support location data. The ubiquity of location data and location data services call for systems that are solely designed and optimized for the efficient support of location data. This paper envisions designing intelligent location+X data systems, ILX for short, where location is treated as a first-class citizen type. ILX is tailored with location data as the main data type (location-first). Because location data is typically augmented with other data types X, e.g., graphs, text data, click streams, annotations, etc., ILX needs to be extensible to support other data types X along with location. This paper envisions the main features that ILX should support, and highlights research challenges in realizing and supporting ILX.

CCS CONCEPTS

• **Information systems** → **Database design and models**; **Location based services**; *Geographic information systems*; *Database management system engines*; *Database query processing*.

KEYWORDS

machine learning-based location+X system, intelligent location servers, extensible location-based server, query processing, indexing

1 INTRODUCTION

Location data is ubiquitous due to the popularity of smart phones and location-detection devices. Moreover, location data services are getting into almost all aspects of life, and are getting very sophisticated. This warrants designing data systems that are well-optimized for handling and processing location data, and that treat location data as a first-class citizen. Unfortunately, this is not the case. Location data is almost always supported in data systems as an afterthought. Typically, systems that are originally optimized with other objectives in mind eventually get extended to support location data as an afterthought. For example, consider the many extensions of relational data systems to support location data. These systems are designed and are optimized to efficiently support relational data with location data being an afterthought add-on feature. Other examples include NoSQL and big data systems, e.g., Hadoop, Spark, etc., that are designed with other objectives in mind, and with location data either being entirely out of the picture, or is supported as an add-on after the fact. Afterwards, researchers try to fit location data into these systems, and possibly apply some tweaks with sub-optimal extensions to support location data into these systems. Contrast this with designing a system that is mainly optimized from the beginning to support location data as first-class citizen. A strong analogy would be when starting from a car and tweaking its design to make it fly vs. designing an airplane from scratch, or starting from a helicopter and extending its design to make it function as a submarine in addition to being a helicopter. While these extensions are possible given good engineering and resources, they would not perform as efficient as an airplane or a submarine that are designed from scratch as such.

Not to pick on any other researchers, the first author lists only the systems that his group has developed that follow that same pitfall above, e.g., AQWA [17] that extends on Hadoop [101], Location-Spark [108] that extends on Spark [122], Tornado/SWARM [34, 75] that extends over Apache Storm [6], SP-GiST [20] that extends PostgreSQL [105], and GRFusion [50, 51] that extends over VoltDB [106]. Clearly, Hadoop, Spark, Spark, PostgreSQL, and VoltDB have been originally optimized for non-location data. Many other researchers and industries follow the same paths with some notable successes, e.g., Oracle Spatial [12], Spatial Hadoop [43], and GeoSpark [119].

In recent years, in two keynote talks, the first author of this paper has been advocating for Location+X systems [18, 19] that highlight several research challenges and potential solutions for location-first systems that are to be augmented with other X data types, e.g., graphs, text, and relational data. This paper extends beyond the ideas in the two keynotes [18, 19], and presents the vision for Intelligent Location+X systems, ILX, for short. The paper highlights the main features of ILX, and identifies important research challenges in realizing it. Notice that there are already existing research that supports ILX-like "location-first" vision in some aspects. This paper helps identify and references these efforts whenever appropriate and as space permits.

2 HIGHLIGHTS OF ILX

ILX stands for Intelligent Location+X systems. In this section, we highlight each of the components of ILX, mainly the Location-first, Intelligence, and Extensibility components, and discuss the features that ILX and its surrounding eco-system should support.

2.1 The "L" in ILX: Location

Location and Location + Time are first-class citizens in ILX. In addition to being optimized for operating on location data, ILX and its supporting eco-system will provide the following important high-level location-based features.

2.1.1 **Protection, Privacy, and the Right to be Forgotten of User's Location Data.** ILX should guarantee the privacy of user's location data, e.g., as in [52, 97, 123]. Users should be able to learn and control what ILX knows about them. More generally, ILX should support the General Data Protection Regulation (**GDPR** [8]). Privacy and protection of user data and the right to be forgotten should be declaratively and intrinsically associated with location data in ILX and should not be left to the applications to enforce. Lower-level layers of ILX should be able to enforce these features, and the user should be able to audit and verify how her/his data is being used and the time it should expire from the system.

ILX should prevent snooping data by implementing an end-to-end encryption technique. Also, ILX should not allow users to track the locations of individuals. ILX should guarantee to only return query answers that cannot be used to reveal the identity of any user of her/his location. Moreover, an intelligent mechanism should be implemented in ILX to detect and block users that are trying to track other users or reveal their identity.

2.1.2 **Discovery, Integration, and Pricing of Location Data.** ILX should be able to support location data lakes [110] and identify relevant location datasets [38, 46] given user requests. ILX should support *location data integration* of the discovered datasets. Query issuers in ILX may not worry about which data set to use to answer a certain query. Thus, ILX's query language should have embedded in it the dataset discovery process. However, what users should be concerned with is the cost of answering their query. Location data collection and preparation is costly. Thus, the query execution engine and query optimizer for ILX should have cost of data and pricing as an optimization parameter while generating query plans and while discovering and selecting the appropriate location datasets needed to answer location-driven queries.

2.1.3 **Location Data Cleaning and Support for Uncertainty.** Like all other data sources, location data contains many errors and needs cleaning. The ILX eco-system should provide cleaning tools for location data, especially the ones uniquely related to location data, e.g., the faulty geographic colocation of two shopping stores in the same location and in the same time duration.

In addition to cleaning location data from data entry mistakes, location data is inherently uncertain, e.g., due to accuracy errors in location-detection devices. ILX's query language and execution engine should deal with the uncertainty in location data and provide query operators that reason given the uncertainty [102].

2.1.4 **Location Data Transactional and Online Analytics Support.** ILX should offer transactional support due to the heavy update nature of location and related data, e.g., due to objects continuously changing their locations and changing the associated data. Moreover, ILX should be able to perform online analytics [54, 85, 98, 116, 118], especially ones that are unique to location data. Finally, because geospatial data is hierarchical in nature (both in space and time), ILX should support hierarchical and multi-resolution analytics both in the space and time dimensions.

2.1.5 **Human-in-the-loop and Crowdsourcing.** Many transactions in ILX will involve human actions [44]. Thus, ILX should natively support Human-in-the-loop, humans-as-query-operators in location-based query evaluation pipelines, and humans as operators in long-standing transactions. ILX should protect the location privacy of crowdsourcing workers and tasks, e.g,. as in [111].

