A 6-DIMENSIONAL HILBERT APPROACH TO INDEX FULL WAVEFORM LiDAR DATA IN A DISTRIBUTED COMPUTING ENVIRONMENT

A.V. Vo[1], N. Chauhan [1], D.F. Laefer [1,2,*], M. Bertolotto[3,4]

[1] Center for Urban Science + Progress, New York University, USA - (anhvu.vo, nc1682, debra.laefer)@nyu.edu
[2] Dept. Civil and Urban Engineering, Tandon School of Engineering, New York University, USA
[3] School of Computer Science, University College Dublin, Ireland – michela.bertolotto@ucd.ie
[4] Earth Institute, University College Dublin, Ireland

Commission IV, WG IV/7

KEY WORDS: aerial laser scanning, full waveform, LiDAR, spatial database, distributed database, spatial indexing, Hilbert, high dimensional

ABSTRACT:

Laser scanning data are increasingly available across the globe. To maximize the data's usability requires proper storage and indexing. While significant research has been invested in developing storage and indexing solutions for laser scanning point clouds (i.e. using the discrete form of the data), little attention has been paid to developing equivalent solutions for full waveform (FWF) laser scanning data, especially in a distributed computing environment. Given the growing availability of FWF sensors and datasets, FWF data management solutions are increasingly needed. This paper presents an attempt towards establishing a scalable solution for handling large FWF datasets by introducing the distributed computing solution for FWF data. The work involves a FWF database built atop HBase – the distributed database system running on Hadoop commodity clusters. By combining a 6-dimensional (6D) Hilbert spatial code and a temporal index into a compound indexing key, the database system is capable of supporting multiple spatial, temporal, and spatio-temporal queries. Such queries are important for FWF data exploration and dissemination. The proposed spatial decomposition at a fine resolution of 0.05m allows the storage of each LiDAR FWF measurement (i.e. pulse, waves, and points) on a single row of the database, thereby providing the full capabilities to add, modify, and remove each measurement record anatomically. While the feasibility and capabilities of the 6D Hilbert solution are evident, the Hilbert decomposition is not due to the complications from the combination of the data's high dimensionality, fine resolution, and large spatial extent. These factors lead to a complex set of both attractive attributes and limitation in the proposed solution, which are described in this paper based on experimental tests using a 1.1 billion pulse LiDAR scan of a portion of Dublin, Ireland.

1. INTRODUCTION

Laser scanning or Light Detection And Ranging (LiDAR) is the technology that uses a laser to collect three-dimensional (3D) geometric data. The most common type of LiDAR uses a pulsed laser to measures the range to a physical object. This is based on the time of flight of the laser pulse reaching the object and returning. The ranger is integrated with a scanning mechanism to move the laser beam to provide complete spatial coverage of objects in the scene. The typical data output is a point cloud (Fig. 1) – which is a collection of discrete sampling points of the measured geometries, as represented by x-, y-, z- tuples and possible additional attributes, most typically intensity as a measure of the energy backscatter. While this is what is most commonly delivered to the end user, these discrete points are not the direct sensing data but are, instead, derived from the integration of the raw ranging data with the position and the orientation of the scanning platform. Traditionally, the raw data have not been available to the end users and the process deriving the point clouds from the raw data has been proprietary to each LiDAR sensor manufacturer.

However, in recent years, there has been a growing interest in exploiting the raw ranging data, which have the format of a time-series of optical signal amplitudes as recorded by the sensor. The time-series data are usually digitized at a temporal resolution of several nanoseconds and are called full waveform (FWF) data.

The most dominant use of FWF LiDAR data has been in forestry for surveys, in which the FWF mode often surpasses the discrete return mode at capturing the structures under tree canopies (Fieber et al., 2013). To a lesser degree, non-forestry applications

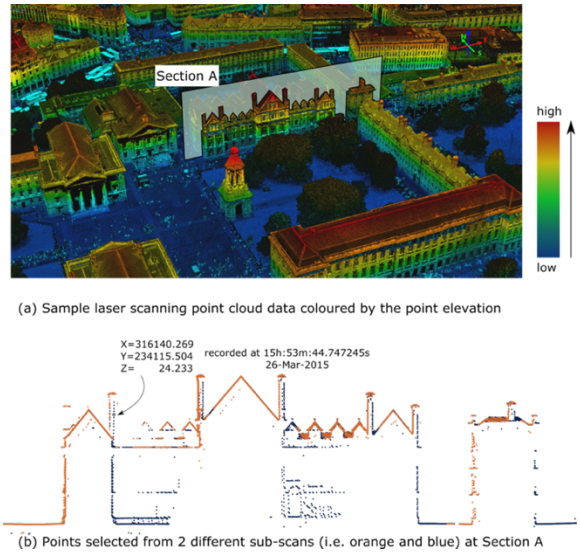

Figure 1. Sample discrete point LiDAR data

* Corresponding author

have also been investigated (Mallet et al., 2011). With the potential usefulness of FWF LiDAR being increasingly recognized, major LiDAR manufacturers (e.g. Riegl and Optech) are including direct export options for the FWF data in their latest aerial and terrestrial product lines. To help support this, standard LiDAR data formats [e.g. ASPRS' LAS (ASPRS, 2011)] have been extended to enable FWF data storage and exchange [e.g. PulseWaves (Isenburg, 2014), Sorted Pulse Data (Bunting et al., 2013)]. Despite these notable effort, FWF data indexing and retrieval remain largely neglected research topics.

Approaching the problem from a different perspective, this paper reasons that enabling FWF data retrieval by spatial and temporal conditions is important to foster its complete exploitation including visualization. Visualization queries typically involve only a selected subset of the data, thereby necessitating a spatial and/or temporal query mechanism. Data retrieval speed is crucial when interactive visualization is expected and requires a spatial index to provide the need data access speed for the data sets that are readily becoming available.

