Query Processing – kNN

Jianzhong Qi and Rui Zhang

Synonyms

K nearest neighbor query; *K*NN query.

Definitions

Consider a set of n data objects $O = \{o_1, o_2, ..., o_n\}$ and a query object (user) q. Each object including the query object is associated with a d-dimensional vector representing its coordinate in a d-dimensional space ($d \in \mathbb{N}_+$). Given a query parameter k ($k \in \mathbb{N}_+$), the *k nearest neighbor* (kNN) query computes a size-k subset $S \subseteq O$ that contains the data objects that are the nearest to q:

$$\forall o_i \in S, o_j \in O \backslash S : dist(q, o_i) \leq dist(q, o_j)$$

Here, $dist(\cdot)$ is a function that returns the distance between two objects. A variety of distance functions have been considered in the literature, such as *Euclidean (L_2) distance*, *Manhattan (L_1) distance*, *Chebyshev (L_∞) distance*, spatial network distance, and *general metric distance*.

*K*NN query processing in the spatial data management context has focused on 2- or 3-dimensional Euclidean space and spatial network space.

Overview

The rapid growth of *location-based services* (LBS) has accumulated a massive amount of spatial data such as mapping data and user trajectories. Popular LBS such as Google Maps are serving millions of users who issue queries such as *finding Alice her nearest petrol stations*, which is a typical example of kNN queries. Most kNN algorithms use *spatial indices* over the data objects to help prune the unpromising search space and obtain high query efficiency. When the amount of data becomes too large to be indexed or queried on a single machine,

parallelism needs to be exploited to overcome such limitations. This raises new challenges of kNN queries in parallel index management and query processing.

Key Research Findings

We start with a brief review of two classic kNN algorithms and then focus on the discussion of recent developments on kNN algorithms that use parallelism to manage the growing amount of data.

Traditional KNN Algorithms

The two arguably most popular kNN query algorithms are the *best-first* (Hjaltason and Samet 1995) and the *depth-first* (Roussopoulos et al 1995) algorithms. These two algorithms rely on an tree index (e.g., R-trees (Guttman 1984)) over the data set O. They traverse the tree index starting from the tree root to find data objects nearest to q. During the traversal, the distance to the k^{th} NN among the data objects already visited is used to facilitate the pruning of unpromising tree branches that do not index nearer data objects. When a tree node N is visited, the best-first algorithm inserts all child nodes of N into a queue Q prioritized by an estimated distance between each child node and q. The next node to be visited is the one in Q with the smallest estimated distance to q. The depth-first algorithm simply visits the first child of each node before visiting the sibling nodes. While these two algorithms have shown high empirical query efficiency due to the pruning power of the tree indices, they are difficult to be parallelized because of the backtracking procedure in traversal.

Parallel KNN Algorithms

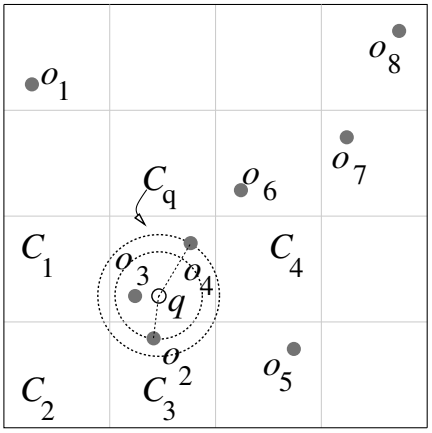

Fig. 1 Grid based kNN computation.

Parallel kNN query algorithms that suit popular parallel computation models such as *MapReduce* (Dean and Ghemawat 2008) have been developed to handle data beyond the processing capability of individual machines. Such algorithms partition the data space with a grid (a.k.a. *tiling*, cf. Fig. 1) so that data objects in different grid cells can be examined in parallel. A two-stage procedure is used in query processing. In the first stage, the cell C_q that contains the query object q is identified, and kNN is computed over the data objects in C_q (e.g., the 2NNs in Fig. 1 are o_3 and o_4). In the second stage, the cells adjacent to C_q are examined to find data objects that may be nearer to q than the k^{th} NN already found (e.g, cells C_1, C_2, C_3, and C_4, which are within the circle defined

by the second NN o_4; o_2 is found to be the new second NN). This step may be repeated over more cells adjacent to the cells examined, until the nearest data object in these adjacent cells is farther away than the k^{th} NN already found.