2.1.6 **Sampling, Predication, and Approximate Location-data Processing.** Given the massive sizes of location data and demands from location-service for online responses, it may not be feasible to process all location data made available for a given task in a timely fashion. Location-data sampling and approximate query processing techniques should be an integral component of ILX's query execution engine. Tradeoffs between the approximation quality and the runtime requirements of the location service tasks should be well-studied in the context of ILX.

2.1.7 **The Time Dimension, Data Streaming, and Continuous Data Support.** In ILX, the time line can be split into three time zones: the past time, the current time (Time NOW), and the future time. ILX should be able to support all three notions of time.

Past-time Data. Past-time data reflects historical location data. ILX should be able to store, update, and query historical location data. Workloads in past-time location data are mainly analytical query workloads. One good example of this category is the historical location data that is in the form of moving object trajectories that have taken place in the past. ILX should be able to handle these historical location data natively.

Current-time Data. Current-time data is continuously arriving data that reflects what is happening in the time NOW. The workload is heavy in updates to ingest all location data updates that reflect changes of objects' locations over time. The workload of current-time data is also heavy in reads in support of continuous queries (that mainly continuously probe current-time data to check current status). Thus, ILX should be able to handle current-time location data workloads that are heavy in both updates and in continuous and snapshot analytics.

Future Time. ILX should be able to support future prediction type of location data. This is useful for what-if scenarios, decision support, and prediction analytical workloads.

2.1.8 **3D and 4D Data Support Beyond GeoLocation Data.** ILX will support spatial data beyond geolocation data. For example, brain data atlases, connectivity networks, and brain simulations [47] are non-geolocation data that fall perfectly within the scope of ILX. Similarly, geolocation data contains 3D and 4D data, e.g., terrain data and simulations of flood over terrain data. ILX dimensionality should extend to support these scenarios.

2.1.9 **Visualization.** ILX will provide a suite of visualization tools that are tightly-integrated into ILX's query processing and sampling components, e.g., as in [40, 49, 109]. Visualization would support 2D, 3D, and 4D data via animations over time.

2.2 The "X" in ILX: Extensibility

Typically, location data is associated with other data types X, e.g., graphs, road networks, points of interest, social network data, click streams, text and tweets, documents, and relational data. The location engine in ILX should be extensible to introduce new data types X as needed by the driving location-service applications. Thus, extensibility for adding new data types X will be a first-class feature

in ILX. Extensibility will be at all engine levels including storage, indexing, query processing operators, and query optimization.

2.2.1 **Extensibility in Multi-model Databases.** Recently, multi-model databases have been gaining significant attention in order to address the big varieties in data applications [74]. Example multi-model databases include ArangoDB [7], OrientDB [13], BigDAWG [42], and Oracle Converged Database [4].

Current multi-model databases either do not support location data at all, or do not support it efficiently, e.g., may support location data as JSON documents, or have geometric data types without indexing support. Notably, Oracle Converged Database [4] supports location data with indexing support but is implemented on top of relational tables, which is against the vision and premise of ILX.

Multi-model database systems support multiple fixed data types within the same system. Having extensibility as a main feature in multi-model databases can be one step in the correct direction.

2.2.2 **Multi-model Data Stream Support.** Current multi-model data systems do not support data streaming. In addition to being multi-model in nature, e.g., as in [7], the multi-model ILX should also support both online streaming in addition to the offline processing of location and location+time data.

2.3 The "I" in ILX: Intelligence

Adopting Machine learning (ML) techniques in systems is a very promising direction given nowadays advances in hardware, GPUs, neural networks, deep learning, and ML software stacks. Location+X systems are no exception. Potential benefits for enabling location-first ILX systems with ML techniques are multi-fold.

2.3.1 **Enhancing over Existing Heuristics.** Many location-related problems involve heuristics that serve as approximations for NP-Complete and NP-Hard problems. Replacing these heuristic solutions with ML-based techniques is expected to produce more efficient and more accurate learning-based solutions, e.g., as in [113]. Another example is handling the dynamic nature and the change in distribution of location data and location queries over time. Reinforcement learning can be used to adapt the underlying organization and partitioning of location data to rebalance the load. One important challenge for using ML in ILX is the need for accurate yet real-time responses to location-based queries. The benefits of augmenting ML into ILX in terms of scalability, adaptivity, real-timeliness, and accuracy need to be investigated.

2.3.2 **Support for Explainability.** Explainability in AI is an important subject. For the same reasons, explainability is needed in ILX to explain why the ML-based decisions in ILX are made, and why other choices are excluded. In the broader sense, explainability is needed in ILX when choices are made. For example, when a well-known shortest path is not chosen, the user should be given feedback as to why the well-known shortest path has not been chosen, e.g., due to new construction, lane closure, accident, etc.

2.3.3 **Recommendation Operators.** ILX should support location-driven recommendations and ranking in its query language and execution engine. ILX's query language should have embedded in it personalized recommendation operators, e.g., based on location-aware *Collaborative Filtering*, to rank the query engine's responses. More generally, recommendations in ILX must be aware of the surrounding context that includes not only the locations of objects but also the time of day, the temperature, the dietary restrictions, etc. Means to automatically collect these contexts and means to incorporate user contexts in query processing and in recommendations need to be incorporated into ILX to return the most relevant and diversified results to the query issuer, e.g., as in [37, 53, 57].

3 INFRASTRUCTURE HIGHLIGHTS OF ILX

The massive sizes of location data can flood any location server with data. Thus, one of the main goals in realizing ILX is scalability. Scalability in ILX will be achieved by a multiplicity of means including adaptivity, elasticity, and the adoption of new hardware and memory platforms, e.g., main-memory and persistent memory clusters, NUMA-awareness, vectorization, and GPU query processing. ILX will adopt important query processing strategies including federated query processing, query compilation, and approximate query processing techniques, e.g., distance oracles and other essential location-related query operators. In this section, we briefly highlight these approaches and their roles in ILX.

3.1 Adaptivity, Elasticity, and Memory Disaggregation

Due to the dynamic changes in location data distributions over space and time, and the occurrence of hot spots, new servers will need to be allocated online while ILX is running. Similarly, servers will need to be dynamically deallocated from lightly loaded geospatial regions. Moreover, ILX will use disaggregated architectures (Compute servers vs. Memory servers) [114], where one can add or remove compute or memory independent of each other.

3.2 Utilizing Modern Hardware

3.2.1 **NUMA Awareness.** Multi-socket systems with non-uniform memory access architectures (NUMA) have been introduced, where each socket is equipped with multiple cores along with its own local memory, and is connected to other sockets, i.e., remote memory with interconnect links. To fully utilize these modern multi-core NUMA hardware, NUMA-aware algorithms [64, 67, 90] are continuously being developed for data systems. ILX should be designed while considering the characteristics of these multi-core NUMA architectures.

