On Native Location-Optimized Data Systems

Walid G. Aref
Department of Computer Science
Purdue University
West Lafayette, USA
aref@purdue.edu

***Abstract*—With the ubiquity of location detection devices and location services and the massive amounts of the location data being produced, there is dire need for developing highly scalable location data systems. Currently, location data is usually supported as an afterthought in existing data systems, and hence these systems are not optimized natively for location data handling. This short paper calls for designing systems with location data being a first-class citizen, and highlights several important aspects that characterize location-optimized data systems. These include deterministic performance, supporting write-optimization, update tolerance, having adaptive location indexes and adaptive data stores, and having tight integration with the graph, document, relational and other multi data models for optimized performance.**

***Index Terms*—location data systems, indexing, location services, deterministic performance, adaptive techniques**

I. Introduction

Location-data systems store and process the locations of objects as they move in space. They are popular due to the ubiquity of smart-phones and location-detection devices, and they offer a broad variety of location-based services [12], [16].

Current location data systems are not optimized for handling location data. Usually, location data is supported by extending existing relational or NoSQL systems, e.g., [1], [6], [7] that have been originally designed and optimized with other goals and objectives in mind. Typically, relational or NoSQL data systems get extended with spatial or spatiotemporal indexes, some spatial query operators, e.g., range or k-Nearest-neighbor operators, and higher-level syntactic sugar to support location data. The ubiquity of location data and location data services calls for systems that are solely designed and optimized for the efficient support of location data.

This short paper advocates for native location-optimized data systems. These systems are designed with the sole objective to support location data as first-class citizens. These systems can be termed location-optimized or location-first data systems. The rest of this paper highlights several important features that location-optimized data systems should support.

II. Deterministic Performance

Location-optimized data systems should offer deterministic performance. Because of the real-time nature of location-optimized systems, location-data systems should provide deterministic performance during query processing, during index operations, and during concurrency control among all other aspects of data management systems. For example, at times, some index insert operation may involve splitting an index node, while at other times, the same index operation may not involve node splitting. This is common, e.g., in B-trees [5], and in R-trees [9] and their variants. When inserting a data item into a leaf node, two scenarios may take place.

(1) If the leaf node is not full, then the data item to be inserted is placed into the leaf node, and the insertion is complete. However, in some cases, e.g., in the R-tree case, the bounding rectangle of that leaf node may need to be expanded, and this may result in expanding the bounding boxes of the ancestor nodes of this leaf node.

(2) If the leaf node is full, it has to split, and additional time has to be spent, e.g., deciding which data items in the R-tree's original leaf node would go to the newly formed first or second children. Also, a new pointer has to be inserted into the parent node, which may result in the parent node's being split as well, and this may propagate up to the root node, and may even result in an increase in the R-tree height.

The above two scenarios highlight discrepancy in the cost of an insert operation between the two scenarios. This demonstrates non-deterministic performance.

Another form of non-determinism is that the same query may require different execution times if executed twice under the same conditions. It has been shown that both the B-tree and the R-tree indexes exhibit non-deterministic query execution time due to what is referred to as the waves of misery phenomenon [8], [18]. Unless mitigated properly, the execution time may increase for the same query if executed at two different times (without experiencing increase in the tree height, for example) [18]. The same non-determinism behavior can be observed in the index delete and update operations.

While the examples above highlight the presence of non-deterministic performance in the context of data indexing, other data systems' components exhibit non-determinism in their performance that is manifested in a variety of ways.

Query processing is another example. One aspect of deterministic performance in location data query processing is that the rate of producing output tuples in a query evaluation pipeline needs to be deterministic. For example, consider nested-loops joins or table-scan-based selects. While some output select or join tuples can be produced fast, e.g., if the qualifying tuples appear early in the table, other output tuples may take significantly more time until they are pro-

Walid G. Aref acknowledges the support of the National Science Foundation under Grant Number: IIS-1910216.

duced, hence exhibiting non-deterministic query processing performance, and non-deterministic output rate. Thus, the algorithms for the building block operators for relational algebra (e.g., selects, joins, group-by, etc.) need to be reworked for deterministic behavior. The same is true for the more complex query evaluation pipelines that are formed by composing these building block operators. Using ML-based approaches, ML-for-DB techniques, and extending relational algebra and query processing with ML inference are promising directions that can be targeted to mitigate the issue of non-deterministic query execution. For example, the works in [15] and [10] illustrate promising steps in this direction.

Generally, location-optimized data systems need to be designed at all layers of the system stack with deterministic performance in mind, including how the data is organized, indexing, query processing techniques, concurrency control, and recovery mechanism, so that all data operations at the various levels are deterministic.

III. Write Optimization and Update Tolerance

Location-optimized data systems should be write-optimized, and should support update-tolerance techniques. With the movements and continuous changes in the locations of objects over time, location data systems are update-heavy. Log-Structured Merge-trees (LSM-trees) [14] have been ubiquitously used for write-heavy workloads. LSM-ified R-tree indexes [11] have been proposed to support efficient querying along with write-optimized R-trees. However, the LSM R-tree [11] does not support efficient updates. LSM-based location indexes need to also be update-tolerant, e.g., along the lines of [17].