Hadoop Based *k*NN algorithms

Hadoop is an open-source implementation of the MapReduce computation model. The MapReduce model processes a task (a MapReduce job) in two phases: *Map* and *Reduce*. The Map phase assigns data to different machines in a cluster, while the Reduce phase runs computation on each machine. The data storage component of Hadoop named *Hadoop Distributed File System* (HDFS) is an implementation of the *Google File System* (Ghemawat et al 2003). HDFS is optimized for large data blocks (e.g., 64 MB per block). Hadoop based *k*NN algorithms need to design indices following this block size.

Aji et al (2012) propose a *k*NN algorithm based on MapReduce with an assumption that the number of data objects is relative small and the entire data set can be replicated to each machine for query processing. They focus on efficient processing of a large number of *k*NN queries at the same time, which is done by a single MapReduce job. In the Map phase, the query objects are partitioned with a grid and assigned to different machines according to their grid cell id. In the Reduce phase, the data set is replicated to each machine, upon which a local index (e.g., an R-tree) is built. The queries are then processed on each machine independently.

SpatialHadoop (Eldawy and Mokbel 2013) is a MapReduce framework for spatial query processing. It employs a two-level index structure that consists of a *global index* and a *local index*. The global index is built by partitioning the data space such that every partition can fit in a block in HDFS. Grid based and R-tree based indices are implemented in SpatialHadoop. A grid based index simply partitions the data space with a uniform grid. An R-tree based index creates an R-tree on a sample of the data set, where the leaf nodes are used to partition the data space. *Minimum bounding rectangles* (MBR) of the partitions are stored in the global index which is kept in the main memory of the master node of the cluster. A local index is built on every partition and kept in HDFS. *K*NN queries are processed with the two-stage search space expansion procedure as described earlier. First, the partition containing the query object is identified using the global index, where a *k*NN query is computed using the local index. Then, adjacent partitions are explored until no more NNs can be found.

AQWA (Aly et al 2015) builds a global index using the k-d tree (Bentley 1975) to create the data partitions where every partition is stored in HDFS as a block. The k-d tree is stored in the main memory of the master node. Additionally, the master node keeps a count of data objects for each partition. This enables processing a *k*NN query with only one MapReduce job as follows. The partitions nearest to the query object q that together contain at least k objects can be identified from the k-d tree index and the data counts. The farthest point of these partitions from q defines a circle centered at q. Only partitions within this circle needs to be fetched for query processing, which can be done by a single MapReduce job.

HBase Based *k*NN algorithms

HBase is an open-source *key-value* store (*NoSQL* database) built on top of HDFS. It hides the underlying block storage from users and hence *k*NN algorithms based on it do not have to consider how to organize the data in blocks.

MD-HBase (Nishimura et al 2011) is a multidimensional data store built on HBase. This system stores the *Z-order* (Orenstein and Merrett 1984) value of a data object together with the data object itself as a key-value pair in HBase. The key-value pairs are partitioned by the key range, and every partition is stored as a *region*. Data in HBase is indexed with a B^+-tree like index structure, where the key range of a region is stored in an index layer called the *META* layer. MD-HBase uses the longest common prefix of the keys (in the binary form) in a region as the index key of the region. *K*NN queries are processed following a search space expansion procedure. At start, the region with an index key that shares the longest common prefix with the Z-order value of the query object q is identified, and an initial *k*NN answer is computed from the region. The k^{th} NN found defines a distance from q where the search should expand to. This procedure repeats until no new *k*NN objects can be found in the expanded search space.

The *KR^+ index* (Hsu et al 2012) follows a similar *k*NN query procedure to that of MD-HBase, but uses an *R^+-tree* (Sellis et al 1987) to group the data objects and Hilbert-order (Lawder and King 2001) values of the MBRs of the tree leaf nodes as index keys.