3.2.2 **Vectorization.** Vectorizing a query execution engine [89] or a standalone database operator, e.g., join [21, 24, 58], scan [115, 124], aggregation [33, 86, 117], sorting[31, 55, 56, 87, 95], Bloom filters [61] or compression [88] to utilize data parallelism has gained popularity in recent times due to the introduction of complex SIMD instructions in modern multi-core CPU platforms and the performance gain while executing database queries. VectorWise [25], DB2 BLU [92], columnar SQL Server [62], Quickstep [82] are example data systems that implement vectorization. It is only natural that ILX-based systems should benefit from vectorization and its potential can be investigated while designing vectorized location database operators for ILX.

3.2.3 **Main-memory Techniques.** There is a surge of interest in main-memory databases [39, 45, 63, 83] because of the dropping price and increasing capacity of main-memory. Thus, it is possible to keep large portions of location data in main-memory for high performance. Many location data techniques need to be revisited for main-memory and the cache hierarchy, e.g., as in [112]. Caching that is aware of location proximity is called for. Optimizing to minimize CPU cache misses while performing memory-based location operations would be critical to high performance. The main-memory location engine needs to be redesigned because of the lack of need for a buffer manager anymore. Thus, efficient location data layout in main-memory and cache-aware location-based indexes are important factors for a highly performant ILX system.

3.2.4 **Persistent Memory.** Typically, location-based data systems have been optimized for the traditional memory hierarchy: cache memory, main-memory, disk (or SSD). With the introduction of Persistent and Non-Volatile Memories, e.g., Intel Optane Persistent Memory [11], the traditional memory hierarchy has changed significantly. First, persistent memory is "persistent". Thus, there is no need for disk-like storage. Second, the speed gap between main-memory and persistent memory is much narrower than what is between main-memory and disk. Thus, location data indexes that have been optimized for disk-based memory hierarchies will need a complete redesign to fit into a persistent-memory-based memory hierarchy, e.g., as in [72, 73]. Another important factor is that the read and write speeds for persistent memory are not symmetric. Writes are multiple of times slower than Reads. Location data indexes over persistent memory need to be optimized for that.

3.2.5 **GPUs.** GPUs have a complex memory architecture with various types of memories including texture memory, which is a read-only off-chip memory with caching enabled [14]. Texture cache is specially optimized for 2D spatial locality that makes it an optimal candidate for handling location data in ILX. Designing GPU-friendly data model and algebra by capturing the geometric properties of spatial data to answer spatial queries over large data sets has been gaining popularity [41] and is in the right direction. ILX needs GPU support to naturally make use of the GPU's 2D cache memory that is quite fit for location data. Moreover, GPUs would help subsidize for the expensive geometric data operations, e.g., polygon-polygon intersections, and spatial joins. However, the issue of impedance mismatch between the GPU and CPU memory spaces still need to be addressed to avoid copying data back and forth between the two memory spaces.

3.3 Query Processing

In ILX, we will adopt several query processing strategies including (a) *Federated Query Processing* due to the variety in the input location data sources, (b) *Multi-model Query Compilation Techniques* to allow location data services and continuous location queries to execute as close to the bare bone of the underlying hardware without multiple software layers that defeat the real-time nature in performing location services, and (c) *Query Operators and Services* unique to the ILX environment including Distance Oracles, Map Matching operators, and Address Translation services. Below, we present highlights of these query processing features in ILX.

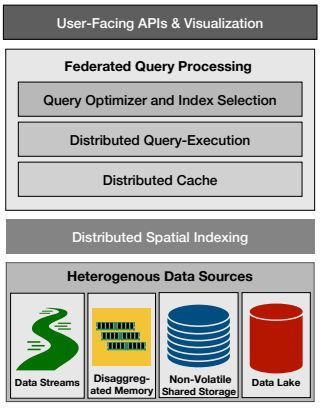

Figure 1: ILX federated query processing architecture.

3.3.1 **Federated Query Processing.** Federated query processing has been adopted by many recent systems to handle the diversity in the sources of data and be able to process queries across the multiplicity of sources. Example systems are F1 [100], Presto [96], Flink [28], and Iceberg [5]. The envisioned ILX will have a layered architecture that builds on federated query processing. Refer to Figure 1. We briefly explain the stack of layers that constitute ILX.

The user-facing layer offers several APIs for issuing the user queries and receiving the query results. The user-facing layer also offers several visualization tools for presenting the query results using visual representation. For streaming applications, the query results update the visualization in an online fashion. The query processing layer consists of three components: a) an optimizer, b) distributed execution, and c) caching. The optimizer is responsible for finding the best plan for the query, as well as finding the best spatial and relational indexes that speed up the execution of the query. The execution units (i.e., operators) of ILX are distributed. A key feature in ILX is that the data is spatially indexed and partitioned according to the spatial features of the data (be it streamed or static), and also according to the query workload distribution. This partitioning would lead to efficient distributed execution with high throughput and low latency. Moreover, the caching layer boosts the performance of repeating queries, i.e., these queries that focus on hotspot locations, e.g., downtown areas, event locations, etc., or continuous query evaluation, e.g., in support of data streaming.

ILX does not rely on a single data format for its data sources. Thus, ILX is founded on a federated query processing platform, where it supports extensible data readers and adapters that can read heterogeneous data formats from different storage and streaming sources. Moreover, ILX can operate on multiple data sources with a variety of formats. A single query can perform a join between a streaming data source and an RDF file from the data lake.

3.3.2 **Multi-model Query Compilation.** Query compilation has proven to be quite effective in enhancing the performance of database systems, e.g., [59, 77]. In contrast to producing a query

evaluation pipeline that the query interpreter executes one-tuple-at-a-time or a vector of tuples at a time, in case vectorization is used, in query compilation, low level C code can be generated and compiled to execute the query. Query compilation eliminates the software interpretation layer, and results in executing the query as close as possible to the bare metal of the hardware.

There has been good efforts in compiling queries that involve spatial predicates, e.g., [107]. However, given the multi-model nature in ILX, query compilation needs to be extended to cover multi-model queries that access , e.g., location, text/JSON, relations, and graph traversal operations.

3.3.3 **Query Operators and Services.** At the core of ILX is a set of unique location-related operators that cater to the unique features of location query processing in ILX. These operators include *Distance Oracle* operators, *Map Matching* operators, and *Address Translation* services. We describe each one briefly below. Notice that some of these operators are offered by service providers, e.g., Google Maps GeoLocation APIs [9]. However, they are not open-source, and are provided at a pay-as-you-go pricing model [10].

