Towards Window Stream Queries Over Continuous Phenomena

J.C. Whittier Silvia Nittel Mark A. Plummer Qinghan Liang
School of Computing and Information Science
University of Maine, USA
{john.c.whittier, mark.a.plummer}@maine.edu, {nittel, qliang}@spatial.maine.edu

ABSTRACT

Technological advances have created an unprecedented availability of inexpensive sensors capable of streaming environmental data in real-time. Data stream engines (DSE) with tuple processing rates of around 500k tuples/s have demonstrated their ability to both keep up with large numbers of spatio-temporal data streams, and execute stream window queries over them efficiently. Typically, geographically distributed sensors take samples *asynchronously*; however, when approximating the reality of a continuous phenomenon – such as the radiation field over an urban region- the objective is to integrate their values correctly over space as well as over time. This paper presents an approach to extend DSEs with support enabling *sliding window queries over dynamic continuous phenomena*, which return both *spatio-temporal snapshot and movies* as *window query results*. We introduce a novel *grid-pane index* as a main memory index structure shared between multi-queries over a phenomenon and an *adaptive, data driven kNN algorithm* for efficiently approximating cells based on available stream data samples. AkNN implements a spatio-temporal inverse distance weighting interpolation (IDW) method that integrates time with space via an anisotropic ratio. Further, we introduce the *shell list template* that allows quick calculation of NN cells by distance in a space-time (ST) *cuboid*. We performed extensive performance evaluations using the Fukushima nuclear event in March 2011 as a test data set.

Categories and Subject Descriptors

H.2.8 [**Database Systems**]: Database Applications, *spatial databases, GIS, data streaming.*

General Terms

Algorithms, Design, Experimentation, Management, Measurement, Performance.

Keywords

Scalable spatio-temporal interpolation, data streams system, sensor data streams, continuous phenomena, stream queries, main memory spatio-temporal index, panes.

1. INTRODUCTION

Real-time stream data acquisition via sensors has been widely used in many applications such as intrusion monitoring, manufacturing, disaster response, radioactive accidents, air quality control, pollen monitoring, and in other types of sensor networks [7, 29]. Sensors directly connected to the Internet enable us to collect high frequency updates of potentially thousands of sensors deployed over a geographic area such as a large metropolitan region [23]. Today's commercial data stream engines (DSE) with tuple processing rates of up to 500k tuples/s have demonstrated their capacity to process these types of spatio-temporal data streams efficiently and provide near real-time responses and continuous query answers [6, 19, 31, 32]. Although DSEs have been successfully used for mobile object monitoring in traffic applications as well as RFID tracking [21, 22, 28], support remains limited for phenomena that are continuous in space *and* time such as fields of pollen, radiation or air pollution [4, 12].

Continuous environmental phenomena like gas distribution, rainfall or the dispersion of radioactive particles are seamlessly distributed over a geographic region and gradually change over space and time. Although, there are no 'hard' boundaries (such as those around cars or streets), for many of these phenomena boundaries of high-threshold subregions can be of interest. Conceptually, the phenomenon's change over space and time can be thought of as a 'movie' showing the value distribution smoothly distributed over space changing at very fine-grained temporal steps. Capturing this dynamic aspect was not possible in the past, but given inexpensive, wireless enabled sensors sampling in high spatial density and rapid temporal frequency, it is possible today. One can now easily envision scenarios such as monitoring the pollen distribution in Boston or radiation depositions in Japan or Germany [4, 11].

Our contributions: In this paper, our objective is to investigate DSE support for monitoring phenomena that are continuous both over space *and* time with massive amounts of live sensor streams. We present a novel DSE approach for monitoring dynamic continuous phenomena based on a *scalable stream operator implementation that enables snapshot and movie window query results*. We introduce a *spatio-temporal inverse distance weighting* (st-IDW) method that includes time as a third dimension and accounts for the difference between spatial and temporal distance by using an anisotropic ratio. Our contributions also include an *adaptive kNN stream based algorithm* for *st-IDW* to efficiently approximate grid cells based on available stream samples. We address the typically resource consuming identification of nearest neighbor (NN) tuples in both space and time within the kNN algorithm with two contributions: a novel space-time *grid-pane index with isotropic time cells* and the *shell list template*. The grid-pane index consists of *subpanes* as the smallest temporal grid dimension units making it possible to *search* the index both by a unified Euclidean distance metric as well as to *discard* outdated tuples based on query time. The *shell list template* allows for quick calculation of NN cells by distance to a cell in a *cuboid*. We performed numerous performance tests using data from the Fukushima nuclear event in March 2011 as a test set. The results show that spatio-temporal snapshot window queries with roughly 250K sensors updating with varying frequencies per query window can be computed in less than 4

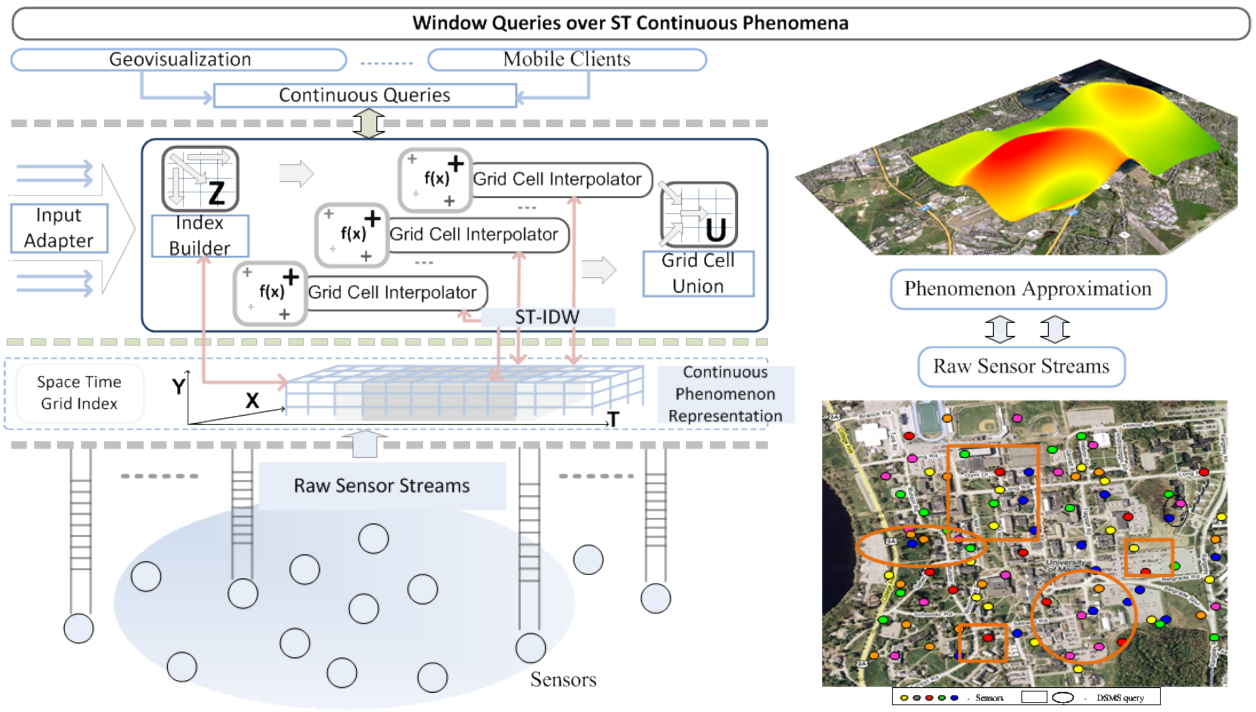

Figure 1. Overview.

seconds on a laptop. Movie window queries delivering 16 frames per query window can be computed in about 10 seconds for the same number of sensors.