Given that perspective, this paper presents the development of a FWF data management system capable of supporting data access across space and time. The primary type of query investigated in the paper is searching data from laser beams that pass/intersect a predefined spatial extent. The query can be constrained to a single flight line, or all flight lines can be included. Fine resolution temporal queries are also supported, even though the current implementation is not optimized for this query. A temporal filter can be combined with a flight line restriction (i.e. purely temporal queries) or with a spatial condition (i.e. spatio-temporal queries) to facilitate different possible data retrieval needs.

The FWF database is created atop HBase – the non-relational (i.e. NoSQL), distributed database software in the Hadoop ecosystem. HBase is a replication of Google's BigTable (Chang et al., 2006), which is a system built for random access to datasets at a petabyte scale and can operate on a group of networked commodity servers (i.e. commodity cluster). In addition to the extreme scalability, HBase is inherently parallel. As a result, the database system is expected to provide reasonable performance. The database shares many features with column-oriented database systems. Specifically, the stored data do not have to conform to a rigid schema, as occurs in relational database systems.

2. BACKGROUND

To facilitate the remaining of the paper, this section provides some background about computer modeling of FWF LiDAR data and an introduction to HBase – the distributed database software used in the paper.

2.1 Full waveform data modelling

The main difference between FWF LiDAR data compared to the typical point cloud format is the time-series, waveform component. In addition to the waveforms, FWF datasets often contain the orientation and position data of the laser beams associated with the waveform segments called the pulse. All of these elements, including the discrete point data, are inter-related. In this paper, the data modeling approach of the PulseWaves file format developed by Isenburg (2014) is extended to model the FWF data components and their relationships. The model considers a FWF dataset as composed of 3 inter-related components: (1) pulse; (2) wave; and (3) point.

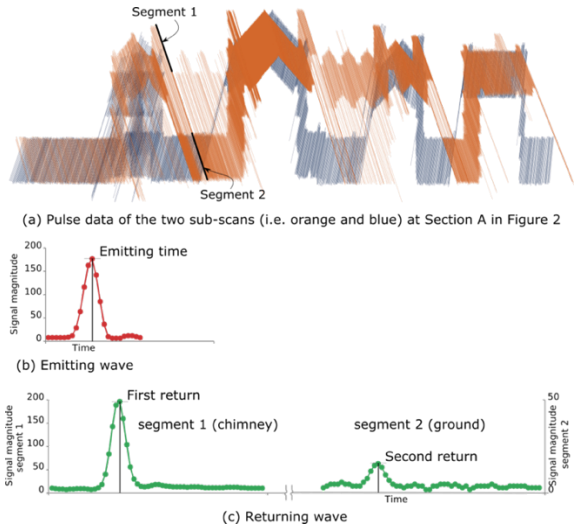

Figure 2. Sample full waveform data

A "pulse" is a line segment representing the location and orientation of a laser beam (Fig. 2a). The pulse encapsulates information about the scanning process such as the location of the aircraft at the time of each laser measurement, the locations of the first and last recorded samples, and some metadata about the laser measurement (e.g. timestamp). In contrast, a "wave" in PulseWaves' terms is a series of discrete signal magnitude values recorded on the sensor (Fig. 2b&c). Each laser pulse emitted can result in several wave samplings. There are typically at least two wave samplings for each laser pulse: one from the outbound sensor that records the emitting signal (Fig. 2b) and one from the inbound sensor that captures the reflected signal (Fig. 2c). However, there can be more than one returning wave sampling per laser pulse and sometimes none return.

Each sampling is decomposed further into segments. Each segment is a continuous recording. Fig. 2 illustrates an example scenario where a laser pulse hits multiple separate objects such as a portion of a building chimney and part of the ground surface (i.e. Segment 1 and Segment 2 in Fig. 2a). In that case, the sensor receives peak in the returning signal from the chimney so the first recording is invoked (Segment 1). The activation of the second recording (Segment 2) corresponds to the moment the laser reaches to the ground surface. Fig. 2a shows the two recording segments spatially, as two collinear line segments. The waveforms of the two segments are shown in Fig. 2c. The last component of a FWF dataset is the discrete point dataset, which is the most common output of laser scanning. The point set is not a separate piece of information, but instead a derivation of the processing of the pulse and waves data and is representative of the peak intensity response in the wave data.

2.2 Distributed databases

HBase is an open-sourced replica of Google's BigTable (Chang et al., 2006), which equips the Hadoop distributed file system with the random data access capability. Since the data are distributed, HBase databases are inherently highly parallelized. Thus, data retrieval is very efficient. Compared to traditional relational database management systems (RDBMS), HBase provides much higher flexibility, as it does not require a rigid data schema or even data types. Notably, all HBase data are internally maintained in a universal binary form and are only decoded when read off the database. Thus, database designers and users have complete control of the data encoding. Consequently, the data do

not have to conform to a pre-defined data type as normally seen in relational databases.

At the lower level, HBase data are structured as a large, sorted, multi-dimensional map, which can be expressed programmatically as in Figure 3 (George, 2011). According to that data structure, an HBase table is a sorted map (1) of pairs of RowKey (2) and List (3). Each element of List (3) is called a column family in HBase. A RowKey is a user-defined, unique identifier of each row in the HBase table. Notably, the RowKey plays an important role in HBase indexing, as it is the primary key for sorting and also distributing the data. As a result, deciding upon the RowKey design is of utmost importance in HBase table design, as will be demonstrated in the latter part of this paper. Each column family [i.e. SortedMap (4)] is composed of pairs of the table column (5) and a list (6) of table value and timestamp pairs [i.e. (6) and (8)]. The value is the actual data content stored in the table, while the timestamp denotes the creation time of the content. The timestamp allows storage of multiple versions of the content in HBase. The data structure of an HBase table is sometimes viewed at a higher level as a collection of key-value pairs, in which a key is composed of a row-key, a column family name, a column name, and a timestamp. The value is the actual datum.

```
SortedMap<RowKey, List<SortedMap<Column, List<Value, Timestamp>>>>
   (1)      (2)      (3)      (4)      (5)      (6)      (7)      (8)
```