Distance Oracles. Distance oracles offer a fast means for computing shortest distance in road networks, e.g., [48, 93, 94]. Based on the amount of storage allowed for preprocessing, they provide a spectrum of approximate solutions with various error bounds (including 0 error). Distance oracles are an integral component for scalability in ILX's query processor. However, current distance oracle technology will need to be extended to allow for operating on arbitrary subsets of the road network, e.g., when a subset of the roads is dynamically selected, e.g., via querying, and then a shortest path computation is required on the selected subset.

Map Matching Operators. A core operator in ILX is the map-matching operator. It maps the physical location of an object to a logical location on the map. Given a road network, say RN, and the physical location of an object O, e.g., O's longitude, latitude from a GPS reading, say $L_O(long, lat)$, the map matching operator returns from RN, the logical location of the object on the map, e.g., the identifier of the road (edge), the intersection (vertex), or the textual address that O *most likely* lies in. Notice that there is the possibility of transient errors in the map matching operator due to the inaccuracy in the GPS measurement devices and the misalignment and misregistration of the underlying maps into physical space. Also, the errors depend on whether the map matching operation is performed online or offline. In the case of offline map matching, the entire trajectory of the object is present, and hence it should be more accurate to predict the location of an object at any given point in time. In contrast, in the case of online map matching, only the current and past locations of the object are available for the map matching operator to decide on the logical location on the map of an object at current time. Hence, in the online case, the map matching operator is prone to more errors. The map matching operator is commonly used in GPS devices to display the location of the object on the logical map and to help with the vehicle navigation process using the logical map as a guide. It is anticipated that ILX will also make heavy use of this operator at both the query processing and optimization levels. Many useful map matching operators exist that we plan to utilize and build on from within ILX, e.g., [26, 71, 78, 80, 121].

Address Translation Operators. Another important and useful building block for query processing and optimization in ILX is the address translation operator. This operator is the inverse of the map matching operator. Given a textual address input, this operator returns the address's corresponding longitude and latitude.

The distance oracle, map matching, and address translation operations will be used extensively in query processing within ILX.

3.4 Location-based Access Methods

3.4.1 **Clustered Location Data Indexes.** Access methods and indexes for location data are essential components in ILX. However, with the location data type being a first-class citizen in ILX, location data indexes need to become the primary storage methods and clustered indexes that host all the other types of data in addition to the location data. For example, if ILX has a quad-tree index to store the coordinates for a point data set, the same quad-tree could serve as a clustered index that also stores the entire description of the point data objects, e.g., the city names, the city population, etc., inside the index. Additional indexing methods will be based on the types X associated with the location data. However, clustering of data will be location-driven.

3.4.2 **Update-Intensive Indexing Techniques.** The continuous move and change in location of objects in space over time results in an update-intensive workload. Thus, an important feature in location access methods is the support for update-intensive indexing. Techniques exist for handling frequent updates in location indexes, e.g., [103]. However, they need to be extended to support (1) memory-based location indexes, (2) become cache- and NUMA-aware, and (3) be optimized for disaggregated memory. Disaggregation has become feasible and practical due to the successful use of high-speed remote direct memory access (RDMA) over the network [114]. Location data indexes need to be adapted to support the disaggregated architecture over RDMA.

3.4.3 **LSM-based Location Indexes.** LSM indexes [79] are optimized for write-intensive key-value workloads. Because location serves as a secondary key, to be effective, LSM indexes need to be adapted in support of update-intensive secondary-key location-data workloads [99].

3.4.4 **Location-based Learned Indexes.** Machine Learning (ML) techniques have been applied successfully to build various types of learned indexes [60]. It has been extended to the multi-dimensional case, e.g., [15, 16]. Learned indexes have shown potential in terms of smaller index size and faster performance in contrast to traditional indexes. Learned indexes work well for static data sets as training of the learned models take place in a preprocessing phase. Realizing learned indexes for dynamic data sets has been a challenge due to the need to continuously retrain the models. There are some very successful attempts to deal with dynamic data in the multidimensional case. Of mention are LISA [66] and RSMI [91]. ILX needs to adopt similar ideas, and extend these learned indexes to accommodate the time dimension to be able to handle real-time trajectory data.

3.5 Concurrency Control, Integrity, and Fault Tolerance

Concurrency control plays a critical role in ILX to coordinate concurrent read and write operations for scalability. Many existing concurrency control protocols in spatial databases are based on locking data objects (e.g., [29, 30, 36, 104]). Two possible approaches can be explored in ILX. First, in contrast to locking data objects, ILX can consider locking the underlying physical space or specific locations in space under the premise that no two objects can share the underlying physical space at the same time. In contrast to data-driven locking, locking physical locations can resemble *space-driven locking* in location data indexes that have disjoint space-driven partitioning of the underlying space. Two issues remain to be addressed for this approach to be credible: (1) Handle consistently the issue of multi-granularity locking in the physical space and (2) Handle the issue of location uncertainty. If the location of an object is uncertain or is not measured precisely, then locking of physical locations may not have one-to-one correspondence with the locations of the objects as stored within ILX or within ILX's location data indexes. This may introduce overlaps in potential locations of where objects might be in space. More research is needed to address the issue of uncertainty in conjunction with physical location locking and the location overlaps it introduces.

The second approach that needs to be explored in ILX is to adopt concurrency control techniques that can scale to hundreds and thousands of cores [22, 120]. It is important to design lock-free concurrency control for spatial access methods along the same lines as the lock-free B-tree (the Bw-tree [65]).

Finally, the new infrastructure that ILX will be deployed in poses additional challenges for concurrency control. For example, in the RDMA-enabled disaggregated memory architecture [27, 114], it is non-trivial to lock the remote objects using RDMA primitives, and hence existing concurrency control protocols need to be revisited.

ILX should be able to tolerate faults, e.g., via replication. ILX should be able to recover its indexes if they get partially or completely lost or damaged due upon faulting. Recovering from faults are to be performed online without system shutdown and while guaranteeing correctness of the system operation e.g., during online repartitioning of data, ILX should guarantee that no data gets lost and no data is reported twice as part of an answer to a query.

3.6 Location Data Compression

Data compression is an important technique especially for spatial databases due to the huge amount of location data. It not only can save memory but also can improve query time due to the smaller data sizes being retrieved. Compression is highly under-studied in spatial databases [32, 69]. It requires a systematic study of compression techniques for both location data and location data indexes. Although there are some compression algorithms for floating-point data [68, 84], it is not clear how they perform on location data because these algorithms usually work well on specific data distributions. For location indexes, e.g., the R-tree, it is important to compress the structural information, similar to B-tree structural compression [23, 70]. Another important design consideration is to support query processing on compressed data and indexes, which will improve the performance. More research is needed to evaluate the impact to compression ratio.