The remainder of this paper is structured as follows: Section 2 comprises a short introduction into DSEs. In Section 3, we state the problem. In Section 4 and 5, we present our data stream framework. Section 6 contains the performance tests and results. Section 7 discusses the related work, and Section 8 summarizes the results and discusses future work.

2. A SHORT PRIMER ON DSEs

In this section, we present a short background of current technology in data stream engines (DSEs) and assess the requirements needed to support continuous phenomena monitoring.

2.1 Data stream engines

With the availability of ever smaller and less expensive sensors, pervasive environmental sampling at high frequencies is possible today. Air quality sensors are being attached to mass transit vehicles, used in citizen science efforts to monitor pollen or radiation [23, 27], and even homeland security applications are appearing. Real-world update rates depend on the application, but one can imagine that for a time-critical radiation or chemical plume monitoring scenario, an update rate of 1 update/minute or faster in a dense geographic deployment would be desirable. A cloud-edge architecture [8] with sensors on the 'edge' of the internet streaming samples via the cloud to a centralized data processing system can be used to reduce latency and improve robustness.

A data stream engine is a software system designed to manage continuous data streams as found in real-time financial, emergency or intrusion monitoring applications. Data streams are considered continuous because the amount of data arriving at a particular time is unbounded and cannot be pre-determined. Sensors add timestamps to the samples before sending them, and the DSEs timestamps tuples upon arrival (with a 2nd timestamp). A generic sensor data packet can be thought of as a tuple or a record which contains attributes like <*sensor id, sensor type, timestamp, location, value measured*>. In traditional database management systems (DBMS), queries are evaluated by pulling data from disk, in DSEs this is reversed: here, queries are evaluated over *data pushed* from the stream sources. Moreover, traditional queries are executed one time, whereas queries running in DSE are continuously re-evaluated and often run indefinitely. This reversal of roles for data and query means that new strategies have to be developed to handle new challenges:

Continuous Query Model: Traditional DBMS queries operate on a finite data set (i.e. a relation) and assume set-based data, i.e. they do not have to consider the order of data. However, continuous queries operate indefinitely on an unbounded data set and take the temporal order of arriving data into consideration using an additional specification for query evaluation intervals, also called query windows [5].

Low Latency: Data from streams are generally critical for answering real-time queries, but their significance is often short-lived.

Variable data rate: The data arrival rate for a stream-based application could vary from a few hundred to millions of updates per second. Additionally, the arrival rate can be unpredictable given fluctuations in the sensor update rates or transmission delays. Irrespective of the bottlenecks, DSE query processing must be robust and aimed towards high throughput.

2.2 Window stream queries

CQL [5] is a quasi-standard stream query language derived from and compatible with Structured Query Language (SQL); CQL is supported today by several commercial DSEs [6, 19, 31, 32]. In CQL, both the syntax and semantics of defining a SQL query remain intact, but new operators are specified to support stream processing like STREAM, NOW, JOIN, SLIDE, WINDOW, UNBOUNDED, etc. A *stream* is an unbounded sequence of time-stamped tuples. A *relation* is a bag of tuples at a particular instance in time and conceptually equivalent to a traditional table; hence, this supports a mapping to traditional query operators. A *window* represents a stream interval with a finite set of tuples over which a query is executed. A *slide* is used to define an incremental unit over which an entire window 'moves' before it is re-evaluated. For example, a window with a range of 5 minutes and a slide of 1 minute will re-execute every minute, taking data from the last 5 minutes into account.

In the context of monitoring a continuous phenomenon, sensors pushing data to the DSE provide spatio-temporal point samples of a dynamic field. However, we are more interested in queries over the phenomenon itself instead of over individual or groups of sensor streams. Achieving a *smooth* representation of a continuous phenomenon over a query window requires the use of the set of *available* samples to approximate and fill in the non-sampled points in space and time. This is achieved through spatio-temporal interpolation such as the spatio-temporal versions of Kriging, Inverse distance weighting (IDW) or others [20]; again the specific choice of methods depends on the application needs. For example, the stream query

SELECT RASTER(sensor.val, sensor.loc, st-idw) AS radiation_distr
FROM sensors WINDOW 5min SLIDE 1min
WHERE sensor_type=radiation AND INSIDE(@Japan, sensor.loc);

specifies a stream query that produces a smooth distribution of the radiation over Japan using spatio-temporal IDW to approximate the missing values, and creates a raster representation as output per window. The raster is created every 1 minute, taking collected samples from the last 5 minutes into account.

2.3 DSE support for spatio-temporal window queries

Under a non-blocking stream processing paradigm, a query consists of stream operators arranged as a directed acyclic graph (DAG). The operators are connected via queues, and each stream operator has an in-memory state consisting of any tuples necessary to perform its operation. Note, that for high throughput all data are stored in main memory only, and this includes index structures. Tuples and indices are often shared between operators. Operators work in a pipelined, non-blocking fashion, i.e., an operator pulls tuples from its input queue(s), performs an operation on the relevant set of tuples for the window, and creates an output tuple available for the next operator. All operators work at the same time and with techniques like cloning operators the performance can be scaled up without change to the operator graph itself [24].

After an active research period in DSE technology, today several commercial DSEs such as Oracle CQL [26, 32], Microsoft Streaminsight [1, 2], IBM Infosphere [6], and Streambase [31] are available. In most commercial systems, the stream option is an extension of the relational DBMS product, and libraries that offer spatial support are available for programming stream queries, too. However, the related work shows that since the spatial library functionality is *not* implemented as non-blocking stream operators today and instead adheres to the traditional disk-based processing paradigm, using such libraries for stream queries creates a significant performance bottleneck [1, 14, 19]. Thus, the spatial libraries are not readily usable in real-time applications today.

2.4 Problem statement

To the best of our knowledge, today DSE support for monitoring continuous phenomena in space and time based on sensor data streams is limited, both in academic and commercial systems [3, 25]. Such support needs to be implemented using a pipelined stream operator approach to provide the necessary spatio-temporal functionality for monitoring and analyzing continuous phenomena. This requires a customizable stream operator template so that different streams can be plugged in and operators can be exchanged for customized ones (e.g. varying the interpolation method, or index structures). Overall, the following constraints have to be considered: *first*, main memory is a limited resource as all incoming data and query operators have to share the available memory. *Second*, stream query operators need to scale to potentially very high data rates and new algorithms need to lend themselves to automatic optimization such as cloning of stream operators. *Third*, the stream operators need to be easily customizable for different types of spatio-temporal queries over continuous phenomena as we show in the following sections.

3. INCREMENTAL SPATIO-TEMPORAL WINDOW QUERY EVALUATION

In this section, we discuss the problem of representing window query results over continuous phenomena, introduce our stream-based approach to compute such phenomena, and introduce several different types of window queries over continuous phenomena.

