Real-time Spatial Interpolation of Continuous Phenomena using Mobile Sensor Data Streams

Silvia Nittel
School of Computing and Information Science
University of Maine, USA
nittel@spatial.maine.edu

J.C. Whittier
School of Computing and Information Science
University of Maine, USA
john.c.whittier@maine.edu

Qinghan Liang
School of Computing and Information Science
University of Maine, USA
qliang@spatial.maine.edu

ABSTRACT
Technology advances have created a wide variety of novel, inexpensive sensors in the millimeter range that can be attached to or embedded into smartphones. These sensors are now directly connected to the Internet enabling us to collect high frequency updates from potentially thousands of mobile sensors densely deployed over an urban area. Today, data stream management systems (DSMS) are powerful data processing tools for update rates of 100,000-500,0000 tuples/s. In this paper, we investigate extending DSMS for monitoring continuous environmental phenomena such as air borne toxins or air quality based on up to 250K individual mobile sensor updates per query window to be spatially interpolated into a smooth, grid-based representation in near real-time. We propose a stream query operator approach and investigate different strategies to achieve near real-time spatial interpolation, while investigating memory footprint, runtime efficiency and interpolation quality of the different strategies.

Categories and Subject Descriptors
H.2.3 [**Database Management, Systems**]: Query Processing H2.8 [**Database Management**, **Database Applications**] Spatial Databases.

General Terms
Algorithms, Performance.

Keywords
Real-time spatial interpolation, DSMS, sensor data streams, continuous phenomenon.

1. INTRODUCTION
Technology advances have created a large variety of novel, inexpensive, small-form sensors that can be attached to or embedded into smartphones. We can expect that next generation smartphones will be sold with a plethora of built-in environmental and health sensors. Via smartphones the sensors are directly connected to the Internet and enable us to collect high frequency updates from potentially hundreds of thousands of mobile sensors deployed over a larger urban area [1,2,3] enabling real-time hazard detection or urban air quality control [3]. Processing and analyzing such large sets of sensor data streams in real-time challenges traditional processing strategies using DBMS and GIS due to the limitations of the DBMS disk-based architecture. Over the last decade, data stream management systems (DSMS) have been developed motivated by the throughput needs of real-time financial analysis and fraud analysis applications. DSMS store incoming updates only in main memory and process them directly using non-blocking, pipelined stream query operators.

In this paper, we focus on extending DSMS to handle the monitoring of *continuous environmental phenomena* with mobile sensor data streams in near real-time. Imagine the following scenario: in a metropolitan area such as Los Angeles (ca. 34,000 mi^2) about 2 million people continuously collect environmental data such as air quality data or volatile organic compounds (VOC) using their smartphones with a rate of 1 update per 30s; sensor data streams are either sent to the cloud or distributed servers, which relay the sensor data to a centralized DSMS. About 67,000 updates per second arrive at the DSMS (for simplicity, we ignore network latency issues and data skew in this paper). A user query requests the following information: "*Continuously monitor the VOC spatial field using the sensor streams s where s.type=VOC and contains(Los Angeles, s.location) using window=15s*".

A user is more interested in the information of the smooth distribution of the VOC field over the geographic region rather than the individual sensor data stream updates, their location and the values (similar to a scatterplot). Thus, the DSMS needs to spatially interpolate all individual sensor data streams to produce this smooth representation and provide a grid output. Spatial interpolation algorithms are widely established [8], but historically they are tuned towards sparse samples over large geographic regions and achieving high interpolation quality, making them computationally complex and expensive [8]. *To the best of our knowledge, real-time spatial interpolation based on sensor data streams using DSMS has not yet been investigated today*.

With up to 1 million sensor update tuples per query window, efficient spatial interpolation that generates a near real-time grid based approximation is computationally challenging and the detailed research focus of this paper. Historically, a wide variety of spatial interpolation approaches exists (e.g. Voronoi, Kriging, Inverse Distance Weighting (IDW), and others) [8]. For our approach to achieve near real-time execution time in a data stream setting, re-designing spatial interpolation methods as non-blocking data streams operators is of key importance. We have to consider the following: *first*, main memory is limited in DSMS as all necessary incoming data and query operators share the available memory. *Secondly*, the computational cost of a spatial interpolation method has to be reduced and we need to provide scalable and adaptive performance for very large numbers of sample points. *Third*, despite these resource restrictions, a redesigned interpolation method still needs to generate a high quality interpolation result.