3.7 Semantics and RDF-based Location Data

Many geospatial datasets are part of the Web of Data. Several geospatial extensions to the SPARQL query language have been introduced to query and reason over geospatial semantic data. ILX should be able to natively store and reason over geospatial RDF data. It is important for ILX to handle the slight geo-semantic inaccuracies, e.g., the predicate "north-of" can roughly describe objects that are slightly towards the northeast direction. ILX should be able to reason over location data given these semantic ambiguities. Moreover, ILX should be able to make use of the interlinked topological relations in the Linked Open Data cloud (LOD), and help produce new geospatial interlinks progressively in LOD as a side effect, e.g., as in [81].

3.8 Security and Resilience to Attacks

ILX should be resilient to malicious activities, e.g., attacks to stop the system, alter, or snoop data. Systems that use dynamic load balancing mechanisms are vulnerable to malicious attacks [35]. This type of attack affects system availability. Attackers can make the system in continuous state of rebalancing. Other types of attacks that can affect ILX need to be investigated, e.g., faking the location of data, hiding the detection of important location data by flooding the system with irrelevant data in the same location. ILX should be resilient to these attacks by having intelligence to detect and block malicious users. It should analyze user behavior as individuals and as groups to detect and prevent any malicious activities.

3.9 Useful EcoSystem Tools

Various geometrical and spatiotemporal toolkits and libraries exist, e.g., [1–3], that can be partly useful for the ILX ecosystem. Also, location data generators, e.g., [76], would be an integral part of the ILX ecosystem.

4 SUMMARY

This paper highlights the main features and challenges in realizing ILX-like systems. Several existing research works follow some aspects of the ILX vision, and hence are in the right direction. Due to space limitation, not all of these research works are cited in this paper. However, this paper helps identify such works.

Benchmarks for testing and tuning the performance of all of ILX's features will be an integral part of ILX's ecosystem. Many such benchmarks already exist in the literature. However, once ILX is realized, targeted micro-benchmarks for specific features of ILX will need to be developed.

5 ACKNOWLEDGEMENTS

The authors acknowledge the support of the U.S. National Science Foundation under Grant Numbers III-1815796 and IIS-1910216.