3.1 Representing continuous phenomena with stream queries

Our objective is to investigate DSE support for monitoring phenomena that are continuous over space and time and to enable the monitoring of their *dynamic changes* based on very large numbers of live stationary or mobile sensor streams. We will call this type of phenomenon a *dynamic phenomenon* in the remainder of the paper. Conceptually, a dynamic phenomenon is represented in temporal 'portions' defined by a query window over the input sensor data streams. Two important questions arise: firstly, *how* to represent the 'reality' of the continuous phenomenon *correctly* over time and space within the query window, and secondly, how to deal with asynchronous updates of very large numbers of sensors.

The problem of representing snapshots of a continuous phenomenon *over space* is well established in traditional geographic information systems (GIS). In our previous work [25], we investigated strategies for stream queries over dynamic phenomena with *synchronous* sensor updates at time t_i for all sensors during a query window. This reduces the complexity of representing a dynamic phenomenon correctly since we can create a *spatial snapshot representation at* t_i. We implemented a parallelizable stream operator graph that performs a purely spatial interpolation of missing grid points based on existing updates. However, as this assumption of synchronous updates is relaxed and made more realistic (the focus of this paper), the temporal variation of samples over space must be considered when representing the phenomenon.

In this paper, we assume the following sensor sampling characteristics: individual sensors send samples that have varying update frequencies (varying within a stream and between streams), and sensors do not update synchronously with other sensors. Device locations may also change between updates.

3.2 Approximating phenomena using a spatial-temporal interpolation center

We conceptualize the available samples within a query window as a spatio-temporal point cloud over the observation region and time window[1] (see Figure 2). However, a smooth distribution over space and time representing the dynamic phenomenon with

[1] In this paper, we do not address the potential of sensing noise.

missing values filled in by approximation is the desired stream query output.

In recent years, the problem of spatio-temporal interpolation (ST interpolation) has received much attention, foremost in the area of adding support for moving objects in DBMS and DSE as well as in spatio-temporal databases. For this problem, ST interpolation focuses on approximating the spatial trajectory of a moving object over time and interpolating missing information about the trajectory to determine the object's path as closely as possible.

ST interpolation methods for continuous phenomena are less well established in the spatial database and spatial information science communities. Such methods provide the estimation of unknown values at non-sampled space-time locations [16, 18]. In traditional GIS, ST interpolation methods treat space and time separately. The work in [16] suggests that for certain applications integrating space and time *simultaneously* yields better interpolation results compared to traditional approaches. With a similar motivation in our paper, we adopt the latter strategy.

A query window is defined by a temporal interval [t_S; t_E] with t_S as start timestamp and t_E as the end timestamp. We can reasonably expect window sizes to vary from seconds to hours. Longer windows are possible but might not require a DSE. All samples available during the window are input candidates for the ST interpolation in order to approximate the continuous phenomenon. In our approach, we first make a simplification, and introduce the concept of an *interpolation center* t_{Center}. The interpolation center is a chosen timestamp for the query window, and it is calculated as a relative offset from the start of the window, thus, $t_S \leq t_{Center} \leq t_E$. It is used to create an approximation of the dynamic phenomenon's state at that instant in time. Samples surrounding the interpolation center in space *and* time serve as input for the spatio-temporal interpolation method to generate a representation of the phenomenon's predicted state at that time instant resulting in a 2D representation. The concrete output representation depends on the chosen ST interpolation method, i.e. the dynamic phenomenon could be represented as a raster, contour lines, Voronoi diagram or other alternatives. In the scope of this paper, we use a raster representation produced by an extended *spatio-temporal inverse distance weighting algorithm* (st-IDW), to be introduced later.

Naturally, samples farther in distance and time from the interpolation center have less impact than samples collected temporally closer. In a practical sense, we still achieve a spatial snapshot interpolation but weight samples based on their spatial proximity and temporal distance. We discuss how we use the interpolation center for window approximation in the next section.

3.3 Types of spatio-temporal window queries over continuous phenomena

Using the interpolation center approach, we distinguish the following *window query types* over continuous phenomenon: spatio-temporal ***snapshot*** window queries and spatio-temporal ***movie*** window queries.

3.3.1 Spatio-temporal snapshot window queries

For the *snapshot* window query type, the dynamic phenomenon is represented as a raster-based snapshot capturing the estimated state of the phenomenon at the time specified by the *interpolation center* t_{Center} based on *all* the samples available during the window.

For real-time monitoring applications users are often most interested in the *latest state* of a phenomenon. Hence, the interpolation center t_{Center} is defined at the query window end t_E. Since any samples taken towards the end of the window have more impact on the predicted snapshot, placing the interpolation center at the end of the window query has a *time decaying* effect of earlier samples collected at the beginning of the window.

For applications that are focused on monitoring and potentially archiving dynamic phenomena, choosing an interpolation center in the middle of the window, i.e. (t_S + t_E)/2, provides the best 'summary' of the phenomenon's changes during the window, considering both samples before and after t_{Center} (until the end of the window) for interpolation. While the 'end of window' and the 'middle of window' are the two most likely interpolation centers, there are no restrictions on when to place t_{Center} within the query window.

3.3.2 Spatio-temporal movie window queries

The snapshot window query produces a single raster per window, which also means that detailed information on the phenomenon's temporal variation within the window is lost. An intuitive way to capture this variation is a 'movie' of the phenomenon, i.e. a series of snapshots or 'frames' within a window.

Each 'frame' of the movie is a spatial raster produced using ST interpolation. However, the input for each interpolation is smaller, i.e. a fraction of the window size. For example, a five minute window can be represented at different 'frame' rates such as one frame every 30 seconds (10 frames overall), or one every minute (5 frames overall). Each frame can be computed using time decaying interpolation centers at the end or middle of the window subinterval, depending on the application needs.

One serious problem for the movie representation can be a lack of available samples for short, non-overlapping subintervals. In this case, the approximation of each 'frame' will be coarse. This problem is discussed in the performance evaluation section.

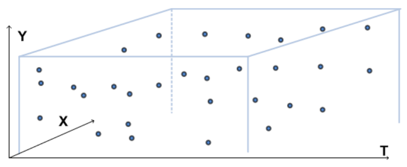

Figure 2. Input sensor samples over space and time

3.4 Spatio-temporal IDW

For this paper, we chose inverse distance weighting (IDW) [30] as the interpolation method because of its linear scalability with the size of the prediction grid and the number of sample points. While traditional spatial IDW does not consider time, we extended the method and added time as another spatial distance dimension. To include time correctly [16], we use an anisotropy ratio between time units and spatial units to account for the difference temporal 'distance' versus spatial 'distance' so that they can both ultimately be treated in the same way. The temporal difference ('distance') between the observation time point t_t of a sensor tuple and the interpolation center t_{Center} is multiplied by the inverse of the anisotropy ratio *a* to convert the temporal 'distance' to a 'spatial' distance. This yields an overall 3D distance calculation for each cell $cell_c$ approximated at t_{Center} of

$$d(cell_c, tuple_t) = \sqrt{(x_c - x_t)^2 + (y_c - y_t)^2 + \left(\frac{t_{center} - t_t}{a}\right)^2}$$

Parameters to st-IDW additionally include a power factor, a varying spatial search radius around a cell, and in some cases a threshold level of k sample points to use to interpolate each grid cell. The anisotropy ratio is currently determined experimentally.