Our contributions: A main challenge of spatially interpolating sensor update streams is the identification of relevant updates for

ACM SIGSPATIAL GIS '12, November 6-9, 2012. Redondo Beach, CA,USA Copyright (c) 2012 ACM ISBN 978-1-4503-1691-0/12/11...$15.00.

the entire estimated region within a stream query, made especially challenging by *moving* sensors. We propose a novel in-memory grid index to keep track of incoming updates per query window, and various strategies of organizing the tuples per grid cell (*unordered materialized list, unordered virtual list, and incremental kNN list*). Our extensive tests show that using up to 250k updates per query window can be interpolated in 1-2s using our proposed methods.

2. PIPELINED OPERATORS FOR SENSOR STREAM SPATIAL INTERPOLATION

In DSMS, a declarative stream query is translated into a query execution plan that consists of a directed acyclic stream operator graph. Each operator contains an *input queue*, the *operator instance* with its *state* (e.g. necessary cached tuples to perform the operation), and is connected to an *output queue* to which the result tuples are pushed. Operators work in a pipelined, non-blocking fashion.

Applying this operator principle to a single data stream query that performs spatial interpolation, the query's first operator needs to keep track of all the relevant incoming sensor tuples within the query window, and assign tuples to the query's other operators (see Figure 1). Once all tuples are available, they are interpolated *temporally* (not discussed in this paper), and *spatially*. The query output is a grid and each grid cell value is interpolated based on the available updates within the cell and surrounding cells. For this paper, we chose Inverse Distance Weighting (IDW) as spatial interpolation method due to its linear computational complexity [6].

2.1 Pipelined stream query operators for real-time spatial interpolation

Our proposed pipelined stream operator query plan is depicted in Figure 1. It is composed of the following stream operators:

Scan/index operator: the scan operator observes the queue of incoming sensor tuples. For each tuple, it checks whether the tuple's location stamp is within the query's spatial region. It then assigns it to one or more cells for which it can be used as a potential kNN value (discussed in Section 3.3).

Grid cell operator: conceptually, a cell operator exists for each grid cell, and cell operators can potentially run in parallel. The cell (or a block of cells) operator maintains a list of relevant tuples per cell, and once all tuples for a query window are available, the cell's tuples are pushed to the input queue of the cell interpolation operator.

Cell interpolation operator: the cell interpolation operator interpolates all tuples per grid cell and estimates the value for the grid cell.

Assemble operator: The assemble operator keeps track of all grid cells for a single grid and generates the output grid.

The individual grid cells, grid cellblocks or the entire grid can be used as input for other stream query operators, e.g. event extraction operators.

2.2 Optimizing sensor data stream based spatial interpolation

While the stream operators for spatial interpolation can be optimized in different generic DSMS ways (stream partitioning, or parallel execution of operator instances), two potential bottlenecks exist: computational complexity and main

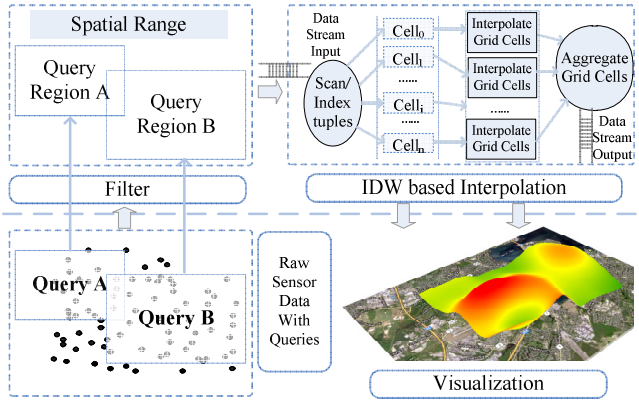

Figure 1. Sensor data stream based spatial interpolation.

memory consumption. Both parameters are influenced by the spatial interpolation configuration as well as the input sample size. If we parameterize the interpolation to generate a 256x256 output grid and assume about 100K sensor updates, about 1.5 tuples per grid cell are available (with uniform spatial sensor distribution). To improve interpolation quality, we start adding the tuples of surrounding cells to a cell's interpolation input to improve quality. While this improves interpolation quality, this also significantly increases the memory footprint and the computation time. Similarly, interpolating finer grids leads to higher computation cost and a larger memory footprint. In the following sections we discuss our proposed options to optimize these parameters.

3. OPTIMIZING INDIVIDUAL GRID CELL INTERPOLATION

As mentioned, we create a grid cell-based index structure between the scan operator and the interpolation stream operators. The grid-based index has the same resolution as the spatial interpolation output grid. In this section, we discuss options of organizing the tuples per grid cell.