REFERENCES

[1] [n.d.]. H3: Uber's Hexagonal Hierarchical Spatial Index. https://eng.uber.com/h3/.
[2] [n.d.]. The LocationTech Technology (LTT). https://projects.eclipse.org/projects/locationtech.
[3] [n.d.]. S2 Geometry. https://s2geometry.io.
[4] 2020. Oracle's Converged Database: How to Make Developers And Data More Productive. https://www.oracle.com/a/otn/docs/databas/oracle-converge database-technicalbrief.pdf.
[5] 2022. Apache Iceberg. https://www.dremio.com/resources/guides/apache-iceberg-an-architectural-look-under-the-covers/.
[6] 2022. Apache Storm. https://storm.apache.org/.
[7] 2022. ArangoDB. https://www.arangodb.com/.
[8] 2022. General Data Protection Regulation (GDPR). https://gdpr-info.eu/art-1-gdpr.
[9] 2022. GeoLocation API. https://developers.google.com/maps/documentation/geolocation/overview.
[10] 2022. GeoLocation API: Usage and Billing. https://developers.google.com/maps/documentation/geolocation/usage-and-billing.
[11] 2022. Intel Optane Persistent Memory. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.
[12] 2022. Oracle Spatial. https://www.oracle.com/database/spatial/.
[13] 2022. OrientDB. https://orientdb.org/.
[14] 2022. Texture Memory in GPU. https://docs.nvidia.com/gameworks/content/developertools/desktop/analysis/report/cudaexperiments/kernellevel/memorystatisticstexture.htm.
[15] Abdullah Al-Mamun, Ch. Md. Rakin Haider, Jianguo Wang, and Walid G. Aref. 2022. The "AI+R"-tree: An Instance-optimized R-tree. In *MDM*.
[16] Abdullah Al-Mamun, Hao Wu, and Walid G Aref. 2020. A tutorial on learned multi-dimensional indexes. In *SIGSPATIAL*. 1–4.
[17] Ahmed M. Aly, Ahmed R. Mahmood, Mohamed S. Hassan, Walid G. Aref, Mourad Ouzzani, Hazem Elmeleegy, and Thamir Qadah. 2015. AQWA: Adaptive Query-Workload-Aware Partitioning of Big Spatial Data. *PVLDB* 8, 13 (2015), 2062–2073.
[18] Walid G. Aref. 2017. Location + X Big Data Systems: Challenges and Some Solutions. In *GeoRich*.
[19] Walid G. Aref. 2019. The Dos and Don'ts of Spatial+X Data Management: A "Systems Perspective". In *MDM*.
[20] Walid G. Aref and Ihab F. Ilyas. 2001. SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees. *J. Intell. Inf. Syst.* 17, 2-3 (2001), 215–240.
[21] Cagri Balkesen, Jens Teubner, Gustavo Alonso, and M. Tamer Özsu. 2013. Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware. In *ICDE*. 362–373.
[22] Tiemo Bang, Norman May, Ilia Petrov, and Carsten Binnig. 2020. The tale of 1000 Cores: an evaluation of concurrency control on real(ly) large multi-socket hardware. In *DaMoN*. 3:1–3:9.
[23] Rudolf Bayer and Karl Unterauer. 1977. Prefix B-Trees. *TODS* 2, 1 (1977), 11–26.
[24] Spyros Blanas, Yinan Li, and Jignesh M. Patel. 2011. Design and evaluation of main memory hash join algorithms for multi-core CPUs. In *SIGMOD*. 37–48.
[25] Peter A. Boncz, Marcin Zukowski, and Niels Nes. 2005. MonetDB/X100: Hyper-Pipelining Query Execution. In *CIDR*. 225–237.
[26] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. 2005. On Map-Matching Vehicle Tracking Data. In *PVLDB*. 853–864.
[27] Wei Cao, Yingqiang Zhang, Xinjun Yang, Feifei Li, Sheng Wang, Qingda Hu, Xuntao Cheng, Zongzhi Chen, Zhenjun Liu, Jing Fang, Bo Wang, Yuhui Wang, Haiqing Sun, Ze Yang, Zhushi Cheng, Sen Chen, Jian Wu, Wei Hu, Jianwei Zhao, Yusong Gao, Songlu Cai, Yunyang Zhang, and Jiawang Tong. 2021. PolarDB Serverless: A Cloud Native Database for Disaggregated Data Centers. In *SIGMOD*. 2477–2489.
[28] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink™: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.* 38, 4 (2015), 28–38.
[29] Kaushik Chakrabarti and Sharad Mehrotra. 1999. Efficient Concurrency Control in Multidimensional Access Methods. In *SIGMOD*. 25–36.
[30] Natalia Chaudhry and Muhammad Murtaza Yousaf. 2022. Concurrency control for real-time and mobile transactions: Historical view, challenges, and evolution of practices. *Concurr. Comput. Pract. Exp.* 34, 3 (2022).
[31] Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, William Macy, Mostafa Hagog, Yen-Kuang Chen, Akram Baransi, Sanjeev Kumar, and Pradeep Dubey. 2008. Efficient implementation of sorting on multi-core SIMD CPU architecture. *PVLDB* 1, 2 (2008), 1313–1324.
[32] Peter Chovanec, Michal Krátký, and Jiří Walder. 2010. Lossless R-tree compression using variable-length codes. In *ICITST*. 1–8.
[33] John Cieslewicz, Kenneth A. Ross, Kyoho Satsumi, and Yang Ye. 2010. Automatic contention detection and amelioration for data-intensive operations. In *SIGMOD*. 483–494.
[34] Anas Daghistani, Walid G. Aref, Arif Ghafoor, and Ahmed R. Mahmood. 2021. SWARM: Adaptive Load Balancing in Distributed Streaming Systems for Big Spatial Data. *ACM TSAS* 7, 3 (2021), 14:1–14:43.
[35] Anas Daghistani, Mosab Khayat, Muhamad Felemban, Walid G Aref, and Arif Ghafoor. 2021. Guard: Attack-Resilient Adaptive Load Balancing in Distributed Streaming Systems. *IEEE TDSC* (2021).
[36] Jing (David) Dai and Chang-Tien Lu. 2017. Concurrency Control for Spatial Access. In *Encyclopedia of GIS*. 285–286.
[37] Florian Daniel, Maristella Matera, Elisa Quintarelli, Letizia Tanca, and Vittorio Zaccaria. 2018. Context-Aware Access to Heterogeneous Resources Through On-the-Fly Mashups. In *CAiSE*. 119–134.
[38] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System. In *CIDR*.
[39] Cristian Diaconu, Craig Freedman, Erik Ismert, Per-Åke Larson, Pravin Mittal, Ryan Stonecipher, Nitin Verma, and Mike Zwilling. 2013. Hekaton: SQL Server's Memory-Optimized OLTP Engine. In *SIGMOD*. 1243–1254.
[40] Liming Dong, Qiushi Bai, Taewoo Kim, Taiji Chen, Weidong Liu, and Chen Li. 2020. Marviq: Quality-Aware Geospatial Visualization of Range-Selection Queries Using Materialization. In *SIGMOD*. 67–82.
[41] Harish Doraiswamy and Juliana Freire. 2020. A GPU-friendly Geometric Data Model and Algebra for Spatial Queries. In *SIGMOD*. 1875–1885.
[42] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stanley B. Zdonik. 2015. The BigDAWG Polystore System. *SIGMOD Record* 44, 2 (2015), 11–16.
[43] Ahmed Eldawy and Mohamed F. Mokbel. 2015. SpatialHadoop: A MapReduce Framework for Spatial Data. In *ICDE*. 1352–1363.
[44] Mohamed Y. Eltabakh, Walid G. Aref, Ahmed K. Elmagarmid, and Mourad Ouzzani. 2014. HandsOn DB: Managing Data Dependencies Involving Human Actions. *IEEE TKDE*. 26, 9 (2014), 2193–2206.
[45] Franz Faerber, Alfons Kemper, Per-Åke Larson, Justin J. Levandoski, Thomas Neumann, and Andrew Pavlo. 2017. Main Memory Database Systems. *Foundations and Trends in Databases* 8, 1-2 (2017), 1–130.
[46] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *ICDE*. 1001–1012.
[47] Richard Frackowiak, Anastasia Ailamaki, and Ferath Kherif. 2016. Federating and Integrating What We Know About the Brain at All Scales: Computer Science Meets the Clinical Neurosciences. (2016), 157–170.
[48] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *WEA*.
[49] Tao Guo, Kaiyu Feng, Gao Cong, and Zhifeng Bao. 2018. Efficient Selection of Geospatial Data on Maps for Interactive and Visualized Exploration. In *SIGMOD*. 567–582.
[50] Mohamed S. Hassan, Tatiana Kuznetsova, Hyun Chai Jeong, Walid G. Aref, and Mohammad Sadoghi. 2018. Extending In-Memory Relational Database Engines with Native Graph Support. In *EDBT*. 25–36.
[51] Mohamed S. Hassan, Tatiana Kuznetsova, Hyun Chai Jeong, Walid G. Aref, and Mohammad Sadoghi. 2018. GRFusion: Graphs as First-Class Citizens in Main-Memory Relational Database Systems. In *SIGMOD*. 1789–1792.
[52] Daeyoung Hong, Woohwan Jung, and Kyuseok Shim. 2022. Collecting Geospatial Data Under Local Differential Privacy With Improving Frequency Estimation. *IEEE TKDE* (2022), 1–12.
[53] Saeid Hosseini, Hongzhi Yin, Meihui Zhang, Xiaofang Zhou, and Shazia Sadiq. 2016. Jointly Modeling Heterogeneous Temporal Properties in Location Recommendation. In *DASFAA*. 490–506.
[54] Yan Huang, Shashi Shekhar, and Hui Xiong. 2004. Discovering Colocation Patterns from Spatial Data Sets: A General Approach. *IEEE TKDE* 16, 12 (2004), 1472–1485.
[55] Hiroshi Inoue, Takao Moriyama, Hideaki Komatsu, and Toshio Nakatani. 2007. AA-Sort: A New Parallel Sorting Algorithm for Multi-Core SIMD Processors. In *PACT*. 189–198.
[56] Hiroshi Inoue and Kenjiro Taura. 2015. SIMD- and Cache-Friendly Algorithm for Sorting an Array of Structures. *PVLDB* 8, 11 (2015), 1274–1285.
[57] Georgios Kalamatianos, Georgios John Fakas, and Nikos Mamoulis. 2021. Proportionality in Spatial Keyword Search. In *SIGMOD*. 885–897.
[58] Changkyu Kim, Eric Sedlar, Jatin Chhugani, Tim Kaldewey, Anthony D. Nguyen, Andrea Di Blas, Victor W. Lee, Nadathur Satish, and Pradeep Dubey. 2009. Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs. *PVLDB* 2, 2 (2009), 1378–1389.
[59] Yannis Klonatos, Christoph Koch, Tiark Rompf, and Hassan Chafi. 2014. Building Efficient Query Engines in a High-Level Language. *PVLDB* 7, 10 (2014), 853–864.
[60] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *SIGMOD*. 489–504.
[61] Harald Lang, Thomas Neumann, Alfons Kemper, and Peter A. Boncz. 2019. Performance-Optimal Filtering: Bloom overtakes Cuckoo at High-Throughput. *PVLDB* 12, 5 (2019), 502–515.
[62] Per-Åke Larson, Cipri Clinciu, Eric N. Hanson, Artem Oks, Susan L. Price, Srikumar Rangarajan, Aleksandras Surna, and Qingqing Zhou. 2011. SQL server