3.5 Discussion

For each cell in the output grid, st-IDW requires fast access to all relevant tuples that are close to the cell in space and time. Thus, incoming tuples must be indexed in three dimensions. If a single grid index is to be used for all query windows (and queries), expired tuples will need to be removed from the index and new ones added continuously. The identification of old tuples from a window is a potentially costly operation, so the index structure should be designed to be efficient to remove old data and add new data. Additionally, the index needs to be stored solely in main memory to maintain high throughput.

4. SPATIO-TEMPORAL GRID-PANES

4.1 Shared execution of spatio-temporal window queries

In a DSE, incoming streams are shared between multiple queries over the same input streams. Although each stream query might run separately (i.e. queries do not share operators), auxiliary structures for operators that organize/index portions of the stream are shared between queries [22].

Our DSE approach for supporting the monitoring of dynamic phenomena consists of two components: *first*, a shared spatio-temporal main memory-based index structure that organizes incoming tuples in space and time to prepare them for fast identification during spatio-temporal interpolation, and *second*, a scalable, adaptive, main memory-based stream operator to predict dynamic phenomena using spatio-temporal inverse distance weighting (st-IDW). Our design considerations and the novel components are presented in the remainder of this section.

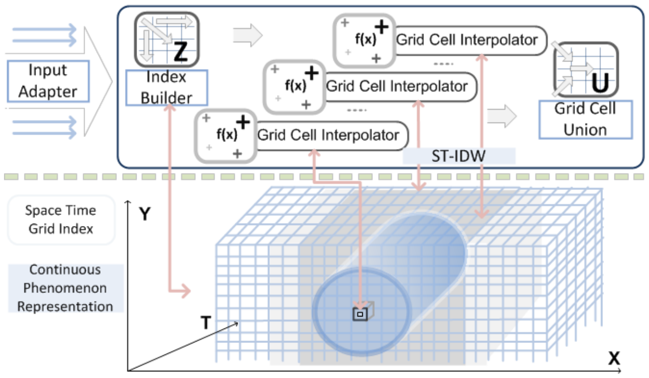

Figure 3. Stream operator graph for ST-interpolation

4.2 Incremental evaluation of spatio-temporal windows

Remember the example from Section 2.2

SELECT RASTER(sensor.val, sensor.loc, st-idw) AS radiation_distr
FROM sensors WINDOW 5min SLIDE 1min
WHERE sensor_type=radiation AND INSIDE(@Japan, sensor.loc);

This stream query creates a raster using st-IDW considering all incoming sensor updates from the last 5 minutes (the location of interpolation center is by default at the window end). At time t_0 the operator waits until all samples in the interval [t_0; t_5] are received; then, it resumes to create the output raster. The operator then 'slides' over the data by one minute, and produces a new raster using the tuples with timestamps in the interval [t_1; t_6]. Essentially, the operator discards all tuples with timestamps in the interval [t_0; t_1] in its operator state and adds the tuples that arrived between [t_5; t_6] to its state. This is called incremental query window evaluation. It is crucial to identify which tuples are still valid, which tuples are newly added, and which are to be discarded tuples quickly to achieve fast execution of the next interpolation step.

In [15] the initial idea of window 'panes' was introduced; 'panes' support the grouping of tuples based on time 'groups' which can correspond to the slide of windows. All the panes that make up the slide contain all tuples relevant for the increment of a slide. We extended this concept in our approach, combineing and adjusting it with a spatial grid search index needed to identify tuples based on location and time.

In the following, we describe the proposed stream operator algorithm, depicted in Figure 3 (upper box).

4.3 Operator I (Indexing): Shared spatio-temporal grid-pane index

A space-time grid index is created to group the sensor tuples for a window in both space and time. Once a window slides for the next interpolation, invalid tuples can be purged from the space-time grid by eliminating one 'slice' over time, and new cells can be added.

Panes [15] can be integrated into a grid index by simply adding another dimension to the grid. This can be achieved in one of three ways: the grid becomes a) a 3D array of ST grid cells, or b) a 1D array of panes and each pane contains a 2D grid, or c) a 2D grid in which each grid cell contains an array of panes. In our approach we chose the first option. The index can be visualized as a rectangular cuboid (see bottom of Figure 3) in which a (x, y) grid column slides over the temporal axis. In a single stream query setting, the spatial size of the grid cell is equal to the size of a cell of the output grid (e.g. 512 x 512 cells over the observation region). For a multi-query environment this spatial aspect of the grid size is variable. Similarly, the length of a cell along the temporal axis defines a temporal 'resolution' and depends on the query itself (i.e. range and slide). In our approach, the length of a cell along the temporal axis is equal to one *subpane* (more details on subpanes are described in 5.3).

To eliminate the need for creating new panes and removing old panes, a circular buffer is used in the space-time grid index that allows panes to be reused. Since a window contains a fixed number of panes it is advantageous to be able to reuse the grid cells in each pane. When a window slides, and the query is reevaluated, the outdated cells in the grid panes index are cleared directly.

The ***first sub-operator*** of the dynamic phenomenon stream operator (see Figure 3) scans the incoming tuples from the input adaptor and inserts pointers to the tuples in the grid-pane index based on location and the original sensing timestamp.

4.2	3.6	3.2	3	3.2	3.6	4.2
3.6	2.8	2.2	2	2.2	2.8	3.6
3.2	2.2	1.4	1	1.4	2.2	3.2
3	2	1	0	1	2	3
3.2	2.2	1.4	1	1.4	2.2	3.2
3.6	2.8	2.2	2	2.2	2.8	3.6
4.2	3.6	3.2	3	3.2	3.6	4.2

Figure 4. Shell list template: Nearest neighbors by cell distance in 2D

4.4 Operator II (finding NN cells): Using the spatio-temporal shell template

In our previous work [25], we determined that the 'Virtual List' (VL) approach has the best performance when performing spatial

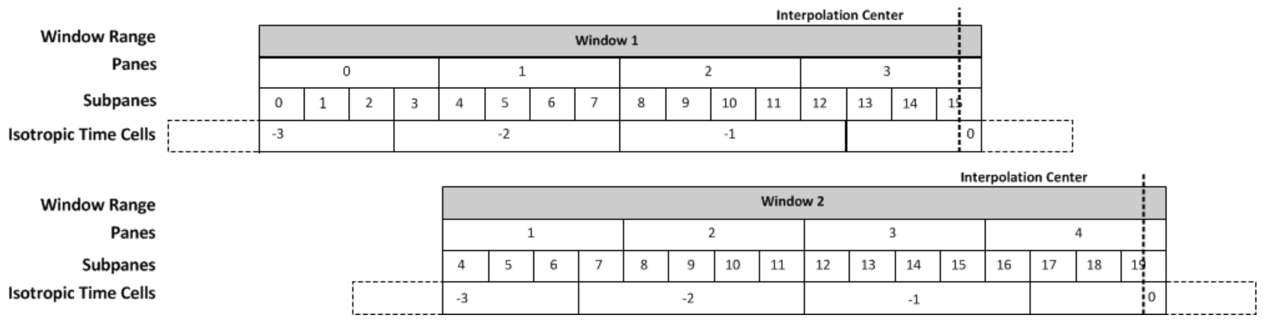

Figure 5: Subpanes

interpolation in a stream-based setting. To interpolate the value of a grid cell, data within a specified spatial and temporal distance to the cell are used to calculate an approximate value. In the Virtual List approach, these tuples are determined by traversing the grid index, and the input tuples are never materialized nor are references copied. Adapting VL to the st-IDW method, the VL now needs to traverse the cells based on distance in the grid index *cuboid*.