(a) Low-level data storage structure in HBase

```
(Table, RowKey, Family, Column, Timestamp) → Value
```

(b) A high-level view of HBase data structure

Figure 3. Logical data structure of HBase

Despite all of its favorable characteristics, HBase is not a replacement for a traditional relational database. While aiming for higher scalability and greater flexibility, the HBase design (as with most other non-relational database systems) loosens parts of the relational features such as the compliance to Codd's 12 rules (Codd, 1985) and the guarantees against transaction validity (a.k.a. ACID: Atomicity, Consistency, Isolation, and Durability) – the traditional, widely-adopted RDBMS standards (George, 2011). Even though these trade-offs are not acceptable in domains such as banking and medical databases, they are not fatally problematic in many applications such as web searching or LiDAR data visualization. The decision between an RDBMS and a more relaxed non-relational option must be based on a rigorous justification of the features of the candidate technologies with respect to the specifics of the data and the affiliated storage and retrieval demands.

3. RELATED WORKS

While several data exchange formats have been developed to accommodate FWF LiDAR data [e.g. ASPRS' LAS (ASPRS, 2011), PulseWaves (Isenburg, 2014)], there are few efforts in data management systems for FWF data. The Sorted Pulse Data system by (Bunting et al., 2013) and the authors' own previous development of a FWF data extension for the Oracle DBMS (Laefer et al., 2018) are the only established works known to the authors.

The Sorted Pulse Data (SPD) approach by Bunting et al., (2013) offers two options for organizing pulse data: (1) non-sequential SPD index and (2) sequential SPD index. Using the non-sequential option, nearby pulses can be distributed over various locations on the storage device. In contrast, the pulses are physically sorted so that pulses in close spatial proximity to each other are stored near other on a computer disk in the sequential SPD option. The sorting requires additional data processing steps but can provide better querying performance. The spatial indexing approach in SPD is a regular, 2D grid based on either the first or the last returns of the laser pulses (Bunting et al. 2013). In other words, that work degrades a line-indexing problem to that of a point-indexing problem. Within that context, the spatial indices (maintained as 2D arrays within the HDF5 structure) facilitate direct spatial access to grid cells relevant to a spatial query. This simple indexing mechanism constrains the type of spatial access to selection of the indexing point (i.e. the first or the last sample). For example, if the first sample is chosen, then only querying on the first echo is supported.

In an attempt to overcome those limitations and enhance the capability of spatial and temporal queries on FWF LiDAR data, Laefer et al. (2018) developed a hybrid data indexing system atop an Oracle database. The spatio-temporal index comprises a Hilbert coded, 2D grid at the top level, and a bottom layer with multiple 3D, regional octrees. These can be transferred between the computer's disk and the main memory via a serialization mechanism. The temporal elements are integrated in the indexing structure at two levels: flight line level and timestamp level (at a microsecond resolution). Unlike what is supported in SPD, the spatial indexing structure by Laefer et al. (2018) wholly preserves the data by not degrading the index and the queries to the point level. Specifically, the laser pulse data were modelled as line segments, and queries were performed directly on the line segment data. The scalability of the system was successfully evaluated on a real-life, dense dataset of 1.1 billion pulses. However, the size of the current index was relatively large (i.e. 20 bytes/pulse). In addition, there was a certain level of redundancy in the octree structure, since a laser pulse can spread over multiple octree nodes and is, thus, indexed multiple times.

There also exists a number of line indexing methods in the literature which are not dedicated to FWF LiDAR data [e.g. (Bertino et al., 1998; Hoelt and Samet, 1992; Jagadish, 1990; Kolovson and Stonebraker, 1991)]. Unfortunately, none of the methods known to the authors are truly usable for the type of line segment data investigated in the paper. Many of the cited methods were built for two-dimensional data such as those of a cadastral database or a circuit board's model. For instance, the approaches proposed by Bertino et al., (1998) and Jagadish, (1990) are not naturally extensible to 3D. In addition to the dimensionality mismatch, the density and spatial distribution of the data addressed in the previous research differ significantly from those of the FWF LiDAR data. For example, the multi-dimensional interval method presented by Kolovson and Stonebraker, (1991) would be inefficient for FWF data, because of the huge level of overlapping between the laser pulses if they were represented as intervals (Fig. 2a). Some of the methods rely on certain assumptions about the data such as non-crossing lines (e.g. Bertino et al., 1998). Such an assumption totally precludes the method from being utilized for FWF data indexing. Amongst the cited approaches, the region quadtree solution introduced by Hoelt and Samet (1992) has the most promising potential in solving the FWF data indexing problem. A 3D version of the method is actually used in the authors' previous FWF data management system (Laefer et al. 2018). However, the tree structure does not well suit the key-value architecture of HBase, thus requiring some modifications to be adopted for FWF data.

The shortage in spatio-temporal solutions for FWF data management and the unsuitability of existing line indexing methods motivated the development of the line indexing solution established in the remaining sections of this paper. The research

presented in this paper extends the previous RDBMS solution by Laefer et al. (2018) to one in a distributed computing environment using a novel indexing design, and thus introducing the first distributed computing solution for FWF data.

4. LINE SEGMENT DATABASE INDEXING FOR INTERSECTION QUERIES

The primary type of query considered in the paper is the search for laser beams that intersect a 3D polygonal geometry. Even though the querying result can include any combination of the pulse, wave, and point datasets, the element that is essentially relevant to the query processing is the pulse portion of the data, which has the form of a line segment. In other words, a 3D line segment database needs to be searched to retrieve entities that intersect the queried geometry. In this section, the spatial index established to enable the primary querying type is described. Essentially, the index is constructed by transforming the spatial query into a multi-dimensional range search so that it can be later implemented in the key-value data store.