3.1 Materialized unordered grid cell list

First, the tuples per grid cell are assigned to a *materialized unordered tuple list per cell (MUCL)*. Tuples are inserted in the order they arrive. Instead of referencing the original tuples, a cell's materialized list consists of <*distance*, *value*> pairs. The *distance* parameter is the spatial distance between the cell c_i's center and the location stamp of the sensor tuple. Additionally, the tuples of neighboring cells are added to c_i's MUCL. If we vary the radius between 0 and 10 around c_i, c_i's tuple list can grow up to 1200 elements for r=10 and a 256x256 grid, and around 260K updates. The tuple list is passed to the cell interpolation operator.

In this configuration, the memory footprint can be excessive since we a) create new derived tuples from the original tuples, and b) derived tuples are replicated within many neighboring cells if a large radius is chosen. On the other hand, this approach redistributes tuples of under-sampled cells, which works well for spatial data skew.

3.2 Virtual unordered grid cell list

In the second approach, we avoid the replication of tuples in neighboring cells and assign each tuple only to the grid cell in which it is contained (*VUCL, virtual unordered tuple list per cell*). Thus, tuples are assigned to at most one grid cell. The cell list consists of *references* to the original tuples, which significantly minimizes the memory footprint and the runtime for the indexing step. An additional *gather operator* creates an individual cell list

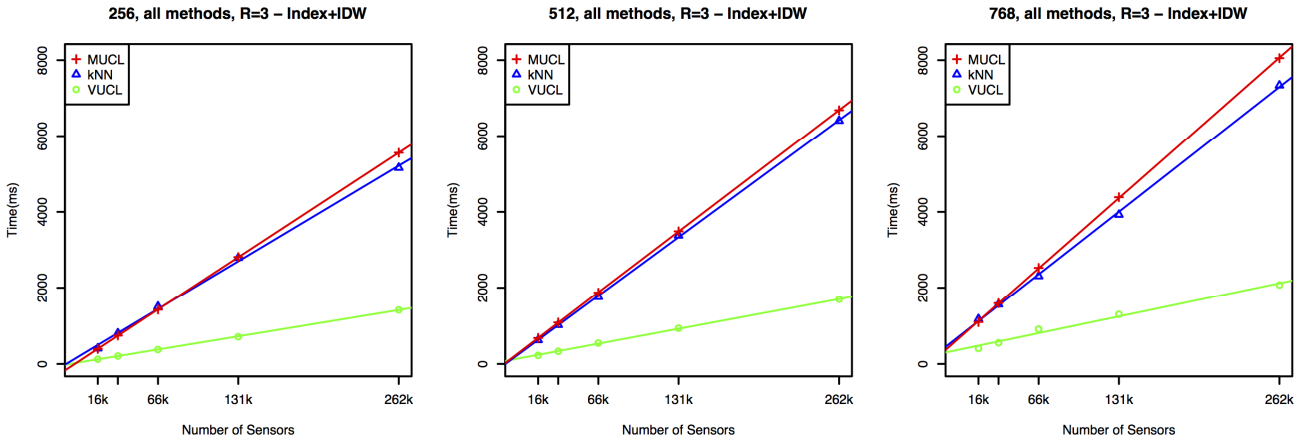

Figure 2: Overall performance results

for c_i by iterating over the neighboring cells of c_i, and aggregating the partial lists of neighboring cells into a new temporary list, which now consists of (*distance*, *value*) pairs. The temporary list is used directly for the interpolation operator and then discarded. In this configuration, the memory footprint is small, but the interpolation operator requires more computation time.

3.3 Materialized kNN grid cell list

In the third option, a *fixed length kNN list* is created per grid cell (*k nearest neighbor cell list, kNNCL*). Compared to the unordered and variable sized lists of 3.1 and 3.2 the structure and content of the kNN list has direct impact on the cell's interpolation quality; since k is static the maximum memory footprint of kNNCL can be estimated. Here, the scan operator reads the input tuples and each tuple t_i's location value. It determines whether t_i is contained in cell c_i and calculates the distance between c_i's center and t_j's (x, y) value. The scan operator tests c_i's existing kNN list and compares the distance of t_i with the distances of the existing tuples t_j in the list. If the list is not full or t_i is located closer to the cell center than the furthest of the other existing tuples t_j t_i is added to the list. Otherwise, t_i is discarded. If the list is over capacity, the furthest tuple t_j is discarded.