To find cells in the order of their distance from a central cell (the cell for which the value needs to be interpolated), we introduce a cylinder shaped ***Shell List*** template. The Shell List is constructed by generating all combinations of coordinates that fall within the search radius and time window and sorting the list based on the cells' distance to the center of the cell being interpolated. The Shell List is a two-leveled hierarchical list of lists describing successive concentric 'shells' of successive distances from a central cell. Each sub-list collects the relative array addresses of the cells that are equidistant to the center; the top-level list orders the sub-lists by their distance from the center. This list is structured by including only cells within the spatial radius and the query time window range. This constrains the shells to expand only to the shape of a cylinder (see Figure 3 for the cylinder shape of the Shell List and Figure 4 for a 2D-based Shell List).

At this point the second level hierarchy is built by collecting coordinates, which are at the same distance (i.e. they are sequential in the list because of the sorting). The time complexity for this algorithm is $O(r^2t\log(rt))$ where r is the limiting radius of the search or equivalently $O(n\log(n))$ where n is the number of cells and t is the length of the time window. The Shell List has the potential to become quite large, but a *single* list suffices for each cell of the entire grid index since the list consists of relative addresses (thus, Shell List *template*). For example, for a central grid cell (i.e. a cell to be interpolated), its x and y coordinates and the interpolation center are combined with each set of (x, y, t) relative addresses in the Shell List to determine the actual cell addresses in the grid where the next tuples to be considered are stored.

The VL algorithm uses the Shell List template to determine for each cell the relevant input cells to be checked for tuples to be used for interpolation. Overall, the data used to approximate the value of a grid cell can be visualized as a cylinder surrounding that grid cell with height equal to the window range and a radius equal to the spatial search radius. All tuples from cells having cell centers within this cylinder are used to interpolate the output grid cell.

4.5 Operator III (Interpolate): ST inverse distance weighting

Figure 3 depicts a set of ***grid cell interpolators*** accessing the relevant cells and their tuples determining their addresses based on the Shell List template. The operators are independent of each other, yet they share read access to the same grid-pane index. The operators have the potential for parallel execution since their functionality is the most time-consuming in the overall interpolation. Once a grid cell has determined all the relevant input tuples by space and time, it generates the value based on the function mentioned in Section 3.4.

The ***Grid Union operato***r, which is a consumer operator of all the grid cell interpolators, assembles the grid for a particular window, and creates a raster as a window-based output. The raster can be visualized, stored or used as input for further stream queries.

4.6 Discussion

The stream operator graph supports any choice of interpolation centers; a common interpolation center is at the end of the window as depicted in Figure 5. The mid-window interpolation center is located in the center of the depicted cylinder (see Figure 3), and the search expands in both direction of the temporal cylinder. The VL algorithm performs well if the search radius r is small with a given window range, but as the radius increases, the number of grid cells that need to be accessed and the number of tuples to be processed increases significantly. In cases when samples are sparse, it might be necessary to require a large preset spatial search radius until sufficient sample data are found, and even simple checking of cells for tuples is expensive. If, however, tuples are abundant, a small search radius is sufficient.

5. ADAPTING TO DATA DISTRIBUTION

Based on the previous discussion a VL operator using a preset search radius can become inefficient for sparse and skewed data sets. In this section, we introduce an improved algorithm that adapts to the available data distribution automatically.

5.1 Adaptive kNN Algorithm

Since st-IDW assigns more weight to samples closer in space and time than to those farther away, samples in greater distance contribute statistically less information about the value of the cell being predicted. At some point, any further samples can be disregarded without affecting the error of the interpolation beyond some tolerance. We exploit this and propose the *Adaptive kNN* (k Nearest Neighbor) approach to st-IDW interpolation. As described above, for each cell in the output raster, tuples from successively more distant cells (distant in both time and space) are used to determine the predicted value, but when a specified number of tuples have been processed, i.e. the closest k tuples, the operator terminates interpolation for this cell.

5.2 Isotropic grid panes

Again, the Shell List template is used to determine the NN cells. The Shell List template is isotropic, i.e. is it treats all dimensions equally with respect to calculating distance. Here, the width of a cell in the x spatial dimension is the same as the width in the y spatial dimension and is the same as in the time dimension as well. Each of the dimensions participates equally in the Euclidean calculation. However, this isotropic view of time does not necessarily conform to the layout of the underlying three

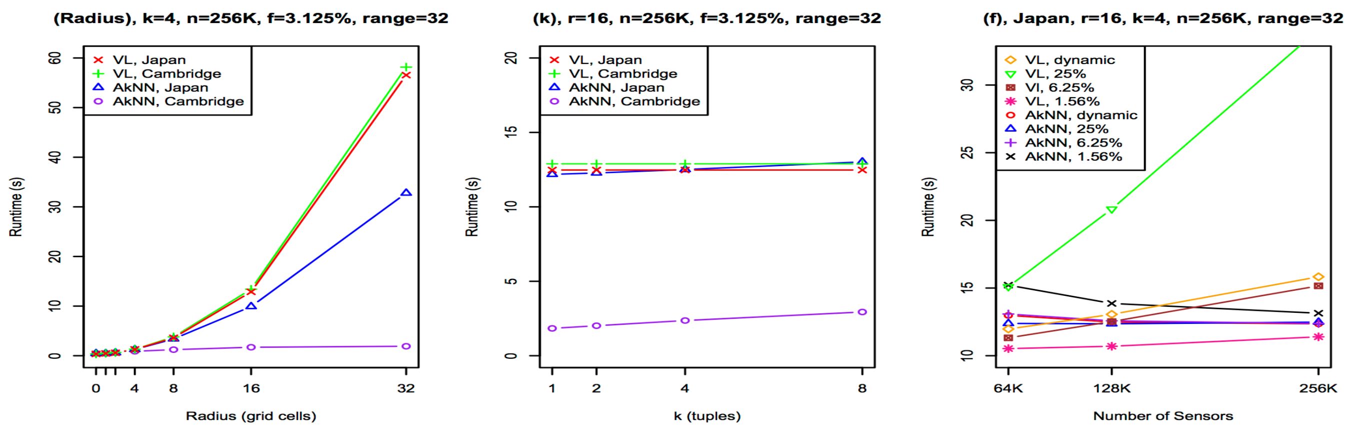

Figure 6: a) impact of neighborhood cell radius r b) impact of parameter k c) impact of temporal sampling frequency

dimensional grid structure: seconds are not equivalent to meters, and even more relevant, panes determined by the window query do not have an equivalent weight when calculating the distance to a grid cell to be approximated. While the x and y spatial dimensions are assumed to be isotropic (distance along one is equivalent to distance along another), time is related to spatial distance through use of an anisotropic ratio (in the distance equation in 3.2). The anisotropic ratio depends on the dynamic behavior of the phenomenon; we can adjust the weight of the temporal units based on the anisotropic ratio. For example, representing a relatively stationary phenomenon over time we choose a 'coarser' cell width along the temporal dimension (e.g. 1 hour instead of 1 minute) capturing a 'slow change' over time. On the other hand, with a fast changing phenomenon we select a 'fine' temporal resolution to capture the change in more detail. The anisotropic ratio is specified in time units per distance units. Since the Shell List is isotropic, we need to adjust the space-time index correctly for the Shell List to find the kNN cells.