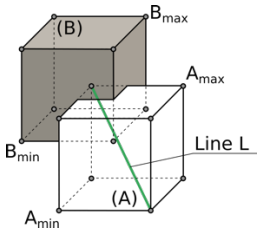

Figure 4. Box intersection

The formulation starts with a system of linear inequalities (Eqn. 1) that defines the intersection relationship between 2 rectilinear boxes (i.e. box A and box B in Fig. 4). The minimum vertices [x_{min}, y_{min}, z_{min}] and maximum vertices [x_{max}, y_{max}, z_{max}] of both of the boxes are needed to identify the spatial relationship.

$$\begin{cases} A.x_{min} \le B.x_{max} \\ A.y_{min} \le B.y_{max} \\ A.z_{min} \le B.z_{max} \\ A.x_{max} \ge B.x_{min} \\ A.y_{max} \ge B.y_{min} \\ A.z_{max} \ge B.z_{min} \end{cases} \quad (1)$$

Eqn. 1 can also be used as a requisite for a line segment L bounded by the box A to intersect the box B (Fig. 4). Stated formally, a line L intersects a box B only, if the bounding box A of L intersects B. Thus, Equation 1 holds true for the coordinates of the line segments. After substituting the box vertices by the line vertices and constraining the ranges by some prior knowledge, Eqn. 1 can be rewritten as in Eqn. 2; where H_{max} and V_{max} are the maximum horizontal and vertical extents of the pulses - which can be computed given the knowledge of the data acquisition process (i.e. pulse length, scan angle). Those constants are computed using Eqn. 3; where ρ_{max} and θ_{max} are the maximum pulse length and the maximum scanning angle.

According to Eqn. 2, the coordinates of the pulses' bounding boxes [$L.x_{min}$, $L.y_{min}$, $L.z_{min}$, $L.x_{max}$, $L.y_{max}$, $L.z_{max}$] can be used to index the pulse data for the spatial intersection query. The role of the index is to rapidly filter and exclude line segments that fail any of the inequalities in Eqn. 2. Those pulses can be immediately excluded from the querying result. Only the pulses that pass all six inequalities need further processing to determine whether or not they intersect the queried geometry.

$$\begin{cases} L.x_{min} \in (B.x_{min} - H_{max}, B.x_{max}) \\ L.y_{min} \in (B.y_{min} - H_{max}, B.y_{max}) \\ L.z_{min} \in (B.z_{min} - V_{max}, B.z_{max}) \\ L.x_{max} \in (B.x_{min}, B.x_{max} + H_{max}) \\ L.y_{max} \in (B.y_{min}, B.y_{max} + H_{max}) \\ L.z_{max} \in (B.z_{min}, B.z_{max} + V_{max}) \end{cases} \quad (2)$$

$$\begin{cases} H_{max} = \rho_{max} \\ V_{max} = \rho_{max}.sin(\theta_{max}) \end{cases} \quad (3)$$

5. IMPLEMENTATION

This section presents the implementation details of the spatial index established in Section 4 atop an HBase architecture and the needed extensions include the temporal queries.

5.1 Row-key design

Key-value data stores, such as the HBase database used in this research, are optimized for data retrieval by a key or by a range of consecutive keys [i.e. one-dimensional (1D) range query]. The keys can be alphanumeric or arbitrary binary strings, as long as they can be sorted by an order (e.g. lexicographical order). Key-value data storage systems have previously been transformed to facilitate multi-dimensional range searches via space filling curves (Nishimura, 2011; Whitby et al., 2017). A space filling curve [e.g. Morton curve, Hilbert curve] maps data from a high-dimensional space onto a 1D space. The mapping allows data indexes and queries in the high-dimensional space to be transformed into the counterparts in a 1D space so that they can suit the key-value architecture of the database. The ability to preserve the spatial proximity of the original space during the mapping is a desirable feature, as it represents the quality of a space filling curve. Starting from the multi-dimensional range search derived in Section 4, this paper uses a 6-dimensional Hilbert curve to transform each tuple of ($L.x_{min}$, $L.y_{min}$, $L.z_{min}$, $L.x_{max}$, $L.y_{max}$, $L.z_{max}$)] to a binary Hilbert code. The Hilbert transformation in this paper is facilitated by the Java libraries by Aioanei, (2008) and Whitby et al., (2017). The spatial resolution of the Hilbert space is selected at a sufficiently fine level (e.g. 5 cm) so that the Hilbert codes are unique for each pulse.

The Hilbert code is prefixed with a flight line ID; a unique identifier corresponding for the given pulse (Fig. 5a). Having the flight ID as the first component of the row-key allows restriction of the querying results to a single flight line. When data from all available flight lines are desired, the query resolver invokes multiple searches across all relevant flight lines. The design allows searches to be handled concurrently by HBase, thereby maximizing parallelism capabilities of the distributed system. The list of flight IDs is obtained by querying a secondary table, herein called the coarse indexing table, which provides a light-weight spatial index for the flight lines. Each row of the table represents a 5m×5m cell (C) within the spatial coverage of the dataset. The row is indexed by a 4D Hilbert code similar to the 6D Hilbert code used for the main table (Section 4) but encapsulates only the x and y coordinates of the cell [($C.x_{min}$, $C.x_{max}$, $C.y_{min}$, $C.y_{max}$)] (Fig. 5b).

