Distributed In-Memory Processing of All k Nearest Neighbor Queries (Extended Abstract)

Georgios Chatzimilioudis*, Constantinos Costa*, Demetrios Zeinalipour-Yazti*,
Wang-Chien Lee† and Evaggelia Pitoura‡

* University of Cyprus, 1678 Nicosia, Cyprus
† Pennsylvania State University, PA 16802, USA
‡ University of Ioannina, 45110 Ioannina, Greece

***Abstract*—A wide spectrum of Internet-scale mobile applications, ranging from social networking, gaming and entertainment to emergency response and crisis management, all require efficient and scalable All k Nearest Neighbor (AkNN) computations over millions of moving objects every few seconds to be operational. In this paper we present Spitfire, a distributed algorithm that provides a scalable and high-performance AkNN processing framework to our award-winning geo-social network named Rayzit. The proposed algorithm deploys a fast load-balanced partitioning along with an efficient replication-set selection, to provide fast main-memory computations of the exact AkNN results in a batch-oriented manner. We evaluate, both analytically and experimentally, how the pruning efficiency of the Spitfire algorithm plays a pivotal role in reducing communication and response time up to an order of magnitude, compared to three state-of-the-art distributed AkNN algorithms executed in distributed main-memory.**

I. Introduction

In the age of smart urban and mobile environments, the mobile crowd generates and consumes massive amounts of heterogeneous data. Such streaming data may offer a wide spectrum of enhanced science and services, ranging from mobile gaming and entertainment, social networking, to emergency and crisis management services [3]. One useful query for the aforementioned services is the *All kNN (AkNN)* query: *finding the k nearest neighbors for all moving objects*.

Formally, the kNN of an object o from some dataset O, denoted as $kNN(o, O)$, are the k objects that have the most similar attributes to o. Specifically, given objects $o_a \neq o_b \neq o_c$, $\forall o_b \in kNN(o_a, O)$ and $\forall o_c \in O - kNN(o_a, O)$ it always holds that $dist(o_a, o_b) \leq dist(o_a, o_c)$[1]. An *All kNN (AkNN)* query generates a kNN graph. It computes the $kNN(o, O)$ result for every $o \in O$ and has a quadratic worst-case bound. An AkNN query can alternatively be viewed as a *kNN Self-Join*: *Given a dataset O and an integer k, the kNN Self-Join of O combines each object $o_a \in O$ with its k nearest neighbors from O, i.e., $O \bowtie_{kNN} O = \{(o_a, o_b) | o_a, o_b \in O$ and $o_b \in kNN(o_a, O)\}$.*

A real-world application based on such a query is *Rayzit* [3][2], our award-winning crowd messaging architecture, that connects users instantly to their *k Nearest Neighbors (kNN)* as they move in space (Figure 1, left). The dynamic AkNN network in Rayzit allows the propagation of microblog messages to the closest neighbors of a user. Similar to other social network applications (e.g., Twitter, Facebook), scalability

[1]In our discussion, *dist* can be any L_p-norm distance metric, such as Manhattan (L_1), Euclidean (L_2) or Chebyshev (L_∞).

[2]Rayzit: http://rayzit.com/

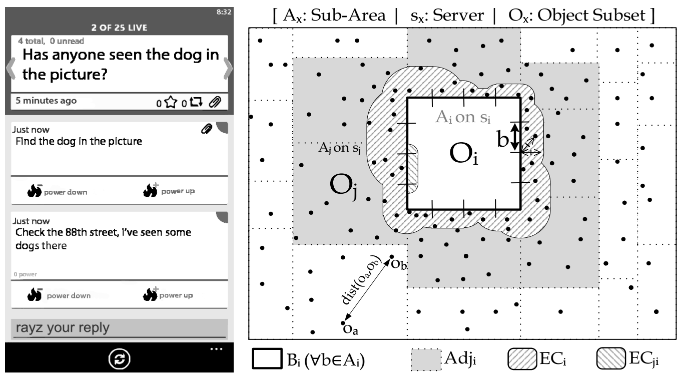
2 OF 25 LIVE
4 total, 0 unread
Has anyone seen the dog in the picture?
5 minutes ago
Just now
Find the dog in the picture
Just now
Check the 88th street, I've seen some dogs there
rayz your reply