5.3 Subpanes

In window queries, the basic time 'unit' is the window interval of the continuous query; additionally, a query window can have an incremental slide. In our framework, the slide is represented by the (time) pane, i.e. the organizational time unit used to add and discard tuples. However, since time and space have a different impact on the interpolation, the 'alignment' of both is achieved by the scaling factor of the anisotropy ratio. Thus, per window, we introduce *two types of panes*: a) the original pane that relates to the slide and the discarding of outdated tuples, and b) an 'aligned' time unit (pane) that corresponds to the Euclidian distance (further called the isotropic time cell). In order to align both the original pane and the isotropic time cell, we introduce subpanes (see Figure 5), which are the basic grid cell time unit. ***Subpanes*** are the smallest units to hold data that can be mapped in multiples to both panes as well as to isotropic time cells. In order to determine the relevant cells for adding and discarding data as well for finding kNN cells, a coordinate transformation must exist between isotropic time cells and subpanes as well as between (time based) panes and subpanes. For example, if the interpolation center is defined at the end of window with a range of 16 time units, the Shell List determines the search for tuples starting with the center of isotropic time cell 0, and expands its search radius to the surrounding isotropic time cells (see Figure 5). A single isotropic time cell corresponds to a multiple of subpanes (here, five subpanes). On the other hand, once the window slides, data are discarded in units of a time pane, which also correspond to multiples of subpanes (here, four subpanes).

6. PERFORMANCE EVALUATION

6.1 Experimental Setup

Since sensor data streams in high spatial and temporal density are not available (yet) for the system we envision, we simulated mobile sensors along a street network of varying density in two geographically different study regions: Cambridge, MA, with a dense road network and a region of Japan surrounding the 2011 Fukushima nuclear incident.

6.1.1 Data sets

To generate high-density data streams, a simulation of moving objects sampling the phenomenon in their environment and creating sensor streams was implemented in NetLogo. The movement of sensors is constrained to links in a street network, in our test cases the networks are a large region of Japan and a small, but densely sampled region of the Cambridge and Boston area. The phenomenon we considered represents the estimated radiation deposition levels in Japan after the Fukushima Daiichi nuclear disaster in March 2011. The predicted radiation levels were calculated in R using data from ZAMG [33] and SPEEDI [9]. The simulation iterates over a sequence of snapshots of the event between the 4th and 5th day after the disaster in 15-minute sampling increments, and a query window is defined as 8h. In a realistic system, we would expect each time unit to correspond to 1 minute or less. The data sets were generated for 1M moving sensor nodes.

6.1.2 Implementation and Runtime Environment

The proposed strategies were implemented in Java in a limited DSE environment, i.e. operators are connected via queues, and work in a pipelined fashion, but we do not consider any of the other DSE components. The experiments were run on a MacBook Pro with a 2.3 GHz Intel Core i7 (Model MacBookPro9.1; a quad core processor with eight virtual cores), 8 GB DDR3 memory at 1600 MHz and Mac OS X 10.8.4 (12E55) (64 bit) and Java 1.6.0_45 (64 bit).

6.2 Adaptive vs. full 'search' of grid cells

For the following tests, a sliding window with a range of 32 time units and a slide of 4 time units was used. Tests were run with ten iterations of four consecutive windows over both the Cambridge and Japan data sets. The data sets composed of all simulated sensor observations have 256K sensors and each of them had a probability of 1/32 (3.125%) of updating for a particular time unit (i.e. on average each sensor node updates once in the window; we

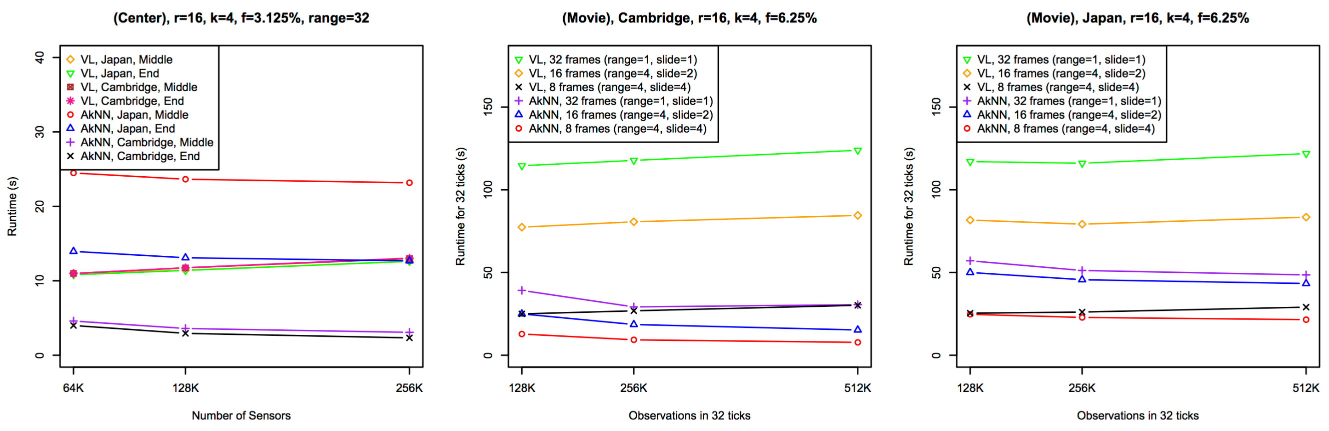

Figure 7: a) position of interpolation center and a movie window query over b) Cambridge and c) Japan

investigated other update frequencies later in the performance section). The interpolation center is set at the end of the window.

6.2.1 *Impact of neighborhood radius size*

First, we investigated the impact of the neighborhood *radius* selected on runtime for VL (virtual list) and AkNN. We tested radii = {0, 1, 2, 4, 8, 16, 32}. For AkNN, we ran tests with k=4. The results are depicted in **Figure 6a** and show that for all configurations there is an (expected) increase in runtime as the search radius is increased.

For **VL**, the *runtime* in the Cambridge and Japanese data sets are very similar (as expected) since VL uses *all* data contained within all grid cells in the space-time cylinder. VL, however, does not scale well once the search radius increases. **AkNN** scales better than VL, but we observed a significant difference in runtime between the Cambridge and Japan datasets. When data in grid cells are sparse, e.g. sampled only along a street network, the AkNN search needs a large search radius to be able to find enough data (r~35 cells). For Cambridge with a dense street network, AkNN can exploit the higher data density and stop searching once k data tuples have been found (r ~ 4-5 cells).

6.2.2 *Impact of parameter k for kNN*

In AkNN, the parameter k can be used to limit the search for tuples of a total of kNN tuples. We tested the impact of k = {1, 2, 4, 8} on runtime for both Cambridge and Japan. A radius of 16 was used based on results from 6.2.1. VL does not use a parameter *k*, so we generated a straight line for VL for both data sets as comparison. For AkNN, runtime increases slightly as k is increased in the densely sampled Cambridge data set. However, AkNN is slower than VL in Japan for k=8. This is due to the introduction of isotropic time cells and subpanes, which results in more cells in time (32 vs. 8) to search through.