The scan operator also potentially assigns t_i to neighboring cells of c_i. Thus, t_i is additionally tested with regard to relevance to the surrounding cells of cell c_i with a radius r around c_i. If we chose k=10 per c_i and generate a 768x768 grid (ca. 600K cells) in this case even with a very large update set per window (e.g. 130K), on average only 0.2 tuples per cell are present. Additionally testing tuples against the neighboring cells with a radius r=5 assures that sufficient kNN tuples for each cell c_i are found.

4. PERFORMANCE EVALUATION

4.1 Experimental Setup

Since real data sets for our research are not yet available, we used synthetic data modeled in a NetLogo environment. Agents move freely according to a random walk model and generate environmental sensor measurements along their paths. We created a reproducible phenomenon that is smoothly distributed over the simulated area and changes its gradient slowly and smoothly over the entire area continuously. We generated the following agent populations: 16,384, 32,768, 65,536, 131,072, 262,144 and 524,288. Each agent sampled the phenomenon at the same time and updated a single sample per query window for the tests in this paper. We tested the grid sizes 256x256, 512x512, and 768x768.

Runtime Environment: The proposed strategies were implemented in Java in a limited DSMS environment, i.e. operators are connected via queues, and work in a pipelined fashion, but we do not consider any of the other DSMS components. The experiments were run on an Apple MacBook Pro with a 2.66 GHz Intel Core i7 (Model i7-620M; a dual core processor with four virtual cores), 8 GB DDR3 memory at 1067 MHz running Mac OS X 10.7.4 (64 bit) and Java 1.6.0_31 (64 bit). The discussed strategies were implemented as data stream components assuming a DSMS environment.

4.2 Indexing strategies for grid cell

We investigated the runtime performance for **MUCL** (materialized unordered cell list), **VUCL** (virtual unordered cell list) and **kNNCL** (k nearest neighbor cell list). Due to space limitations in this paper, we present only a small subset of all results graphically (Figure 2). Besides *overall runtime performance*, we investigated the *impact of the neighborhood radius* selected with regard to adding tuples to a cell's list and ran tests with radii = {0,1,2,3,5,10}.

4.2.1 MUCL

As expected, enlarging the radius using the MUCL strategy increases runtime for the indexing step significantly. For example, for a 512x512 grid with 262K samples, indexing for radii 0-2 takes about 3s, while r=5 requires about 30s. We determined that r=5 can be considered the upper search bound for the following experiments. In the MUCL case, the interpolation step itself is computationally inexpensive; for a 512x512 grid and radii 0-5 the runtime is around 1s using a degree of 4 in parallelization.

4.2.2 VUCL

Out of all our investigated methods, the VUCL method shows the best performance. Since tuples are only assigned to at most one cell, the indexing step is efficient, between 0.15s and 0.4s for all data set and grid sizes. The interpolation step is more expensive, since it includes the gathering step that computes the distances between the tuples and c_i's center. Nevertheless, it is still acceptable, varying between 3-4 s for 262K and a 768x768 grid with r=5 (maximum radius). For a more practical radius of r=3, the interpolation takes between 1-2s for 262K samples.

4.2.3 kNN

Using a kNN-based list, we investigated both the impact of r and k on runtime performance. Again, one can observe that a larger radius has a significant negative impact on the runtime performance. Although the memory footprint remains limited using kNN (at most k-sized grid cell lists), tuples still are tested

against a rapidly increasing number of cells as radius increases. Ultimately, we selected r=3 as a neighborhood radius for the data sets and grid sizes we tested. Investigating the impact of the k parameter (results not shown), the tests reveal that the chosen k has little impact on the runtime performance of indexing and the interpolation step as well as on the interpolation quality, since the runtimes are very similar for all k={5, 10, 15}.

4.2.4 Comparing overall performance

Figure 2 depicts a summary of our experiments. The methods are tested for all grid sizes, r=3, and the indexing time and interpolation time per methods are added. For kNN the list length=10 was selected. As can be seen, the VUCL approach is the best performing grid cell organization and cell interpolation method (green line) of the tested methods. It outperforms all other methods by a factor of 6 in runtime, and also exhibits the smallest memory footprint, while not suffering interpolation quality. MUCL and kNN are very similar in runtime performance with kNN being slightly faster, a trend that is more pronounced with larger data sets. Due to space limitation, we do not discuss the impact of the selected parameters on the interpolation quality.