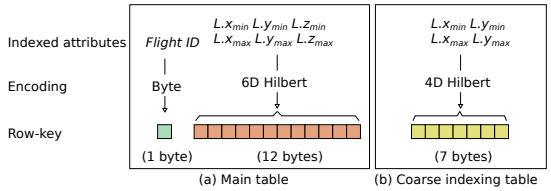

Figure 5. Row-key structures

The aggregation of multiple components into the content of a row-key for the main table is relatively common in key-value data stores. The approach is often called the compound key method (George, 2011). Given the FWF LiDAR data stored in HBase and indexed by the compound row-key, the pulse intersection query can be resolved by firstly decomposing the given querying window to a series of Hilbert segments in the same 6D space, which is used to construct the index. The relevant flight IDs retrieved from the coarse indexing table are then added to the minimum and maximum values of each Hilbert range. The resulting ranges are used to query directly the row-key of the HBase database for intersecting candidates. After being read off disks, the candidate pulse records continue to be filtered by their bounding boxes and then by their actual geometries. This is done to identify those pulses actually intersecting the queried geometry. The additional filtering layers can be skipped to reduce the querying time at the cost of including false positive results in the resulting set. Another technique which can be used to reduce the querying time is to ignore the small gaps between the Hilbert segments to reduce the number of segments (Whitby et al., 2017). The technique is effective, as the querying time is often governed by the number of calls invoked against the database, which is equivalent to the number of Hilbert segments. The downside of this technique is that it leaves a larger number of false positive records for subsequent data filtering layers.

5.2 Custom temporal filter

While the compound row-key design in Section 5.1 can support the primary intersection query with an option of adding flight line constraints, the key does not allow actual temporal queries (i.e. search for pulses collected within a temporal range). In the current implementation, the temporal queries are supported via a custom filter In HBase, a custom filter is a mechanism that allows interim querying results to be filtered on the server-side before returning results to the client requesting the data. Using a filter is not as efficient as an index, since every data candidate passed to the filter needs to be checked. However, having the filter implemented on the server-side as in HBase is significantly better than filtering the data on the client side. This is because the former solution requires less data to be transferred over the network, which is often the main bottle neck in a computing system. In addition, there is often more computing power on the server side to facilitate demanding operations such as spatial checks. In this paper, the temporal constraint is processed by partially processing the interim querying results, which are derived from the search by the row-key, to read the timestamp of each laser pulse. Prior to being returned to the querying client, the FWF data entries are checked against the specified temporal range to remove unqualified candidates.

5.3 Column structure

The row-key design presented in Section 5.1 designates one row in the database for all data available from a laser measurement. The data include the pulse (in the form of a line segment) and several waveform segments (time-series data). The pulse has a fixed length, while the number of waveform segments and the waveform’s length are variable. The examples in Fig. 6 illustrate the complex structure of the FWF data; Pulse A has one outgoing wave segment, one returning wave segment corresponding with one discrete return, while the number of returning wave segments (i.e. also the number of discrete returns) of Pulse B is one. For such diverse, non-tabular data, the schema-less feature of HBase is beneficial. Since there is not an anticipation that any compositions of the data components are more frequently queried together, all the data are stored in one single column family (i.e. column family ‘s’ in Fig. 7a). Depending on the expected data usage pattern, the design can be changed so that only the highly related components are grouped into the same column family to optimize the data retrieval performance.

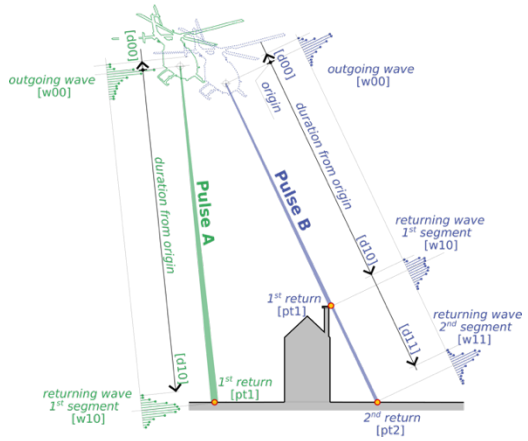

Figure 6. Examples of single segment and multiple segment LiDAR waveforms

The complex structure of the FWF data is modeled using the column structure in HBase. This consists of one pulse column (named “pls”) per measurement. The actual datum of the pulse is encoded as a binary string closely following the format used in the PulseWaves format (Isenburg 2014). For the waveform data, each wave segment requires 2 columns. The first (named “dij”) stores the temporal offset from the moment the laser pulse leaves the emitter to the time corresponds to the first echo of the wave segment. The terms i and j are the indices of the sampling and the segment, which are encoded directly in the column names. The second column type (named “wij”) is for the waveform’s echo values. That encoding allows retrieval of the raw waveforms by the sampling and segment indices. By combining waveforms with the associating duration value and metadata of the laser beam encoded in the pulse column, each waveform echo can be registered into a 3D space similar to what is typically done with the discrete return point. The separation of the duration value in a separate column allows the pulse geometry to be rapidly geo-referenced without registering all waveform echoes in the 3D space. Notably, the rows in the main table need not conform to the same column structure thanks to the schema-less feature of HBase. Also, empty column values do not occupy storage space.

The column structure of the coarse indexing table is illustrated in Fig. 7b. The table contains one column family (i.e. ‘f’). The flight line identification numbers in the integer format are used as the column name. If a rectilinear cell (i.e. a row) contains data of a flight line, a 1-byte dummy value is written to the HBase cell corresponding to the row and the column. Otherwise, the cell is left empty, thereby not consuming any storage space.