Fig. 1. (Left) Our Rayzit crowd messenger enabling users to interact with their k geographic Nearest Neighbors. (Right) Distributed main-memory AkNN computation in Rayzit is enabled through the *Spitfire* algorithm, which is a batch-oriented algorithm working in three steps.

is key in making *Rayzit* functional and operational. Therefore we are challenged with the necessity to perform a fast computation of an AkNN query every few seconds in a scalable architecture. The wide availability of off-the-shelf, shared-nothing, cloud infrastructures brings a natural framework to cope with scalability, fault-tolerance and performance issues faced in processing AkNN queries. Only recently researchers have proposed algorithms for optimizing AkNN queries in such infrastructures.

The state-of-the-art AkNN algorithm for the cloud environment [4] consists of three phases, namely *partitioning* the geographic area into sub-areas, computing the kNN *candidates* for each sub-area that need to be *replicated* among servers in order to guarantee correctness and finally, computing locally the global AkNN for the objects within each sub-area taking the *candidates* into consideration. The given algorithm has been designed with an offline (i.e., analytic-oriented) AkNN processing scenario in mind, as opposed to an online (i.e., operational-oriented) AkNN processing scenario we aim for in this work. The *performance* of [4] can be greatly improved, by introducing an optimized partitioning and replication strategy. These improvements, theoretically and experimentally shown to be superior, are critical in dramatically reducing the AkNN query processing cost yielding results within in a few seconds, as opposed to minutes, for million-scale object scenarios.

II. The Spitfire Algorithm

In [1], we devise a high-performance distributed main-memory algorithm named *Spitfire*. The name *Spitfire* is derived by a syllable play using the meaning of these three steps, namely **Split**, **Re**fine **re**plicate, which implies good mechanical performance. Below we outline its three internal steps and its intrinsic characteristics.

Step 1 (Partitioning): Initially, O is partitioned into (disjoint) sub-areas with an approximately equal number of objects, i.e., $O = \bigcup_{1 \leq i \leq m} O_i$. We use a simple but fast centralized hash-based adaptation of equi-depth histograms to achieve good load balancing in $O(n+\sqrt{nm})$ time, where n denotes the number of objects. It first hashes the objects based on their locations into a number of sorted equi-width buckets on each axis and then partitions each axis sequentially by grouping these buckets in an equi-depth fashion. Figure 1 (right) shows one such partitition, where each O_i belongs to area A_i handled by server s_i, B_i denotes the border segments surrounding A_i (i.e., right-dashed area outside O_i), and Adj_i the adjacent servers to s_i (i.e., gray area). Its important to notice that the partitioning step guarantees that each s_i will have at least k objects.

Step 2 (Replication): Subsequently, each s_i computes a subset of O_i, coined *External Candidates* EC_{ji}, which is possibly needed by its neighboring s_j for carrying out a local AkNN computation in Step 3 (refinement). The given set EC_{ji} is transmitted by s_i to s_j (i.e., left-dashed area within O_i, as depicted again in Figure 1 (right)). Since each s_j applies the above operation as well, we also have the notion of EC_{ij}. The union of EC_{ij} for all neighboring $s_j \in Adj_i$ defines the External Candidates of O_i, i.e., $EC_i = \bigcup_{1 \leq j \leq Adj_i} EC_{ij}$. The cardinality of all EC_i defines the *Spitfire replication factor*.

Step 3 (Refinement): Finally, each s_i performs a local $O_i \ltimes_{kNN} (O_i \cup EC_i)$ computation, which is optimized by using a heap structure along with internal geographic grouping and bulk processing.

Spitfire's main advantages to prior work follow are as follows: i) it is suitable for *online operational* AkNN workloads as opposed to *offline analytic* AkNN workloads. Particularly, it is able to compute the AkNN result-set every few seconds as opposed to minutes required by state-of-the-art AkNN algorithms configured in main-memory; ii) it uses a pruning strategy that achieves replication factor $f_{Spitfire}$, which is shown analytically and experimentally to be always better than that achieved by state-of-the-art AkNN algorithms; and iii) it is a single round algorithm as opposed to state-of-the-art AkNN algorithms that require multiple rounds.