6.2.3 *Mid-window vs. end-window snapshots*

In this group of tests we investigated different ways of interpolating the data within a query window. Our st-IDW produces a snapshot of a phenomenon at a particular time instant (i.e. interpolation center). We tested the performance difference of choosing a specific time at the *middle* or at the *end* of the window. All tests were run with radius=16 and k=4. In VL, all data for the window are found in the same way regardless of the position of interpolation center, so very similar runtimes are expected. In AkNN, however, using an interpolation center at the end of the window results in a faster runtime. This effect comes from the fact that data are only to be found in grid cells that intersect the street network. Therefore, although we are performing an expanding search in both space and time, it is the expansion in space that is most likely to result in encountering data. With an interpolation center at the end of the window, we search in an expanding hemisphere. The search only navigates in one direction, towards the beginning of the window, so fewer grid cells need to be checked in expansion. This results in AkNN finding data in less time with an interpolation center at the end of the window than if the interpolation center was in the middle of the window.

6.2.4 *Movie window queries*

We tested several "movies" query results representing the evolving change of the phenomenon over the entire window. All tests were run with r=16 and k=4. The original query window has a length of 32 time units with a slide of 4 time units. We investigated dividing a window into a sequence of result 'frames' and a sensing frequency of 1/16 (6.25%) per tick with sensor populations of 64K, 128K, and 256K. The query results were presented in movies of 8, 16, or 32 frames/window. The runtime represents the total time to generate all frames within the 32 time unit window. As we can see (Figure 7 b and c), generating a 'movie' representation over the Cambridge data set with 8 frames over 32 time units using AkNN required approximately 8 sec. A movie with 32 frames and about 500K sensors observations takes about 35 seconds. These results do not include any parallelization of the grid cells interpolator.

6.3 Varying temporal sampling

The final tests investigate changes in the number of sensors sampling at a particular time. We tested fixed sampling rates where 1/64, 1/16, and 1/4 (1.56%, 6.25%, and 25%) of the sensors updating at each time unit. We also tested dynamic sampling in which the percentage of sensors sampling changed at each time unit. To allow for reproducibility among runs, a static list of random probabilities was generated with values between 1/64 and 1/4 and with a mean of approximately 1/16. VL showed that an increase in sampling rate and number of sensors leads to increased runtime. In AkNN, however, the opposite behavior was observed. Here, an increase in the number of observations in a window, either due to a higher sample rate or an increase in the number of sensors, resulted in a decrease in the runtime (more samples, short search for k).

6.4 Summary

Figure 8 depicts the original estimated radiation deposits over the Fukushima region on day 5 after the March 15 2011 nuclear incident in the top left corner. The darker regions are radiation fallout accumulation, and the dark dots represent moving sensors on the road network. The remaining figures show approximations of the fallout using st-IDW with consideration radius of 10, 20 and 30 cells around the predicated cell (subfigures are in clockwise order). Due to the sparse road network the approximation is coarse, but the deposits are clearly identifiable.

Our performance tests show that snapshot queries using AkNN run in under ten seconds per window with a search radius of 16 cells around a cell to be interpolated, a query window of 32 time units, roughly 256K observations, and sparse sampling in a large region along a street network (Japan). Not only is it feasible to produce a snapshot interpolation representing a window, we have also shown that it is possible to produce a sequence of frames (movie) representing the continuous change over time of a phenomenon within a query window. We found a significant difference in performance using an interpolation center at the middle of a window compared to the end of the window with the latter being more efficient due to searching in an expanding hemisphere rather than sphere. Finally, our tests show that new approaches are needed to handle sparse and skewed data collected along street networks.

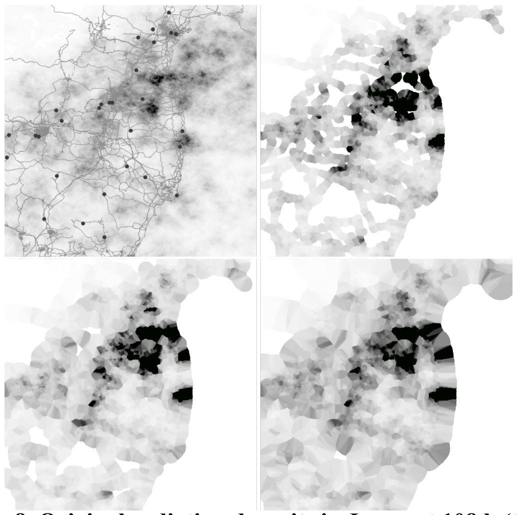
Figure 8. Original radiation deposits in Japan at 108 h (top left) and approximation with AkNN, search radius (8, 16, 32) (clockwise)

7. RELATED WORK

DSEs have been used for handling sensor data streams in space and time in several sensor-based application domains, mostly in real-time moving object management [21, 22]. However, this type of DSE support is concerned with tracking individual objects, and answering mobile queries efficiently using a shared spatial index. The need for a shared grid-based index is similar to our work; however, the purpose of our spatio-temporal pane based index is different since it is designed to efficiently identify input tuples for ST interpolation. Nile-PDT [3] introduces the MJoin operator to detect and monitor events in continuous phenomena; it compares n:m sensor data streams searching for similar values and identifies regions once streams with similar values are detected. In our approach, all sensor data streams are interpolated into a smooth spatio-temporal representation. We currently focus on representing the entire continuous phenomena, not the events.

The approach of dividing overlapping sliding windows into equal “panes” was first introduced in [15]. The original idea was to efficiently evaluate continuous aggregate queries by reducing space and computation time. In this paper, we utilize the concept of panes to divide the sensor data streams into equal sized ‘slices’ based on which we build a 3D grid index. However, we introduce two new types of panes, first, the so-called *isotropic time cell*, which aligns the spatial cell dimensions with time cell dimension via an anisotropic mapping, and secondly, the unit of subpanes, which allows easy mapping between query window panes and ‘grid panes’. In DBS for moving objects [10], spatio-temporal interpolation functions are proposed to estimate the locations at any timestamp. Nevertheless, only ST-interpolations for individual moving object and trajectories are discussed. Few researchers have addressed spatio-temporal interpolation of continuous phenomena [18]. In traditional GIS, space and time are treated separately in spatio-temporal interpolation. For certain applications, integrating space and time simultaneously yields better interpolation results compared to the traditional approaches, and the need for anisotropic ratio has also been addressed in [16]. [16–18] use shape functions based on finite element methods to integrate space and time simultaneously. In recent years, real-time spatial interpolation has caught the attention of research in other areas of geographic information science, e.g. [13] investigates the redesign of IDW for parallelization and runs the process using a single GPU. However, these approaches are outside of the context of DSE for processing streaming sensor data and focus solely on spatial interpolation. Our previous work [25] has addressed challenges of spatially interpolating sensor update streams in near real-time in the DSE framework. This paper focuses on the more realistic assumption that sensors update asynchronously over a window, and we have presented an approach for highly efficient stream window based *spatio-temporal* interpolation.