	Column family: *s*							
Row-key:	**Col:** *pls*	**Col:** *d00*	**Col:** *w00*	**Col:** *d00*	**Col:** *w00*	**Col:** *d10*	**Col:** *w11*	…
…								
…								

(a) Main table

	Column family: *f*			
Row-key:	**Col:** *0*	**Col:** *1*	**Col:** *2*	…
…				
…				

(b) Coarse indexing table

Figure 7. Table schema

6. PERFORMANCE EVALUATION

6.1 Study data

To aid in evaluating the proposed FWF data storage models, a 2015 FWF ALS data set over a 2 km^2 area of the city center of Dublin, Ireland was employed (Laefer et al., 2017). The data acquisition was conducted in March 2015, using a Riegl Q680i system with full waveform digitization. The ALS flight followed the approach proposed by Hinks et al., (2009) to maximize data coverage on building façades. Specifically, the scan was conducted at the lowest permissible altitude (approximately 300m) with scanning angles ranging from -30° to 30°. The flight paths were oriented at 45° to the city's major street axes to minimize self-shadowing effects. The flight paths within each parallel set were spaced at 100m intervals so that each location in the scanned area was measured up to 6 times and from multiple angles. The original FWF data were available in 2 formats: APSRS' LAS with external waveform data packets and Rapidlasso's PulseWaves. The total file sizes are 217.5GB for the LAS format and 172.5GB for PusleWaves format. In this paper, the dataset in PulseWaves format was used as the input. In order to evaluate the database's scalability, the data ingestion and querying speed were evaluated in two subsets of varying sizes (i.e. Small and Medium in Fig. 8a) in addition to the original PulseWaves dataset (i.e. Large).

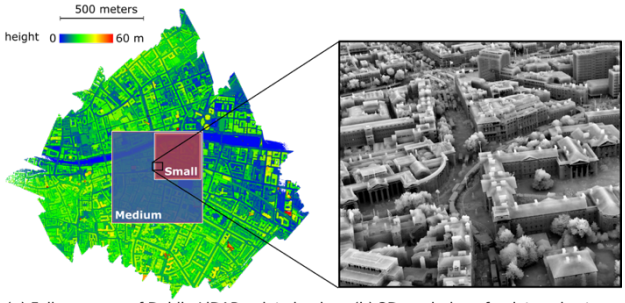

Figure 8. 2015 Dublin dataset

6.2 Data ingestion

Table 1 shows the disk consumption and the data ingestion time for the three datasets when loaded into the HBase structure described in Section 5. The data in the HBase database were 20% more compact than the input PulseWaves files, even though the HBase disk consumption includes the indexing structures, while the original PulseWaves data are not indexed. The compactness of the HBase solution is attributable to the use of the Snappy compression in the database (Google, 2018). Compared to the authors' previous FWF database (Laefer et al., 2018), ingesting data into the HBase data model was 2.5 times faster. The disk consumption and ingestion time for the 3 datasets are relatively constant, thereby demonstrating the scalability of the design.

Dataset	Number of pulses	Disk consumption	Data ingestion time
		bytes/pulse	pulses/s
small	78,618,244	138.2	105,349
medium	316,454,155	137.8	112,072
large	1,010,310,084	138.2	94,173

Table 1. Data ingestion costs

6.3 Data querying

As described in Section 5, the pulse intersection query presented in this paper is processed in multiple steps: (1) obtain the flight IDs from the coarse, 4D indexing table; (2) decompose the querying windows to a set of 6D Hilbert segments; (3) retrieve data from the main HBase table using the RowKey derived from the previous steps. The following subsections analyze the runtime of each individual step.

6.3.1 Data retrieval from HBase

Retrieving FWF data from the main HBase table is arguably the most important and time-consuming step of the data querying process. During this step, the physical locations of the candidate data records are identified based on their RowKeys, the relevant data segments are read off the disks, and further filtering is applied to decide the data records to be returned. Fig.s 9&10 present the data querying speed for two different cases: small and large querying window. The querying windows are constructed from a set of 200 data points randomly selected from the Small dataset (Fig. 8). Since the Small dataset is subset of the Medium and the Large sets, the 200 samples extracted from the Small dataset are contained in all of the 3 testing datasets. The points are the center of the querying boxes which have the dimensions of [1m×1m×1m] for the small queries and [50m×50m×50m] in the large querying cases.

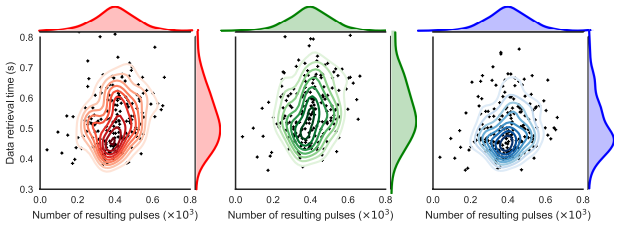

Figure 9. Data retrieval from HBase for small queries (1m×1m×1m)

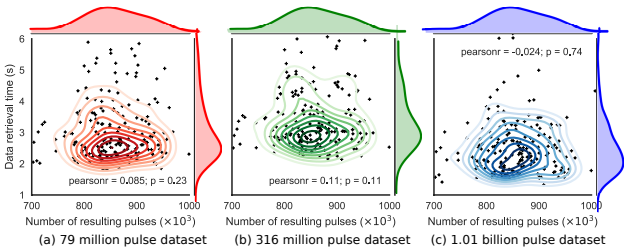

Figure 10. Data retrieval from HBase for large queries (50m×50m×50m)

Each black dot in Fig.s 9&10 represents the time required to retrieve the data for each querying window being tested and the number of pulses intersecting the window. The contour lines and the two histograms are resulted from the Kernel Density Estimation, which summarizes and smoothens the distribution of the testing results. The median data retrieval times are approximately 0.5s and 2.5s for the small and large querying cases, respectively. Even though the speed (2.5s per 0.85 million pulses) does not surpass the authors' previous relational database (Laefer et al., 2018), the speed is sufficiently high to make the database useful for many applications such as data dissemination. In terms of data scalability, there is no performance degradation observed when the data volume increases up to 1 billion pulses.