IV. Adaptive and Evolving Location Indexes and Data Stores

Location-optimized data systems should have adaptive and continuously evolving data indexes and location data stores that morph from being write-optimized at times into being read-optimized at other times, and vice versa. In addition to being write-heavy, the ubiquitous location-services and real-time analytical activities required of location-data systems mandate the need for location-data systems to handle read-heavy workloads as well. Thus, in addition to being write-optimized at times, location data systems should also be read-optimized to handle heavy analytical and location-services workloads, e.g., shortest path queries during rush hours.

Oscillating between being write-optimized at times to maximize high data-ingestion rates and being read-optimized at other times to handle location services and analytical workloads based on the spatial and temporal dimensions call for having adaptive location-based indexing techniques and location data stores that morph into being write-optimized at times and being read-optimized at other times.

LSM-like indexes being write-optimized cannot support efficiently read-heavy and analytics workloads. Adaptive and evolving indexes, and more generally, data stores that seamlessly morph between being read- or write-heavy are essential in location-optimized systems. Example access methods and systems along this direction include [2], [4], [13], [19], [20].

V. Native Multi-model Support

Typically, location data is associated with other data types, e.g., text and document data, road network and social network graph data, time series and trajectory data, image, video and audio data, etc. In addition to efficiently supporting location data, a native location-optimized data system should support and tightly integrate these multi-models to avoid impedance mismatch and performance hits when processing queries across multiple models.

Other Important Features. Native location-optimized data systems should support other important features including supporting high concurrency, awareness of new hardware and vectorization trends, privacy, among many other important features. The reader is referred to [3] for further elaboration.

References

[1] "Oracle big data spatial and graph." [Online]. Available: https://www.oracle.com/database/technologies/bigdata-spatialandgraph.html

[2] I. Alagiannis, S. Idreos, and A. Ailamaki, "H2o: a hands-free adaptive store," in *ACM SIGMOD '14*, 2014, p. 1103–1114.

[3] W. G. Aref, A. M. Aly, A. Daghistani, Y. Rayhan, J. Wang, and L. Zhou, "ILX: intelligent "location+x" data systems (vision paper)," *CoRR*, vol. abs/2206.09520, 2022.

[4] L. Arge, "The buffer tree: A technique for designing batched external data structures," *Algorithmica*, vol. 37, no. 1, pp. 1–24, 2003.

[5] R. Bayer and E. M. McCreight, "Organization and maintenance of large ordered indices," *Acta Informatica*, vol. 1, pp. 173–189, 1972.

[6] A. Davoudian, L. Chen, and M. Liu, "A survey on nosql stores," *ACM Computing Surveys*, vol. 51, no. 2, p. 1–43, 2019.

[7] A. Eldawy and M. F. Mokbel, "The era of big spatial data: A survey," *Foundations & Trends in Databases*, vol. 6, no. 3-4, pp. 163–273, 2016.

[8] N. Glombiewski, B. Seeger, and G. Graefe, "Waves of misery after index creation," *BTW 2019*, 2019.

[9] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1984, pp. 47–57.

[10] K. Karanasos and et al., "Extending relational query processing with ml inference," in *10th Annual Conference on Innovative Data Systems Research (CIDR '20)*, 2020.

[11] Y.-S. Kim, T. Kim, M. J. Carey, and C. Li, "A comparative study of log-structured merge-tree-based spatial indexes for big data," in *IEEE ICDE*, 2017, pp. 147–150.

[12] M. F. Mokbel and et al., "Towards mobility data science (vision paper)," *CoRR*, vol. abs/2307.05717, 2023.

[13] M. H. Moti, P. Simatis, and D. Papadias, "Waffle: A workload-aware and query-sensitive framework for disk-based spatial indexing," vol. 16, no. 4, 2022.

[14] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil, "The log-structured merge-tree (lsm-tree)," *Acta Informatica*, vol. 33, no. 4, pp. 351–385, 1996.

[15] I. Sabek and T. Kraska, "The case for learned in-memory joins," 2022.

[16] S. Shekhar, S. K. Feiner, and W. G. Aref, "Spatial computing," *Commun. ACM*, vol. 59, no. 1, pp. 72–81, 2016.

[17] J. Shin, J. Wang, and W. G. Aref, "The LSM rum-tree: A log structured merge r-tree for update-intensive spatial workloads," in *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 2285–2290.

[18] L. Xing, E. Lee, T. An, B. Chu, A. Mahmood, A. M. Aly, J. Wang, and W. G. Aref, "An experimental evaluation and investigation of waves of misery in r-trees," *Proc. VLDB Endow.*, vol. 15, no. 3, pp. 478–490, 2021.

[19] F. Zardbani, N. Mamoulis, S. Idreos, and P. Karras, "Adaptive indexing of objects with spatial extent," *Proc. VLDB Endow.*, vol. 16, no. 9, p. 2248–2260, may 2023.

[20] K. Zoumpatianos, S. Idreos, and T. Palpanas, "ADS: the adaptive data series index," *VLDB J.*, vol. 25, no. 6, pp. 843–866, 2016.