column store indexes. In *SIGMOD*. 1177–1184.
[63] Juchang Lee, Yong Sik Kwon, Franz Färber, Michael Muehle, Chulwon Lee, Christian Bensberg, Joo-Yeon Lee, Arthur H. Lee, and Wolfgang Lehner. 2013. SAP HANA Distributed In-memory Database System: Transaction, Session, and Metadata Management. In *ICDE*. 1165–1173.
[64] Viktor Leis, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2014. Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age. In *SIGMOD*. 743–754.
[65] Justin J. Levandoski, David B. Lomet, and Sudipta Sengupta. 2013. The Bw-Tree: A B-tree for new hardware platforms. In *ICDE*. 302–313.
[66] Pengfei Li, Hua Lu, Qian Zheng, Long Yang, and Gang Pan. 2020. LISA: A Learned Index Structure for Spatial Data. In *SIGMOD*. 2119–2133.
[67] Yinan Li, Ippokratis Pandis, René Müller, Vijayshankar Raman, and Guy M. Lohman. 2013. NUMA-aware algorithms: the case of data shuffling. In *CIDR*.
[68] Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. 2022. Chimp: Efficient Lossless Floating Point Compression for Time Series Databases. *PVLDB* 15 (2022).
[69] Hung-Yi Lin and Shih-Ying Chen. 2008. High Indexing Compression for Spatial Databases. In *CIT Workshops*. 20–25.
[70] David B. Lomet. 2001. The Evolution of Effective B-tree: Page Organization and Techniques: A Personal Account. *SIGMOD Rec.* 30, 3 (2001), 64–69.
[71] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. 2009. Map-matching for low-sampling-rate GPS trajectories. In *SIGSPATIAL*. 352–361.
[72] Baotong Lu, Jialin Ding, Eric Lo, Umar Farooq Minhas, and Tianzheng Wang. 2021. APEX: A High-Performance Learned Index on Persistent Memory. *PVLDB* 15, 3 (2021), 597–610.
[73] Baotong Lu, Xiangpeng Hao, Tianzheng Wang, and Eric Lo. 2020. Dash: Scalable Hashing on Persistent Memory. *PVLDB* 13, 8 (2020), 1147–1161.
[74] Jiaheng Lu and Irena Holubová. 2019. Multi-model Databases: A New Journey to Handle the Variety of Data. *ACM Computing Surveys (CSUR)* 52, 3 (2019), 55:1–55:38.
[75] Ahmed R. Mahmood, Ahmed M. Aly, Thamir Qadah, El Kindi Rezig, Anas Daghistani, Amgad Madkour, Ahmed S. Abdelhamid, Mohamed S. Hassan, Walid G. Aref, and Saleh M. Basalamah. 2015. Tornado: A Distributed Spatio-Textual Stream Processing System. *PVLDB* 8, 12 (2015), 2020–2023.
[76] Mohamed F. Mokbel, Louai Alarabi, Jie Bao, Ahmed Eldawy, Amr Magdy, Mohamed Sarwat, Ethan Waytas, and Steven Yackel. 2013. MNTG: An Extensible Web-Based Traffic Generator. In *SSTD*. 38–55.
[77] T. Neumann. 2011. Efficiently Compiling Efficient Query Plans for Modern Hardware. *PVLDB* 4, 9 (2011), 539–550.
[78] Paul Newson and John Krumm. 2009. Hidden Markov map matching through noise and sparseness. In *SIGSPATIAL*. 336–343.
[79] Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth J. O'Neil. 1996. The Log-Structured Merge-Tree (LSM-Tree). *Acta Informatica* 33, 4 (1996), 351–385.
[80] Takayuki Osogami and Rudy Raymond. 2013. Map Matching with Inverse Reinforcement Learning. In *IJCAI*. 2547–2553.
[81] George Papadakis, George Mandilaras, Nikos Mamoulis, and Manolis Koubarakis. 2022. Static and Dynamic Progressive Geospatial Interlinking. *ACM TSAS* 8, 2 (2022).
[82] Jignesh M. Patel, Harshad Deshmukh, Jianqiao Zhu, Navneet Potti, Zuyu Zhang, Marc Spehlmann, Hakan Memisoglu, and Saket Saurabh. 2018. Quickstep: A Data Platform Based on the Scaling-Up Approach. *PVLDB* 11, 6 (2018), 663–676.
[83] Jignesh M. Patel, Harshad Deshmukh, Jianqiao Zhu, Navneet Potti, Zuyu Zhang, Marc Spehlmann, Hakan Memisoglu, and Saket Saurabh. 2018. Quickstep: A Data Platform Based on the Scaling-Up Approach. *PVLDB* 11, 6 (2018), 663–676.
[84] Tuomas Pelkonen, Scott Franklin, Paul Cavallaro, Qi Huang, Justin Meza, Justin Teller, and Kaushik Veeraraghavan. 2015. Gorilla: A Fast, Scalable, In-Memory Time Series Database. *PVLDB* 8, 12 (2015), 1816–1827.
[85] Shangfu Peng and Hanan Samet. 2015. Analytical Queries on Road Networks: An Experimental Evaluation of Two System Architectures. In *SIGSPATIAL*.
[86] Orestis Polychroniou and Kenneth A. Ross. 2013. High throughput heavy hitter aggregation for modern SIMD processors. In *DaMoN*. 6.
[87] Orestis Polychroniou and Kenneth A. Ross. 2014. A comprehensive study of main-memory partitioning and its application to large-scale comparison- and radix-sort. In *SIGMOD*. 755–766.
[88] Orestis Polychroniou and Kenneth A. Ross. 2015. Efficient Lightweight Compression Alongside Fast Scans. In *DaMoN*. 9:1–9:6.
[89] Orestis Polychroniou and Kenneth A. Ross. 2019. Towards Practical Vectorized Analytical Query Engines. In *DaMoN*. 10:1–10:7.
[90] Iraklis Psaroudakis, Tobias Scheuer, Norman May, Abdelkader Sellami, and Anastasia Ailamaki. 2016. Adaptive NUMA-aware data placement and task scheduling for analytical workloads in main-memory column-stores. *PVLDB* 10, 2 (2016), 37–48.
[91] Jianzhong Qi, Guanli Liu, Christian S Jensen, and Lars Kulik. 2020. Effectively learning spatial indices. *PVLDB* 13, 12 (2020), 2341–2354.
[92] Vijayshankar Raman, Gopi K. Attaluri, Ronald Barber, Naresh Chainani, David Kalmuk, Vincent KulandaiSamy, Jens Leenstra, Sam Lightstone, Shaorong Liu, Guy M. Lohman, Tim Malkemus, René Müller, Ippokratis Pandis, Berni Schiefer, David Sharpe, Richard Sidle, Adam J. Storm, and Liping Zhang. 2013. DB2 with BLU Acceleration: So Much More than Just a Column Store. *PVLDB* 6, 11 (2013), 1080–1091.