To further understand the efficiency of the RowKey design presented in Section 4, the numbers of false positives (FP) and the false positive rates of the RowKey filtering step are reported in Table 2. The number of FPs are the number of data records that pass the rowkey filtering layer but ultimately do not satisfy the querying conditions. From the results presented in Table 2, the numbers of FPs (i.e. the number of data records have to be forwarded for further spatial filtering) are approximately 160,000 in the small querying case and 2.5 million in the large querying case. The number of FPs is equivalent to the FP rates of 0.02% for the small queries and 0.26% for the large queries. FP rates are computed as the ratio of the number of FP to the total number of negatives. In other words, the FP rates represents the amount of unqualified data records that the system failed to filter out, with respect to the number of unqualified records successfully filtered by the RowKeys. The relatively small FP rates prove that the RowKey design is reasonably efficient in helping to bypass irrelevant data records.

Query	Number of FPs		FP rate	
	Median	Std. Dev.	Median	Std. Dev.
[1×1×1]	160,212	88,845	0.02%	0.01%
[50×50×50]	2,560,149	1,396,375	0.26%	0.14%

Table 2. False positive rates of the RowKey filtering step

6.3.2 Hilbert decomposition

The Hilbert decomposition transforms a querying window in a high dimensional space to a set of 1D, Hilbert segments. The Hilbert encoding technique is well proven and widely used for managing spatial data, including spatial data handling in distributed, key-value data stores (Dimiduck and Khurama, 2012; Nishimura, 2011; Whitby et al., 2017). The computational cost of the Hilbert decomposition is often inconsiderable compared to the costs of retrieving data from the database. For example, the range decomposition time rarely exceeds 100ms in the authors' previous adoption of 3D Hilbert encoding for point cloud management (Vo et al., 2018). However, this supposedly insignificant step is actually a major bottle neck in the query processing pipeline in the 6D Hilbert solution presented in this paper (Fig. 11).

Compared to the data retrieval time of less than 3s, the Hilbert decomposition time as high as 30s as shown in Fig. 11 are forbiddingly high. In addition, both the size and the location of the querying windows influence the Hilbert decomposition time in an untenable manner. As observed in Fig. 11a, the decomposition cost increases with the size of querying window up to a certain level before the cost drastically drops. These rises and drops occur in a periodic manner when the size of the querying window incrementally increases. There is not a comparable clear pattern in the relationship between the Hilbert decomposition time and the location of the querying window (Fig. 11b). Nevertheless, the decomposition costs appear to vary broadly when the querying window moves. Even though the adverse effects may be specific to the particular Hilbert implementation adopted in the paper (Aioanei, 2008; Whitby et al., 2017), the issue does demonstrate an unfavourable characteristic observed in a high-dimensional application but are unobvious in the lower-dimensional cases.

6.3.3 Other overheads

The majority of the querying time is consumed by the Hilbert decomposition and the HBase data retrieval steps discussed in Section 6.3.2 and 6.3.1. Other overhead including the time required to obtain flight IDs from the coarse indexing table is marginal. More particularly, the overhead is approximately 90ms in the large querying window cases and 20ms in the small querying window cases. The insignificant overhead justifies the usage of the coarse indexing table.

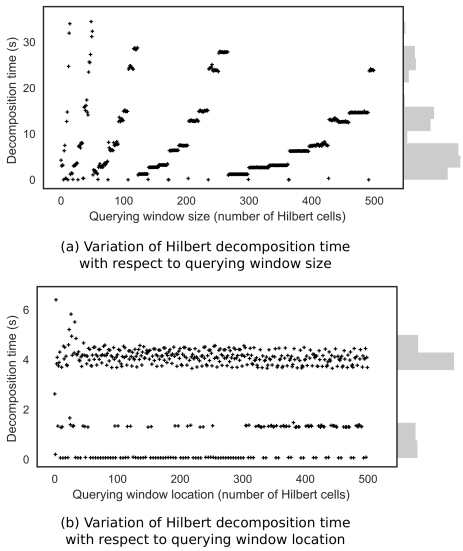

Figure 11. Hilbert decomposition time

7. CONCLUDING REMARKS

This paper presents the first distributed computing based spatio-temporal indexing solution for FWF LiDAR data. The index maps the laser pulses to a 6D Hilbert space that encodes the x, y, and z coordinates of the two ends of the laser pulses. The primary type of query supported by the 6D Hilbert index is the intersection query against a rectilinear bounding box. The Hilbert codes are prefixed by a flight line ID to form a compound RowKey for structuring the data in HBase - a distributed, non-relational, key-value data store. Having the flight ID as the first component of the compound RowKey allows the data to be temporally filtered at the flight line level prior to the spatial evaluation. When data from all flight lines are needed, a light-weight, coarse indexing table is used to identify the flight lines needing to be scanned. The identified flight lines are then processed concurrently by HBase. The solution is successfully implemented. The column structure is designed to exploit the schema-less feature of HBase for accommodating the irregular and complex structure of the FWF data. The developed database is capable of providing a wide range of spatial, temporal, and spatio-temporal queries useful for FWF data exploration. The temporal constraints can be enforced at the flight line level or at the LiDAR timestamp level (i.e. micro-seconds). The fine temporal filtering is provided by a custom filter which processes the data in parallel on the server side. The spatial evaluation can be set at different levels of accuracy depending on users' expectations. At the lowest level, the data are only spatially evaluated by the Hilbert codes. This querying mode is likely to be optimal for querying time. However, some false positive records may remain in the resulting data. Additional filtering layers by bounding boxes and by the line geometries can be added to eliminate the false positive records, but at the cost of increased querying time.