5. RELATED WORK

DSMS have been used for other types of sensor data stream management, mostly moving object management in real-time traffic analysis [10], RFID management [11], and some applications also regarding continuous environmental phenomena detection like we propose on this paper [12,13]. Nile-PDT [12] takes a different approach to monitoring phenomena and focuses on events using the MJoin operator [13]. MJoin joins a sensor stream with potentially *m* other streams to match similar values and identify groups of streams with similar values. In our approach, each sensor data stream is used once as input for the interpolated representation that covers the entire query region avoiding expensing m:n joins. Real-time spatial interpolation outside the context of DSMS has caught the attention of research in research areas of geographic information science. The authors of [14] investigate a massively parallel implementation of IDW running on a GPU architecture generating one-time snapshots.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated extending data stream management systems for real-time spatial interpolation of environmental phenomena that are continuous in space and time using up to 260,000 individual mobile sensor data streams. We proposed and evaluated different strategies to optimize a pipelined stream operator approach to achieve near real-time spatial interpolation throughput, considering the memory footprint, runtime efficiency and interpolation quality of the different strategies. We conclude that near real-time spatial interpolation in DSMS is efficient and scalable. Many interesting open questions remain such as optimizing multi-queries and investigate temporal aggregation methods. Adaptive methods to deal with data skew are necessary.

Acknowledgements

This research is based up work supported by the National Science Foundation under grants CAREER-0448183 and IGERT-0504494. The authors also would like to thank Christopher Dorr for his valuable input to this paper.

7. REFERENCES

[1] Campbell, A. T., Eisenman, S. B., Lane, N. D., Miluzzo, E., Peterson, R. a., Lu, H., Zheng, X., et al. (2008). The Rise of People-Centric Sensing. *IEEE Internet Computing*, *12*(4), 12-21.

[2] Campbell, A., & Choudhury, T. (2009). Toward Societal Scale Sensing using Mobile Phones. *Proc. NSF Workshop on Future Directions in Network Sensing Systems*.

[3] Opensense Project, EPFL, Lausanne, http://opensense.epfl.ch/wiki/index.php/Main_Page

[4] Golab, L., & Ozsu, M. T. (2003). Issues in Data Stream Management. *ACM SIGMOD record*, *32*(2), 5-14.

[5] Jain, N., Mishra, S., Srinivasan, A., Gehrke, J., Widom, J., Balakrishnan, H., Cetintemel, U., et al. (2008). Towards a streaming SQL standard. *Proceedings of the VLDB Endowment*, *1*(2), 1379-1390.

[6] Shepard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. *Proceedings of the 1968 23rd ACM Nat Conf* (pp. 517-524). New York, NY.

[7] Renka, R. J. (1988). Multivariate interpolation of large sets of scattered data. *ACM Trans. Math. Softw.*, *14*(2), 139-148. New York, NY, USA: ACM.

[8] Mitas, L., & Mitasova, H. (1999). Spatial interpolation. In P. Longley, M. F. Goodchild, D. J. Maguire, & D. W. Rhind (Eds.), *Geographical Information Systems: Principles, Techniques, Management and Applications* (2nd ed., pp. 481-492). Wiley.

[9] Wang, S., & Armstrong, M. P. (2003). A quadtree approach to domain decomposition for spatial interpolation in Grid computing environments. *Parallel Computing*, *29*(10), 1481-1504.

[10] Mokbel, M. F., & Aref, W. G. (2007). SOLE: scalable on-line execution of continuous queries on spatio-temporal data streams. *The VLDB Journal*, *17*(5), 971-995. doi:10.1007/s00778-007-0046-1

[11] Wu, E., Diao, Y., & Rizvi, S. (2006). High-performance complex event processing over streams. *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06* (p. 407). New York, New York, USA: ACM Press.

[12] Ali, M. H., Aref, W. G., Bose, R., Elmagarmid, A. K., Helal, A., Kamel, I., & Mokbel, M. F. (2005). NILE-PDT: A phenomenon detection and tracking framework for data stream management systems. *Proceedings of the 31st Int' Conference on Very Large Data Bases*, 1295–1298.

[13] Ali, M. H., Mokbel, M. F., Aref, W. G., & Kamel, I. (2005). Detection and tracking of discrete phenomena in sensor-network databases. *Proceedings of the 17th international conference on Scientific and statistical database management*, 163–172.

[14] Henneböhl, K., Appel, M., & Pebesma, E. (2011). Spatial interpolation in massively parallel computing environments. In R. W. T. F. Geertman S. (Ed.), *Proc. of the 14th AGILE International Conference on Geographic Information Science (AGILE 2011)*