HGrid (Han and Stroulia 2013) uses another index key formulation, which is based on a two-level grid structure. The first level uses a coarse grid where the grid cells are indexed by the Z-order values. The second level further partitions each first-level grid cell with a regular grid where the cells are indexed by their row and cell numbers. The key value of a data object is formed by a concatenation of the index keys of the first and second level cells that this object locates at. HGrid has a slightly different *k*NN algorithm. The algorithm starts by estimating a region that may contain sufficient data objects to form a *k*NN answer based on the data density. Then, an initial *k*NN answer is computed from this region, and the search space expansion strategy is used to refine the answer.

COWI and CONI (Cahsai et al 2017) build a quad-tree (Finkel and Bentley 1974) over the data set where the leaf node numbers are used as the index keys of the data objects. COWI stores the non-leaf levels of the quad-tree index in the memory of a master machine. The index also contains the number of data objects in each leaf node, which is used to help query processing in a way similar to that of AQWA. CONI considers the case where the quad-tree is too large to be held in memory. It stores the tree as a separate table in HBase. At query time, the tree index is accessed first to determine the leaf nodes that may be relevant for query processing.

Spark Based *k*NN algorithms

Spark provides an in-memory computation cluster framework to handle massive data. A set of data to be processed is modeled as a *resilient distributed dataset* (RDD). RDDs are stored in partitions across machines in the cluster.

GeoSpark (Yu et al 2015) encapsulates RDDs with an abstraction named the *spatial resilient distributed dataset* (SRDD) to store spatial data. An SRDD is partitioned using a uniform grid. Local indices such as quad-trees may be created for data objects in the partitions. *K*NN query processing on GeoSpark is done by following the grid based search space expansion strategy as discussed earlier.

Simba (Xie et al 2016) uses both global and local indices. It creates a new RDD abstraction called the *IndexRDD* to store a local index (e.g., an R-tree) together with the data objects in each partition, while the partitions are created by the *STR* algorithm (Leutenegger et al 1997). The global index (e.g., an R-tree) is created over the partitions and held in-memory in the master node for fast query processing. *K*NN queries on Simba also follows the search space expansion strategy, but varies from the standard procedure slightly by fetching multiple partitions at start instead of only the partition containing the query object. The k^{th} NN from the fetched partitions is used to define the search space to be explored next.

Examples of Application

*K*NN queries have a large variety of applications in different fields such as spatio-temporal databases, location-based services, and data mining, just to name but a few. In spatial data management, a typical application is in digital mapping, e.g., Google Maps, where *k*NN queries may be used for finding the nearest *Points-of-Interest* (POI) for query users. A closely related application is in ride-sharing, e.g., Uber, where *k*NN queries may be used to find the nearest cars for car riders. Another application is in augmented reality gaming, e.g., Pok*é*mon GO, where *k*NN queries may be used to find the nearest gaming objects of game players.

Future Directions for Research

The techniques discussed above consider stationary data objects whose locations do not change over time. With the growing popularity of smart mobile devices, location-based services for moving data objects (users) become prevalent, which call for efficient *k*NN algorithms over massive sets of moving objects (Nutanong et al 2008; Wang et al 2014; Li et al 2014, 2016; Gu et al 2016) and streaming location data (Koudas et al 2004). Simply recomputing the query whenever there is an object location update may be infeasible due to the larger number of objects and updates. Keeping the query result updated under such constraints is a challenging task.

*K*NN queries in higher dimensional spaces (Jagadish et al 2005) is another challenge. As a major application domain, data mining requires *k*NN computation over feature spaces that may have thousands of dimensions. The techniques discussed are inapplicable, because they use spatial indices for low dimensional spaces. Approximate *k*NN algorithms are in need to cope with the high processing costs in high dimensional spaces.