[93] Peter Sanders and Dominik Schultes. 2005. Highway Hierarchies Hasten Exact Shortest Path Queries. In *ESA*. 568–579.
[94] Jagan Sankaranarayanan and Hanan Samet. 2009. Distance Oracles for Spatial Networks. In *ICDE*. 652–663.
[95] Nadathur Satish, Changkyu Kim, Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, Daehyun Kim, and Pradeep Dubey. 2010. Fast sort on CPUs and GPUs: a case for bandwidth oblivious SIMD sort. In *SIGMOD*. 351–362.
[96] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, and Christopher Berner. 2019. Presto: SQL on Everything. In *ICDE*. 1802–1813.
[97] Sina Shaham, Gabriel Ghinita, Ritesh Ahuja, John Krumm, and Cyrus Shahabi. 2021. HTF: Homogeneous Tree Framework for Differentially-Private Release of Location Data. In *SIGSPATIAL*. 184–194.
[98] Shashi Shekhar, Chang-Tien Lu, and Pusheng Zhang. 2003. A Unified Approach to Detecting Spatial Outliers. *GeoInformatica* 7, 2 (2003), 139–166.
[99] Jaewoo Shin, Jianguo Wang, and Walid G. Aref. 2021. The LSM RUM-Tree: A Log Structured Merge R-Tree for Update-intensive Spatial Workloads. In *ICDE*. 2285–2290.
[100] Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, John Cieslewicz, Ian Rae, Traian Stancescu, and Himani Apte. 2013. F1: A Distributed SQL Database That Scales. *PVLDB* 6, 11 (2013), 1068–1079.
[101] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *MSST*. 1–10.
[102] Yasin N. Silva, Walid G. Aref, Per-Åke Larson, Spencer Pearson, and Mohamed H. Ali. 2013. Similarity queries: their conceptual evaluation, transformations, and processing. *VLDB J.* 22, 3 (2013), 395–420.
[103] Yasin N. Silva, Xiaopeng Xiong, and Walid G. Aref. 2009. The RUM-tree: supporting frequent updates in R-trees using memos. *VLDB J.* 18, 3 (2009), 719–738.
[104] Seok Il Song, Young Ho Kim, and Jae Soo Yoo. 2004. An Enhanced Concurrency Control Scheme for Multidimensional Index Structures. *IEEE TKDE* 16, 1 (2004), 97–111.
[105] Michael Stonebraker and Lawrence A. Rowe. 1986. The Design of POSTGRES. In *SIGMOD*. 340–355.
[106] Michael Stonebraker and Ariel Weisberg. 2013. The VoltDB Main Memory DBMS. *IEEE Data Eng. Bull.* 36 (2013), 21–27.
[107] Ruby Y. Tahboub and Tiark Rompf. 2016. On supporting compilation in spatial query engines: (vision paper). In *SIGSPATIAL*. 9:1–9:4.
[108] MingJie Tang, Yongyang Yu, Qutaibah M. Malluhi, Mourad Ouzzani, and Walid G. Aref. 2016. LocationSpark: A Distributed In-Memory Data Management System for Big Spatial Data. *PVLDB* 9, 13 (2016), 1565–1568.
[109] Wenbo Tao, Xiaoyu Liu, Yedi Wang, Leilani Battle, Çagatay Demiralp, Remco Chang, and Michael Stonebraker. 2019. Kyrix: Interactive Pan/Zoom Visualizations at Scale. *Comput. Graph. Forum* 38, 3 (2019), 529–540.
[110] Ignacio G. Terrizzano, Peter M. Schwarz, Mary Roth, and John E. Colino. 2015. Data Wrangling: The Challenging Yourney from the Wild to the Lake. In *CIDR*.
[111] Hien To, Cyrus Shahabi, and Li Xiong. 2018. Privacy-Preserving Online Task Assignment in Spatial Crowdsourcing with Untrusted Server. In *ICDE*. 833–844.
[112] Dimitrios Tsitsigkos, Panagiotis Bouros, Nikos Mamoulis, and Manolis Terrovitis. 2019. Parallel In-Memory Evaluation of Spatial Joins. In *SIGSPATIAL*. 516–519.
[113] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2021. A Learned Query Optimizer for Spatial Join. In *SIGSPATIAL*. 458–467.
[114] Ruihong Wang, Jianguo Wang, Stratos Idreos, M. Tamer Özsu, and Walid G. Aref. 2022. The Case for Distributed Shared-Memory Databases with RDMA-Enabled Memory Disaggregation. *CoRR* abs/2207.03027 (2022).
[115] Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, Hasso Plattner, Alexander Zeier, and Jan Schaffner. 2009. SIMD-Scan: Ultra Fast in-Memory Table Scan using on-Chip Vector Processing Units. *PVLDB* 2, 1 (2009), 385–394.
[116] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient In-Memory Spatial Analytics. In *SIGMOD*. 1071–1085.
[117] Yang Ye, Kenneth A. Ross, and Norases Vesdapunt. 2011. Scalable aggregation on multicore processors. In *DaMoN*. 1–9.
[118] Jin Soung Yoo, Shashi Shekhar, and Mete Celik. 2005. A Join-Less Approach for Co-Location Pattern Mining: A Summary of Results. In *ICDM*. 813–816.
[119] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2015. GeoSpark: a cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL*. 70:1–70:4.
[120] Xiangyao Yu, George Bezerra, Andrew Pavlo, Srinivas Devadas, and Michael Stonebraker. 2014. Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores. *PVLDB* 8, 3 (2014), 209–220.
[121] Jing Yuan, Yu Zheng, Chengyang Zhang, Xing Xie, and Guangzhong Sun. 2010. An Interactive-Voting Based Map Matching Algorithm. In *MDM*. 43–52.
[122] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *HotCloud*.

[123] Sepanta Zeighami, Ritesh Ahuja, Gabriel Ghinita, and Cyrus Shahabi. 2022. A Neural Database for Differentially Private Spatial Range Queries. *PVLDB* 15, 5 (2022), 1066–1078.

[124] Jingren Zhou and Kenneth A. Ross. 2002. Implementing database operations using SIMD instructions. In *SIGMOD*. 145–156.