III. Experimental Evaluation

To validate our proposed ideas and evaluate *Spitfire*, we conduct a comprehensive set of experiments over our VCenter IaaS cloud. For our experiments, we use 9 Ubuntu 12.04 server images (i.e., denoted earlier as s_i), each featuring 8GB of RAM with 2 virtual CPUs (@ 2.40GHz).

Methodology: We use four datasets scaling up to 1M objects: *Random (synthetic), Oldenburg (realistic), Geolife (realistic)* and *Rayzit (real)*. We compare one centralized (Proximity [2]) and four Hadoop-oriented algorithms, namely H-BNLJ [5], H-BRJ [5] and PGBJ [4]. Given that Hadoop transfers intermediate results between tasks through a *disk-oriented* Hadoop Distributed File System (HDFS), we deploy UC Berkeley's Tachyon in-memory file system to enable *memory-oriented* data-sharing across MapReduce jobs. Our metrics are i) *Response Time*, which represents the actual time required by

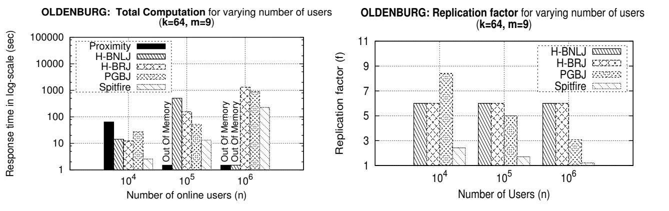

Fig. 2. (Left) AkNN query response time with increasing number of users. (Right) Replication factor f with increasing number of users. The optimal value for f is 1, signifying no replication between workers.

a distributed AkNN algorithm to compute its result; and ii) *Replication Factor* (f), which represents the number of times the n objects are replicated between servers to guarantee correctness of the AkNN computation. Below we present an indicative result for the above algorithms with an emphasis on the Oldenburg dataset.

Results: In Figure 2 (left), we increase the workload of the system by growing the number of online users (n) exponentially and measure the response time and replication factor of the algorithms under evaluation. We can clearly see that *Spitfire* outperforms all other algorithms in every case. It is also evident that H-BNLJ and H-BRJ do not scale. H-BRJ achieves the worst time for 10^6 users. Our analysis shows that H-BRJ's response time is spent on communication, which is indicated by its communication complexity of $O(\sqrt{m}n)$.

For 10^4 online users, *Spitfire* outperforms all algorithms by at least 85% for all dataset, whereas for 10^5 *Spitfire* outperforms PGBJ, by 75%, 75% and 53% for the Random, Oldenburg and Geolife datasets, respectively. *Spitfire* and PGBJ are the only algorithms that scale. For a million online users (n=10^6), *Spitfire* and PGBJ are the fastest algorithms, but *Spitfire* still outperforms PGBJ by 67%, 75%, 14% for the Random, Oldenburg and Geolife datasets, respectively.

Figure 2 (right) shows the replication factor for the distributed algorithms. It is noteworthy that the replication factor $f_{Spitfire}$ of *Spitfire* is always close to the optimal value 1. *Spitfire* only selects a very small candidate set around the border of each server and that provides it with a better performance than PGBJ.

References

[1] G. Chatzimilioudis, C. Costa, D. Zeinalipour-Yazti, W.-C. Lee, and E. Pitoura. Distributed in-memory processing of all k nearest neighbor queries. *Knowledge and Data Engineering, IEEE Transactions on*, x(x):x–x, 2016.

[2] G. Chatzimilioudis, D. Zeinalipour-Yazti, W.-C. Lee, and M. Dikaiakos. Continuous all k-nearest-neighbor querying in smartphone networks. In *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on*, pages 79–88, July 2012.

[3] C. Costa, C. Anastasiou, G. Chatzimilioudis, and D. Zeinalipour-Yazti. Rayzit: An anonymous and dynamic crowd messaging architecture. In *Mobile Data Management (MDM), 2015 16th IEEE International Conference on*, volume 2, pages 98–103, June 2015.

[4] W. Lu, Y. Shen, S. Chen, and B. C. Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *Proc. VLDB Endow.*, 5(10):1016–1027, June 2012.

[5] C. Zhang, F. Li, and J. Jestes. Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 38–49, New York, NY, USA, 2012. ACM.