The database implementation was rigorously evaluated using a FWF dataset of 1.1 billion pulses, covering more than 2km^2 of Dublin city. The experiments prove that the design can comfortably scale to accommodate large geo-spatial data sets. Specifically, no performance degradation was observed when the

dataset grew from 79 million to 1.1 billion pulses. Due to the use of a compression mechanism native in HBase and the light-weight design of the index, the total disk consumption of the data and the index in HBase was 20% less than the original FWF data in PulseWaves format. This was done while providing extensive additional functionality. The data ingestion speed of approximately 100,000 pulses/sec is 2.5 times faster than the speed of the previous relational database implementation by Laefer et al. (2018). The rowkey design is justified by the relatively small false positive rate (i.e. 0.02% and 0.26% for the small and large querying cases, respectively) and the reasonable HBase data retrieval (i.e. 0.5s and 2.5s the 2 cases). The HBase data retrieval time is the time required for fetching the data from HBase given that the rowkey ranges are already known. However, the experimental evaluation revealed a significant bottleneck in the query processing pipeline – the Hilbert decomposition – which takes place outside HBase and is independent of the table design. The Hilbert decomposition time was affected by the size and location of the querying window. While the observed issue may be specific to the implementations by Aioanei (2008) and Whitby et al. (2017), which is adopted in the paper, the observation does demonstrate an adverse effect caused by the increase in dimensionality. Notably, the same library has been successfully utilized in lower dimensional spaces, including the use for LiDAR point cloud data management in 3D (Vo et al., 2018; Whitby et al., 2017) without exhibiting this problem.

ACKNOWLEDGEMENTS

The authors would like to thank Professor Peter van Oosterom for the suggestion of using a 6D Hilbert encoding for FWF data management. This work was supported in part through the NYU IT High Performance Computing resources, services, and staff expertise and through the Center for Urban Science + Progress. Additional computing resources used for the presented work was provided by allocation TG-CIE170036 - Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562 (Towns et al., 2014). The dataset was made available via European Research Council grant ERC-2012-StG 20111012 "RETURN - Rethinking Tunnelling in Urban Neighbourhoods" Project.

REFERENCES

Aioanei, D., 2008. Uzaygezen: Multi-dimensional indexing with Hilbert curves. Retrieved from https://opensource.googleblog.com/2008/08/uzaygezen-multi-dimensional-indexing.html

ASPRS., 2011. LAS specification version 1.4 - R13.

Bertino, E., Catania, B., & Shidlovsky, B. (1998). Towards optimal indexing for segment databases. In *Extending Database Technology*, pp. 39–53.

Bunting, P., Armston, J., Lucas, R. M., & Clewley, D., 2013. Sorted pulse data (SPD) library. Part I: A generic file format for LiDAR data from pulsed laser systems in terrestrial environments. *Computers & Geosciences*, *56*, pp. 197–206.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., … Gruber, R. E., 2006. Bigtable: A distributed storage system for structured data. *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, pp. 205–218.

Codd, E. F., 1985. Is your DBMS really relational? Computer World.

Dimiduck, N., & Khurama, A., 2012. Scaling GIS on HBase. In *HBase in Action*. Manning.

Fieber, K. D., Davenport, I. J., Ferryman, J. M., Gurney, R. J., Walker, J. P., & Hacker, J. M., 2013. Analysis of full-waveform LiDAR data for classification of an orange orchard scene. *ISPRS Journal of Photogrammetry and Remote Sensing*, *82*, pp. 63–82.

George, L., 2011. *HBase - The definitive guide* (1st ed.). O'Reilly.

Google., 2018. Snappy, a fast compressor/decompressor. Retrieved from http://google.github.io/snappy/

Hinks, T., Carr, H., & Laefer, D. F., 2009. Flight optimization algorithms for aerial LiDAR capture for urban infrastructure model generation. *Journal of Computing in Civil Engineering*, *23*(6), pp. 330–339.

Hoelt, G., & Samet, H., 1992. A qualitative comparison study of data structures for large line segment databases. In *SIGMOD '92 Proceedings of the 1992 ACM SIGMOD international conference on Management of data*.

Isenburg, M., 2014. PulseWaves: an open, vendor-neutral, stand-alone, LAS-compatible full waveform LiDAR standard. Retrieved from http://rapidlasso.com/pulsewaves/

Jagadish, H., 1990. On indexing line segments. In *Proceedings of the sixteenth international conference on Very large databases*, pp. 614–625. Brisbane, Australia: Morgan Kaufmann.

Kolovson, C. P., & Stonebraker, M., 1991. Segment indexes: dynamic indexing techniques for multi-dimensional interval data. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data - SIGMOD '91*, *20*(2).

Laefer, D., Abuwarda, S., Vo, A.-V., Linh, T.-H., & Hamid, G., 2017. 2015 aerial laser and photogrammetry survey of Dublin city collection record. New York University. DOI: 10.17609/N8MQ0N

Laefer, D. F., Vo, A.-V., & Bertolotto, M., 2018. A spatio-temporal index for aerial full waveform laser scanning data. *ISPRS Journal of Photogrammetry & Remote Sensing*, *138*, pp. 232–251.

Mallet, C., Bretar, F., Roux, M., Soergel, U., & Heipke, C., 2011. Relevance assessment of full-waveform lidar data for urban area classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, *66*(6), pp. S71-S84.

Nishimura, S., 2011. MD-HBase : A scalable multi-dimensional data infrastructure for location aware services. In Proceeding of the 12th IEEE Intl Conf on *Mobile Data Management (MDM)*, vol. 1, pp. 7-16.

Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., … Wilkins-Diehr, N., 2014. XSEDE: accelerating scientific discovery. *Computing in Science and Engineering*, *16*(October), pp. 62–74.

Vo, A. V., Konda, N., Chauhan, N., Aljumaily, H., & Laefer, D. F., 2018. Lessons learned with laser scanning point cloud management in Hadoop HBase. In *Lecture Notes in Computer Science*n, pp. 1–24. Lausanne: Springer.

Whitby, M., Fecher, R., & Bennight, C., 2017. GeoWave: Utilizing distributed key-value stores for multidimensional data. In *Advances in Spatial and Temporal Databases*, pp. 105–122.