References

Aji A, Wang F, Saltz JH (2012) Towards building a high performance spatial query system for large scale medical imaging data. In: Proceedings of the 20th International Conference on Advances in Geographic Information Systems (SIGSPATIAL), pp 309–318

Aly AM, Mahmood AR, Hassan MS, Aref WG, Ouzzani M, Elmeleegy H, Qadah T (2015) Aqwa: Adaptive query workload aware partitioning of big spatial data. Proceedings of the VLDB Endowment 8(13):2062–2073

Bentley JL (1975) Multidimensional binary search trees used for associative searching. Communications of the ACM 18(9):509–517

Cahsai A, Ntarmos N, Anagnostopoulos C, Triantafillou P (2017) Scaling k-nearest neighbours queries (the right way). In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp 1419–1430

Dean J, Ghemawat S (2008) Mapreduce: Simplified data processing on large clusters. Communications of the ACM 51(1):107–113

Eldawy A, Mokbel MF (2013) A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. Proceedings of the VLDB Endowment 6(12):1230–1233

Finkel RA, Bentley JL (1974) Quad trees a data structure for retrieval on composite keys. Acta Informatica 4(1):1–9

Ghemawat S, Gobioff H, Leung ST (2003) The google file system. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP), pp 29–43

Gu Y, Liu G, Qi J, Xu H, Yu G, Zhang R (2016) The moving K diversified nearest neighbor query. IEEE Transactions on Knowledge and Data Engineering 28(10):2778–2792

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

Han D, Stroulia E (2013) Hgrid: A data model for large geospatial data sets in hbase. In: 2013 IEEE 6th International Conference on Cloud Computing (CLOUD), pp 910–917

Hjaltason GR, Samet H (1995) Ranking in spatial databases. In: Proceedings of the 4th International Symposium on Advances in Spatial Databases (SSD), pp 83–95

Hsu YT, Pan YC, Wei LY, Peng WC, Lee WC (2012) Key formulation schemes for spatial index in cloud data managements. In: 2012 IEEE 13th International Conference on Mobile Data Management (MDM), pp 21–26

Jagadish HV, Ooi BC, Tan KL, Yu C, Zhang R (2005) idistance: An adaptive b+-tree based indexing method for nearest neighbor search. ACM Transactions on Database Systems 30(2):364–397

Koudas N, Ooi BC, Tan KL, Zhang R (2004) Approximate nn queries on streams with guaranteed error/performance bounds. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB) - Volume 30, pp 804–815

Lawder JK, King PJH (2001) Querying multidimensional data indexed using the hilbert space-filling curve. SIGMOD Record 30(1):19–24

Leutenegger ST, Lopez MA, Edgington J (1997) Str: a simple and efficient algorithm for r-tree packing. In: Proceedings 13th International Conference on Data Engineering (ICDE), pp 497–506

Li C, Gu Y, Qi J, Yu G, Zhang R, Yi W (2014) Processing moving knn queries using influential neighbor sets. Proceedings of the VLDB Endowment 8(2):113–124

Li C, Gu Y, Qi J, Yu G, Zhang R, Deng Q (2016) INSQ: an influential neighbor set based moving knn query processing system. In: Proceedings of the 32nd IEEE International Conference on Data Engineering (ICDE), pp 1338–1341

Nishimura S, Das S, Agrawal D, Abbadi AE (2011) Md-hbase: A scalable multi-dimensional data infrastructure for location aware services. In: Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management (MDM) - Volume 01, pp 7–16

Nutanong S, Zhang R, Tanin E, Kulik L (2008) The v*-diagram: A query-dependent approach to moving knn queries. Proceedings of the VLDB Endowment 1(1):1095–1106

Orenstein JA, Merrett TH (1984) A class of data structures for associative searching. In: Proceedings of the 3rd ACM SIGACT-

SIGMOD Symposium on Principles of Database Systems (PODS), pp 181–190

Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD), pp 71–79

Sellis TK, Roussopoulos N, Faloutsos C (1987) The r+-tree: A dynamic index for multi-dimensional objects. In: Proceedings of the 13th International Conference on Very Large Data Bases (VLDB), pp 507–518

Wang Y, Zhang R, Xu C, Qi J, Gu Y, Yu G (2014) Continuous visible k nearest neighbor query on moving objects. Information Systems 44:1–21

Xie D, Li F, Yao B, Li G, Zhou L, Guo M (2016) Simba: Efficient in-memory spatial analytics. In: Proceedings of the 2016 SIGMOD International Conference on Management of Data (SIGMOD), pp 1071–1085

Yu J, Wu J, Sarwat M (2015) Geospark: a cluster computing framework for processing large-scale spatial data. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL), pp 70:1–70:4

Cross-References

- Indexing
- Query Processing – Comp Geometry
- Query Processing – Joins