8. CONCLUSIONS AND FUTURE WORK

Today, the unprecedented availability of inexpensive sensors has enabled us to collect massive amounts of asynchronously updating environmental data streams in real-time. DSEs have demonstrated their capacity to keep up with throughput of up to 500 tuples/s. In this paper, we presented a novel approach to extend DSEs to support the monitoring of dynamic continuous phenomena and *enable snapshot and movie window queries*. Our contributions include an *adaptive kNN stream based algorithm* for *st-IDW* to efficiently approximate grid cells based on available stream samples. We address the resource consuming identification of NN tuples in both space and time within st-IDW with two contributions: a novel, shared space-time *grid-pane index with isotropic time cells* and the *shell list template*. The *shell list template* allows quickly calculating NN cells by distance in a ST *cuboid*. The grid-pane index consists of *subpanes* as the smallest temporal grid dimension unit and make it possible to achieve two tasks: a) *searching* the index by a unified Euclidean distance metric using the shell list, and b) *discarding* outdated tuples based on query time. We performed extensive performance evaluations using the Fukushima nuclear event in March 2011 as test data. The results show that spatio-temporal snapshot window queries with about 250K sensors updating in high frequency per window query can be computed in less than 4 seconds on a laptop. Movie window queries with 16 frames per window can be computed in about 10 seconds. Future work includes identifying strategies to improve ST interpolation of spatially sparse sampling.

Acknowledgements

This material is based up work supported by the National Science Foundation under grant NSF DGE-0504494. The authors also would like to thank Balaij Venkatesan and Sytze de Bruin for their valuable input to this paper.

9. REFERENCES

[1] Ali, M. et al. 2010. Real-time spatio-temporal analytics using Microsoft StreamInsight. *Proceedings of the 18th SIGSPATIAL International Conference on Advances in*

Geographic Information Systems - GIS '10 (New York, New York, USA, 2010), 542–543.

[2] Ali, M. et al. 2011. The extensibility framework in Microsoft StreamInsight. *2011 IEEE 27th International Conference on Data Engineering* (Apr. 2011), 1242–1253.

[3] Ali, M.H. et al. 2005. NILE-PDT: A phenomenon detection and tracking framework for data stream management systems. *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)* (2005), 1295–1298.

[4] Amano, H. et al. 2012. Radiation measurements in the Chiba Metropolitan Area and radiological aspects of fallout from the Fukushima Dai-ichi Nuclear Power Plants accident. *Journal of Environmental Radioactivity*. 111, (2012), 42–52.

[5] Arasu, A. et al. 2005. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*. 15, 2 (Jul. 2005), 121–142.

[6] Biem, A. et al. 2010. IBM InfoSphere Streams for Scalable , Real-Time, Intelligent Transportation Services. *SIGMOD '10: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (Indianapolis, IN, 2010), 1093–1103.

[7] Campbell, A.T. et al. 2008. The Rise of People-Centric Sensing. *IEEE Internet Computing* (Jul. 2008), 30–39.

[8] Chandramouli, B. et al. 2012. RACE : Real-time Applications over Cloud-Edge. *SIGMOD '12: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), 2–5.

[9] Chino, M. et al. 1993. SPEEDI and WSPEEDI: Japanese Emergency Response Systems to Predict Radiological Impacts in Local and Workplace Areas due to a Nuclear Accident. *Oxford Journals Mathematics & Physical Sciences Radiation Protection Dosimetry*. 50, 2 (1993), 145–152.

[10] Erwig, M. et al. 1999. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica*. 3, 3 (1999), 269–296.

[11] Heuvelink, G.B.M. et al. 2010. Optimization of mobile radioactivity monitoring networks. *International Journal of Geographical Information Science*. 24, 3 (Mar. 2010), 365–382.

[12] Huan G, F. et al. 2008. A Field-based Database Management Method for City Air Pollutants Information System. *IEEE International Geoscience and Remote Sensing Symposium (GARSS 2008)*. 3, (2008), 1296–1299.

[13] Henneböhl, K., et al. 2011. Spatial interpolation in massively parallel computing environments. *Proc. of the 14th AGILE International Conference on Geographic Information Science (AGILE 2011)*. R.W.T.F. Geertman S., ed. Springer Berlin Heidelberg.

[14] Kazemitabar, S. et al. 2010. Geospatial stream query processing using Microsoft SQL Server StreamInsight. *Proceedings of the VLDB Endowment*. 3, 1-2 (2010), 1537–1540.

[15] Li, J. et al. 2005. No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. *ACM SIGMOD Record*. 34, 1 (2005), 39–44.

[16] Li, L. et al. 2011. Spatiotemporal Interpolation Methods for Air Pollution Exposure. *SARA*. (2011), 75–81.

[17] Li, L. and Revesz, P. 2004. Interpolation methods for spatio-temporal geographic data. *Computers, Environment and Urban Systems*. 28, 3 (May. 2004), 201–227.

[18] Li, L. and Revesz, P.Z. 2002. A Comparison of Spatio-temporal Interpolation Methods. *GIScience '02: Proceedings of the Second International Conference on Geographic Information Science*, 145–160.

[19] Miller, J. et al. 2011. An Extensibility Approach for Spatio-temporal Stream Processing using Microsoft StreamInsight. *SSTD'11: Proceedings of the 12th international conference on Advances in spatial and temporal databases* (2011), 496–501.

[20] Mitas, L. and Mitasova, H. 1999. Spatial interpolation. *Geographical Information Systems: Principles, Techniques, Management and Applications*. P. Longley et al., eds. Wiley. 481–492.

[21] Mokbel, M. et al. 2004. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. *Proceedings of the 2004 ACM SIGMOD* (2004), 623–634.

[22] Mokbel, M.F. and Aref, W.G. 2007. SOLE: scalable on-line execution of continuous queries on spatio-temporal data streams. *The VLDB Journal*. 17, 5 (Apr. 2007), 971–995.

[23] Murty, R.N. et al. 2008. CitySense: An Urban-Scale Wireless Sensor Network and Testbed. *2008 IEEE Conference on Technologies for Homeland Security*. (May. 2008), 583–588.

[24] Ng, K. et al. 1999. Dynamic Query Re-Optimization. *SSDBM '99 Proceedings of the 11th International Conference on Scientific and Statistical Database Management* (1999).

[25] Nittel, S. et al. 2012. Real-time spatial interpolation of continuous phenomena using mobile sensor data streams. *Proceedings of the 20th International Conference on Advances in Geographic Information Systems - SIGSPATIAL '12* (New York, NY, USA, 2012), 530-534.

[26] Oracle, A. and Paper, W. 2008. Oracle Complex Event Processing Performance. November (2008).

[27] Paulos, E. et al. 2008. Citizen science: Enabling participatory urbanism. *Urban Informatics: community Integration and Implementation*. 1–16.

[28] Piao, Y. et al. 2009. Optimization of continuous query processing for RFID sensor tag data stream. *Proceedings of the 2nd International Conference on Interaction Sciences Information Technology Culture and Human ICIS 09*. (2009), 586–591.

[29] Resch, B. et al. 2009. Real-Time Geo-awareness – Sensor Data Integration for Environmental Monitoring in the City. *2009 International Conference on Advanced Geographic Information Systems & Web Services* (Feb. 2009), 92–97.

[30] Shepard, D. 1968. A two-dimensional interpolation function for irregularly-spaced data. *Proceedings of the 1968 23rd ACM National Conference* (New York, NY, USA, 1968), 517–524.

[31] StreamBase 2013. StreamSQL online documentation.

[32] Witkowski, A. et al. 2007. Continuous queries in Oracle. *VLDB '07 Proceedings of the 33rd international conference on Very large data bases*. (2007), 1173–1184.

[33] Wotawa, G. and Skomorowski, P. 2012. Long-range transport of particulate radionuclides from the Fukushima NPP accident: sensitivity analysis for wet deposition. *EGU General Assembly 2012* (Vienna, Austria, 2012), 10494.